AWS CloudFormation Level-Up

Chuck Meyer, Sr. Dev Advocate AWS CloudFormation

Agenda

Intro

Authoring

Deploying

Testing

Demos along the way

Intro

Who I am

Chuck Meyer

cmmeyer@amazon.com
Sr Developer Advocate, AWS CloudFormation

- 5+ years at AWS
- Major: Infrastructure as code and DevOps
- Minor: Security automation / DevSecOps
- 20+ Years in Technology
- Bass player

@chuckm

Infrastructure as code

Declarative or imperative statements describing hardware, software and services and their relationships.

Infrastructure as code

Declarative or imperative statements describing hardware, software and services and their relationships.

```
Resource: MyWebServer
    Class: Server
    Type: ExtraBig
    Ports:
        - 443
```

Infrastructure as code

Declarative or imperative statements describing hardware, software and services and their relationships.

```
Resource: MyWebServer
    Class: Server
    Type: ExtraBig
    Ports:
        - 443
```

```
server_names = [ 'Red', 'Blue', 'Green']
for name in server_names:
    launch_server(name, 'web')
```

Infrastructure as code

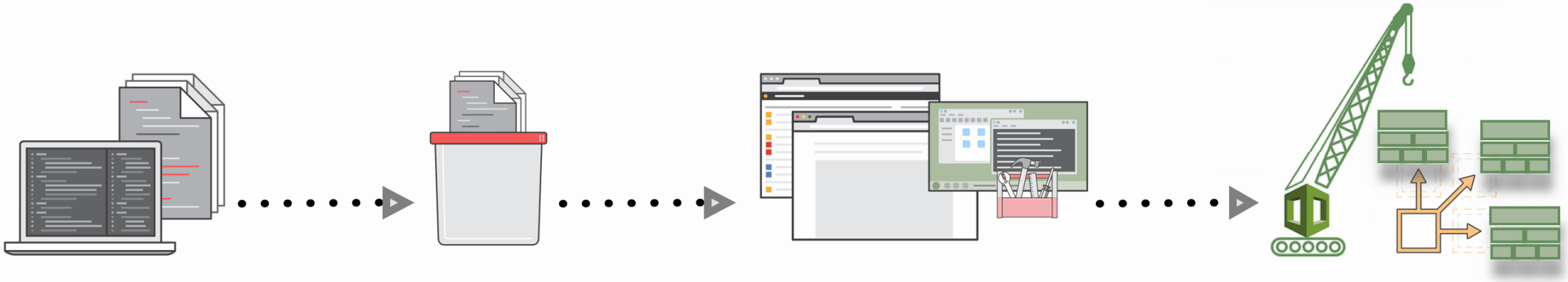
- **Single source** of truth for provisioning and configuration
- Infrastructure that you can **replicate**, **re-deploy**, and **re-purpose**
- Control **versioning** on your infrastructure and your application together
- **Roll back** to the last good state on failures
- Build and deploy your infrastructure through your **CI/CD** pipeline

AWS CloudFormation

- A simplified way to create and manage a collection of AWS resources
- Enables orderly and predictable provisioning and updating of resources
- Fully managed service
- Integrates with the AWS Management Console, the AWS Command Line Interface (CLI), or AWS APIs
- Only pay for the resources you create

CloudFormation at a glance

Enables provisioning and management of your infrastructure as code



Code in YAML or JSON directly or use sample templates

Upload local files or from an S3 bucket

Create stack using console, API or CLI

Stacks and resources are provisioned

Authoring

CloudFormation syntax

JSON

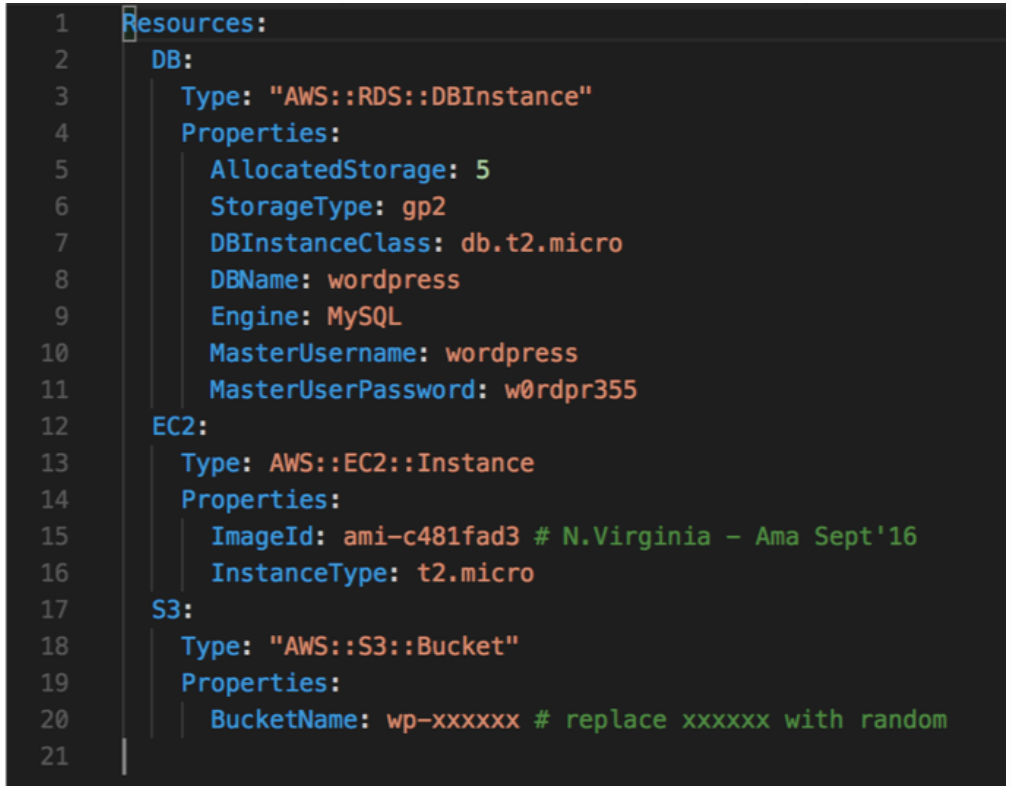
- JavaScript Object Notation
- Attribute/Value pairs
- Similar to XML
- Designed to be machine readable

```
{
"AWSTemplateFormatVersion" : "2010-09-09",
"Description" : "Create a Simple S3 bucket with parameter to choose own bucket name",
"Parameters": {
    "S3NameParam" : {
        "Type": "String",
        "Default" : "saurabh-dafaultbucket",
        "Description" : "Enter the Bucket Name",
        "MinLength" : "5",
        "MaxLength" : "30"
            }
        },

"Resources" : {
    "Bucket" : {
        "Type" : "AWS::S3::Bucket",
            "Properties" : {
                "AccessControl" : "PublicRead",
                "BucketName" : {"Ref" : "S3NameParam" },
                "Tags" : [ {"Key" : "Name" , "Value" : "MyBucket"} ]
                        }
                    }
        },

"Outputs" : {
    "BucketName" : {
        "Description" : "BucketName" ,
        "Value" : { "Ref" : "S3NameParam"}
                }
            }

}
```

CloudFormation syntax

YAML

- YAML ain't a markup language
- **Human readable** data serialization standard
- Comments (use #)
- No `}` or `;`



```yaml
Resources:
  DB:
    Type: "AWS::RDS::DBInstance"
    Properties:
      AllocatedStorage: 5
      StorageType: gp2
      DBInstanceClass: db.t2.micro
      DBName: wordpress
      Engine: MySQL
      MasterUsername: wordpress
      MasterUserPassword: w0rdpr355
  EC2:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-c481fad3 # N.Virginia - Ama Sept'16
      InstanceType: t2.micro
  S3:
    Type: "AWS::S3::Bucket"
    Properties:
      BucketName: wp-xxxxxx # replace xxxxxx with random
```

CloudFormation syntax

Template Anatomy

1. Format version
2. Transforms
3. Description
4. Metadata
5. Parameters
6. Mappings
7. Conditions
8. Resources* (required)
9. Outputs

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Create an EC2 instance running the latest Amazon Linux AMI.
Parameters:
  KeyPair:
    Description: The EC2 Key Pair to allow SSH access to the instance
    Type: String
Resources:
  Ec2Instance:
    Properties:
      ImageId: ami-9d23aeea
      InstanceType: m3.medium
      KeyName: !Ref 'KeyPair'
    Type: AWS::EC2::Instance
Outputs:
  InstanceId:
    Description: The InstanceId of the newly created EC2 instance
    Value: !Ref 'Ec2Instance'
```

Simple template – create EC2 instance

```
Parameters:
  KeyPair:
    Description: 'The EC2 Key Pair to allow SSH access to the instance'
    Type: 'AWS::EC2::KeyPair::KeyName'
```

You enter a value for the `KeyPair` parameter when you create your stack.

Simple template – create EC2 instance

```
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-9d23aeea'
      InstanceType: 'm3.medium'
      KeyName: !Ref 'KeyPair'
```

Includes statically defined properties (`ImageId` and `InstanceType`) and a reference to the `KeyPair` parameter.

`ImageId` is the AMI specific to the region that you want to launch this stack in (eu-west-1 region in this example).

Simple template – create EC2 instance

```
Outputs:
  InstanceId:
    Description: 'The InstanceId of the newly created EC2 instance'
    Value: !Ref 'Ec2Instance'
```

These outputs are returned after the template has completed execution.

CloudFormation syntax – Resources

- The only section of the template that is **required**
- AWS services that will be created, updated, or deleted from your account
- Supports 342 resource types (and growing)

```
Resources:
  Ec2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-9d23aeea'
      InstanceType: 'm3.medium'
      KeyName: !Ref 'KeyPair'
```

CloudFormation syntax – Parameters

- Enable you to input custom values to your template each time you create or update a stack with input validation and restrictions
- Parameter types: `String`, `Number`, `List<Number>`, `CommaDelimitedList`, `Parameter Store` values, and AWS-specific types (`AWS::EC2::Image::Id`, `AWS::Route53::HostedZone::Id`)
- Use the `Ref` and `Fn::Sub` intrinsic functions to reference parameters
- Pseudo-Parameters are predefined by AWS CloudFormation and used just like normal parameters (`AWS::Region`)

Intrinsic functions

Basic programmatic functions available in-line for your declarative templates.

- Retrieve external values (`Ref`, `Fn::Sub`, `Fn::FindInMap`, `Fn::GetAtt`, `Fn::GetAZs`)
- Manipulate strings (`Fn::Sub`, `Fn::Split`, `Fn::Join`, `Fn::Base64`, `Fn::Transform`)
- Conditional logic (`Fn::If`, `Fn::Equals`, `Fn::Not`)

Dynamic references

- Inject values from SSM Parameter Store and Secrets Manager
- KMS encrypted strings
- Versioned and secured by IAM

```
MyIAMUser:
     Type: AWS::IAM::User
     Properties:
       UserName: 'MyUserName’
       LoginProfile:
         Password: '{{resolve:ssm-secure:IAMUserPassword:10}}'
```

CloudFormation syntax - Conditions

Resource creation can depend on logical conditions:

```
Conditions:
  isProd: Fn::Equals [ !Ref EnvType, prod ]
Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Condition: isProd
    Properties:
      ImageId: Fn::FindInMap [RegionMap, !Ref "AWS::Region", AMI]
```

You can use conditions with intrinsic functions (`Fn::If, Fn::Equals, Fn::Not`) to create complex logic for property values.

CloudFormation syntax - Outputs

- Outputs from successful operations
- View them in the console or pass them along as inputs to other stacks
- Used with nested stacks and cross stack references

```
Outputs:
  Environment:
    Description: 'Environment type'
    Value:
      Fn:If: [ isProd, 'Production', 'Development' ]
```

Authoring Level-up

- Use a linter (cfn-python-lint)

Authoring Level-up

- Use a linter (cfn-python-lint)
- Decompose architecture by lifecycle (short vs. long lived)

Authoring Level-up

- Use a linter (cfn-python-lint)
- Decompose architecture by lifecycle (short vs. long lived)
- Isolate stateful resources (databases, caches)

Authoring Level-up

- Use a linter (cfn-python-lint)
- Decompose architecture by lifecycle (short vs. long lived)
- Isolate stateful resources (databases, caches)
- Don't write, **Recycle!**

Authoring Level-up

- Use a linter (cfn-python-lint)
- Decompose architecture by lifecycle (short vs. long lived)
- Isolate stateful resources (databases, caches)
- Don't write, **Recycle!**
  - AWS Quick Starts
  - Documentation examples

Authoring Level-up

- Use a linter (cfn-python-lint)
- Decompose architecture by lifecycle (short vs. long lived)
- Isolate stateful resources (databases, caches)
- Don't write, **Recycle!**
  - AWS Quick Starts
  - Documentation examples
- Use DSLs or the CDK (Troposphere, Sparkleformation, GoFormation)

Deploying

Creating a stack

CloudFormation uses your template as a blueprint to provision resources into a construct called a stack.

On create, the CloudFormation service:

1. Retrieves template from S3 (or API)
2. Parses template and validates parameters
3. Provisions resources in parallel or serial based on dependencies
4. Waits for resources to stabilize
5. Populates outputs and signals completion

-or-

- Rolls back and signals failure

Updating a stack

CloudFormation evaluates your changes to the template or parameters against the last known state of the provisioned resources in the stack. At then modifies the resources as needed.

On update, the CloudFormation service:

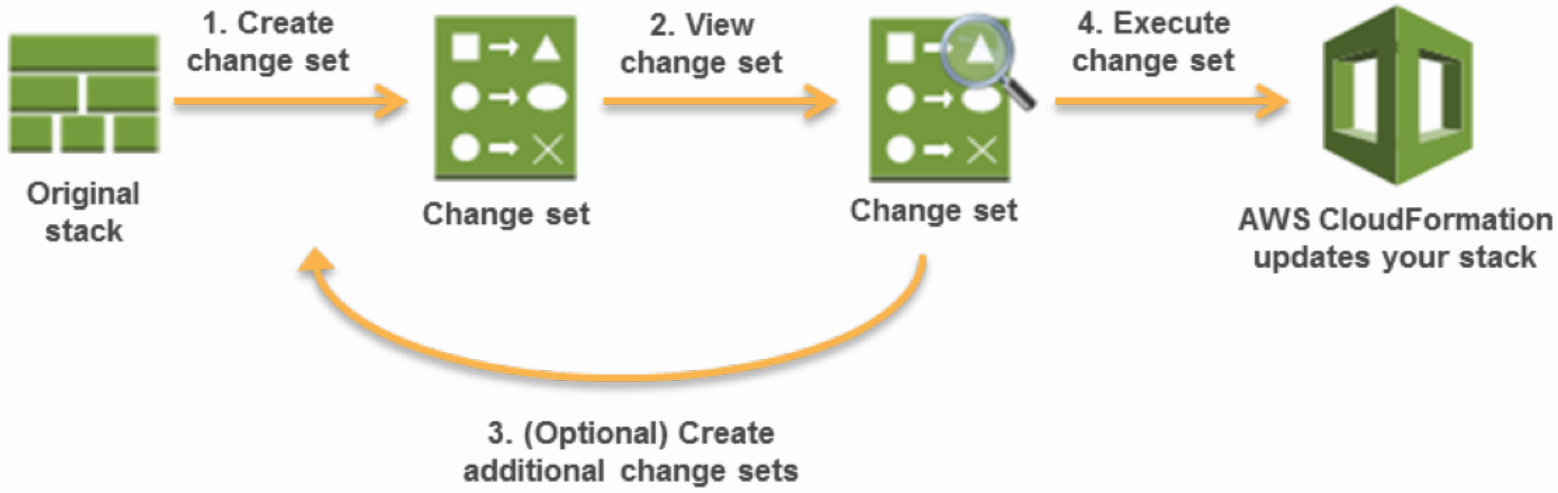
1. Retrieves template from S3 (or API)
2. Compares the template ands parameters to the last known state
3. Changes resources in place or creates new immutable resources
4. Waits for resources to stabilize
5. Updates outputs and signals completion

-or-

- Rolls back and signals failure

CloudFormation change sets

Preview the impact to your stack of changes by comparing the new template and parameters to the last known state of the stack.



CloudFormation makes the changes to your stack only when you decide to execute the change set.

Cross-stack references (Exports)

Allows you to share information between independent stacks.

Export a stack's output values. Other stacks in the same account and region can import the exported values.

```
Network Stack

Outputs:
  VPC
    Description: 'VPC ID'
    Value: !Ref 'VPC'
    Export:
       Name: 'ProdVPC'
```

========>

```
App Stack

Resources:
 myTargetGroup:
   Type: 'AWS::ELBV2::TargetGroup'
   Properties:
      VpcId:
         Fn::ImportValue: 'ProdVPC'
```

Nested stacks

```
Application

Resources:
  NetworkResources:
    Type: 'AWS::CloudFormation::Stack'
  ContainerResources:
    Type: 'AWS::CloudFormation::Stack'
```

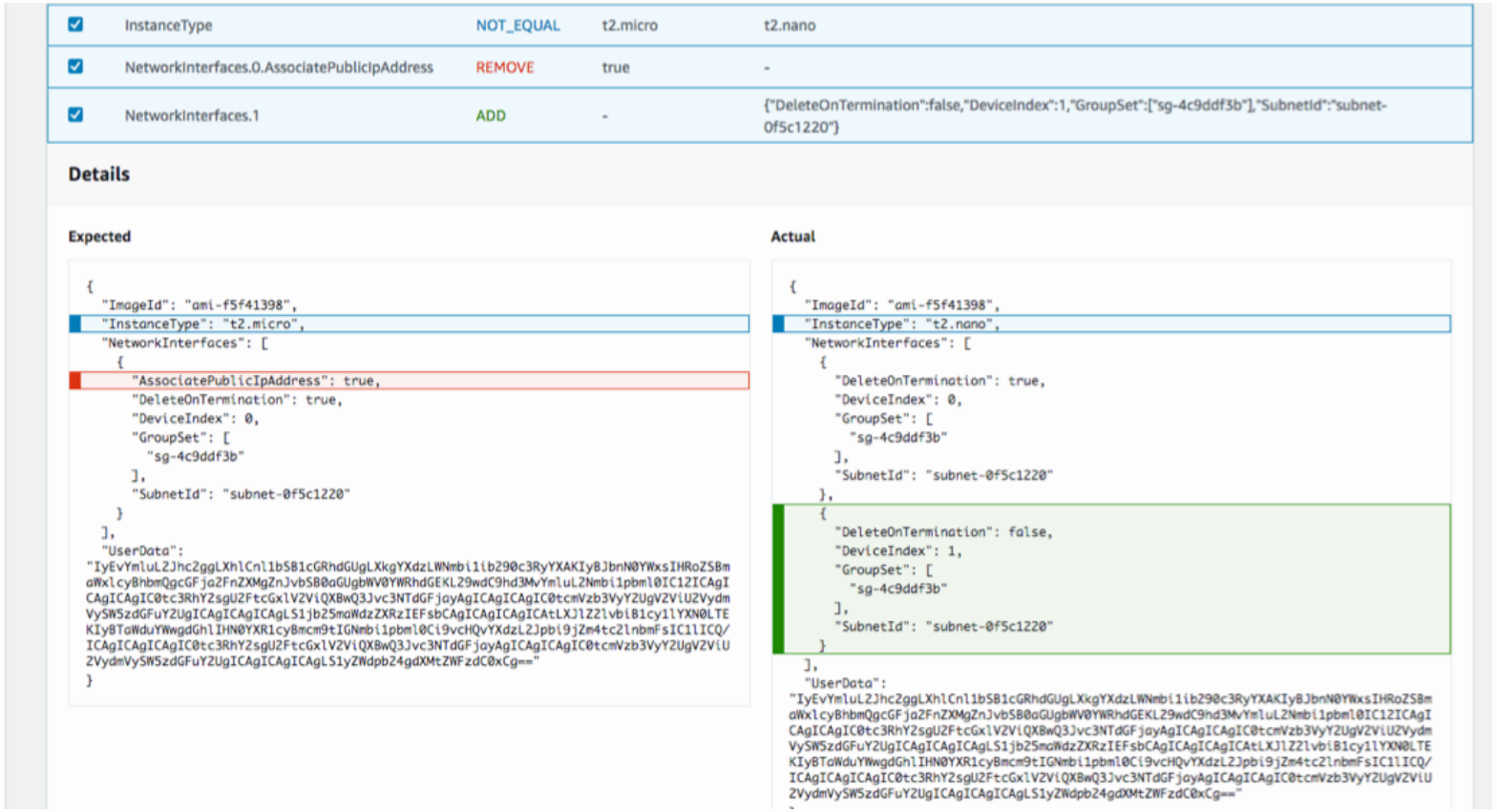
```
Network Resources

Resources:
  MyVPC
      Type: 'AWS::EC2::VPC'
```

- Create a hiearchy of stacks composed of multiple templates.
- Re-use templates with frequently used resources.
- Reference resources across stacks.

Drift detection

Compares the last known state of the stack to current resource configurations.

Shows if configuration changes were made to your stack resources outside of CloudFormation.



CloudFormation StackSets

Create, update, and delete stacks in multiple accounts and regions using a single operation

Deployment Level-up

- Build in guard rails
  - Termination protection
  - Stack policies
  - UpdateReplace and Deletion policies
  - IAM

Deployment Level-up

- Build in guard rails
  - Termination protection
  - Stack policies
  - UpdateReplace and Deletion policies
  - IAM
- Use an orchestration tool to promote environments
  - Jenkins, CodePipeline

Deployment Level-up

- Build in guard rails
  - Termination protection
  - Stack policies
  - UpdateReplace and Deletion policies
  - IAM
- Use an orchestration tool to promote environments
  - Jenkins, CodePipeline
- Use other services to manage configuration
  - SSM Parameter Store or Secrets Manager

Deployment Level-up

- Build in guard rails
  - Termination protection
  - Stack policies
  - UpdateReplace and Deletion policies
  - IAM
- Use an orchestration tool to promote environments
  - Jenkins, CodePipeline
- Use other services to manage configuration
  - SSM Parameter Store or Secrets Manager
- Use changesets whenever possible

Testing

Infrastructure ~~as~~ is code!

- Template code should be in a repo
- Track issues and history
- Commits can trigger test suites and builds
- Use tools and utilities for validation
- Hook into Jenkins, Bamboo, Ansible, Chef, Puppet...

The challenge

> If this is ”infrastructure as code” why are we still testing by continually deploying and fixing failures?

Our goal: Catch errors early to reduce authoring time.

We are living in a golden age of tools

cfn-lint

- Validate AWS CloudFormation yaml/json templates against the AWS CloudFormation spec and additional checks

cfn-nag

- Look for patterns in templates that may indicate insecure infrastructure.

Taskcat

- Catch problems that aren’t obvious in a single template/stack

cfn-lint

> "Can I deploy this template?"

Community-driven open source tool to validate CloudFormation YAML/JSON templates against the CloudFormation resource specification + additional checks.

IDE plugins (VS Code, Atom, Sublime, IntelliJ, vim)

https://github.com/awslabs/cfn-python-lint

```
pip install cfn-lint
```

cfn-nag

> "Should I deploy this template?"

Looks for patterns in CloudFormation templates that may indicate insecure infrastructure.

- IAM rule wildcards
- Security group wildcards
- Access logs that aren't enabled
- Encryption that isn't enabled

https://github.com/stelligent/cfn_nag

```
gem install cfn-nag
```

taskcat

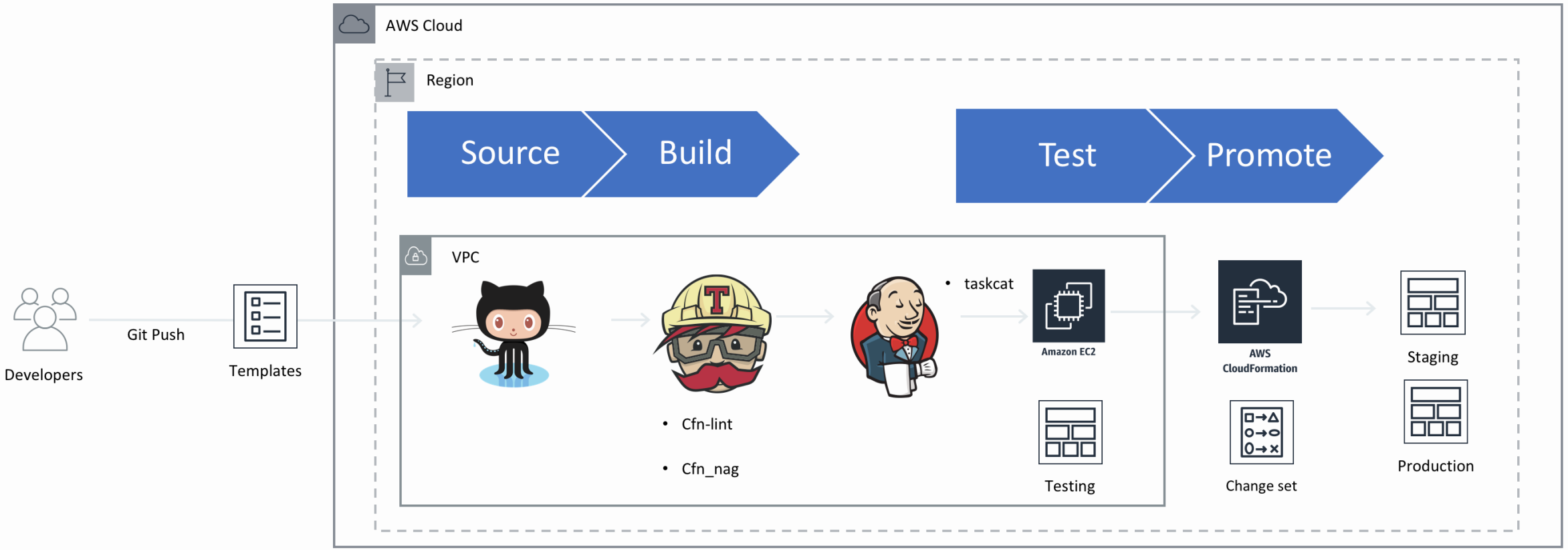
> "Will this template deploy everywhere?"

Catches problems that aren't obvious in a single template/stack.

Tests your templates by creating stacks in multiple AWS regions simultaneously.

Generates a report with a pass/fail grade for each region

Validation pipeline



Testing Level-up

- Static analysis saves time (cfn-lint, cfn_nag)...

Testing Level-up

- Static analysis saves time (cfn-lint, cfn_nag)...
- ...but can't catch everything (taskcat)

Testing Level-up

- Static analysis saves time (cfn-lint, cfn_nag)...
- ...but can't catch everything (taskcat)
- Shorter templates are easier to test and maintain

Testing Level-up

- Static analysis saves time (cfn-lint, cfn_nag)...
- ...but can't catch everything (taskcat)
- Shorter templates are easier to test and maintain
- More reuse == less testing


ASK