at computing

www.atcomputing.nl

eBook

AWS CloudFormation

Table of contents

What is AWS Cloudformation?

AWS CloudFormation provides a common language to describe and provision all infrastructure resources in your cloud environment. CloudFormation allows you to use a simple text file to model and provision, in an automated and secure manner, all resources needed for your applications across all regions and accounts. Therefore this file serves as the single source of truth for your cloud environment.

Enjoy reading this Ebook.

Vincent Lamers, Linux-consultant

AT Computing

Templates, stacks and change sets

Cloudformation Templates

A template is a description of the desired end state of the infrastructure. It can be written in JSON or YAML and contains several sections. The only required section is Resources. That's where you put the resources you're going to use. The following example in YAML consists of two resources (EC2 instance and an elastic IP). It creates a t2.micro EC2 instance with a keypair (testkey) and an additional EBS volume. The image-id refers to the AMI that you need. This is region specific. Finally the elastic IP is assigned to the EC2 instance.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: A sample template
Resources:
  MyEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: "ami-0ff8a91507f77f867"
      InstanceType: t2.micro
      KeyName: testkey
      BlockDeviceMappings:
        -
          DeviceName: /dev/sdm
          Ebs:
            VolumeType: io1
            Iops: 200
            DeleteOnTermination: false
            VolumeSize: 20
  MyEIP:
    Type: AWS::EC2::EIP
    Properties:
      InstanceId: !Ref MyEC2Instance
```

The EIP is attached to the instanceId via Ref. You can use this builtin function to refer to the logical name of another resource in your template. The value that Ref returns depends on the resource type. In general is returns the name of the resource, but that's not always the case. Here you'll find a table that lists the values returned by common resource types.

CloudFormation Stack

In AWS you manage the related resources in a single unit. This is called a stack. A template describes the resources and when CloudFormation executes the template, it creates a stack. You create, update or delete collection of resources by creating, updating and deleting stacks. To prevent unexpected interruptions to the resources in the stack, you can use ChangeSets to review the changes in the template before executing it.

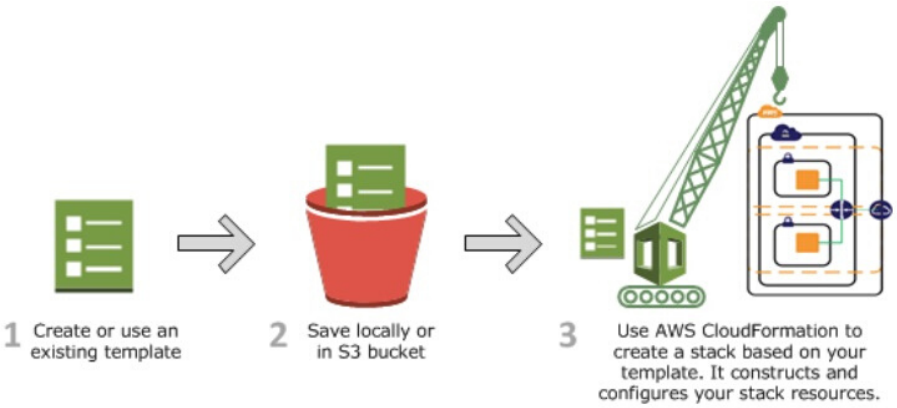
CloudFormation ChangeSet

A ChangeSet will allow you to see how the changes will impact your running resources. Rather then updating the resource, CloudFormation may delete and recreate the resource. This depends on the nature of the change. Renaming a RDS database instance for example will recreate an instance. This definitely will cause downtime to your RDS instance. So prior performing updating on the stack, CloudFormation can create a ChangeSet that provides visibility to the actually changes that would be taken.

Basic work flow of CloudFormation

When you create a stack, AWS CloudFormation makes underlying service calls to AWS to provision and configure your resources. Note, that AWS CloudFormation will only perform actions you are permitted to. For example, to create EC2 instances by using AWS CloudFormation, you need permission to create instances. Likewise, you need permission to terminate instances when you delete stacks with instances. To manage permissions, use AWS Identity and Access Management (IAM).

You declare the calls that AWS CloudFormation makes in your template. Suppose for example, you have a template that describes an EC2 instance with a t1.micro instance type. When you use that template to create a stack, AWS CloudFormation calls the Amazon EC2 create instance API and specifies the instance type as t1.micro. The following diagram summarizes the AWS CloudFormation work flow for creating stacks.



If you specify a template stored locally, an S3 bucket is created by CloudFormation and will be used for each CloudFormation deployment after that. This is done for each region you're working in. Before deploying it, CloudFormation will upload the template to the S3 bucket (in your account). It is also possible to create your own S3 bucket for your templates. In that case you need to specify the location of your S3 bucket before creating or updating your stack.

Designing Templates

CloudFormation Designer

You can write templates locally and upload it to CloudFormation or use the CloudFormation Designer. If you specify a template file stored locally, AWS CloudFormation uploads it to an S3 bucket in your AWS account. AWS CloudFormation creates a bucket for each region in which you upload a template file.

The CloudFormation Designer gives a graphical layout of the template. Basically it is just drag and drop of the resources. Configuring the individual resources is still done by editing the template. Let's just focus on writing templates.

A template consists of several sections. Some of them are optional, like parameters and mapping. The resource section is the only mandatory part of a template.

Resource section

The resources in this section are declared with a logical name, type and a set of properties. We can call this an entity. To declare an entity you can use a fixed set of properties in your template. The Resource type reference describes them in detail, including what to expect when changing one of them (interruption or replacement). Each property has a certain type (a string, Boolean etc.) resulting in a key/value notation. In addition, AWS provides some specific data types. Properties of this type are composed of their own set of properties. BlockDeviceMappings is an example of that.

When using multiple resources that are related to each other (ref), you don't have to worry about ordering. CloudFormation will handle that for you when it executes the template.

Intrinsic functions

AWS CloudFormation provides several built-in functions that help you manage your stacks. Use intrinsic functions in your templates to assign values to properties that are not available until run time. Here is the complete list of the available AWS CloudFormation functions. Basically these functions provides you functionality to use logic in your template which eventually gives you more flexibility.

Pseudo parameters

Pseudo parameters in AWS are like environment variables. They are predefined and can be used in your template. For example, when you need to refer to the region where your resource is creating, there is a pseudo parameter AWS:Region.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: A sample template
Resources:
  MyEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: "ami-0ff8a91507f77f867"
      InstanceType: t2.micro
      AdditionalInfo: !Ref AWS::Region
```

A complete list of the available pseudo parameters can be found here. To give an idea how you can make use of pseudo parameters, here is one example of AWS::NoValue.

This pseudo parameter is used to remove the property of an resource. As seen below , an RDS instance is defined as a resource. In the properties of this resource the if condition says that if “UseDbSnapshot” is “True” , use the “DBSnapshotname” as “DBSnapshotIdentifier” property. Otherwise, set the “DBSnapshotIdentifier” property , it will have “NoValue”.

```
"MyDB" : {
  "Type" : "AWS::RDS::DBInstance",
  "Properties" : {
    ..
    <snippet>
    ..
    "DBSnapshotIdentifier" : {
      "Fn::If" : [
        "UseDBSnapshot",
        {"Ref" : "DBSnapshotName"},
        {"Ref" : "AWS::NoValue"}
      ]
    }
  }
}
```

Mapping section

Mappings are useful to use input values to determine another value. First start with a mapping section in your template.

```
Mappings:
  RegionMap:
    us-east-1:
      "AMI": "ami-0b69ea66ff7391e80"
    us-west-1:
      "AMI": "ami-0245d318c6788de52"
    eu-west-1:
      "AMI": "ami-0ce71448843cb18a1"
```

Here we use the FindInMap function to find a specific value in a map based on the value of a pseudo parameter. FindInMap accepts three parameters. The first is the RegionMap, which is the logical name of the map. Next, we call the pseudo parameter AWS::Region. This is done by the Ref intrinsic function. The last parameter is 'AMI' which is the name of the value pair.

This is a good example of how to use one template across multiple regions in AWS.

In this example the input selection is handled by the CloudFormation via mappings. Sometimes you need to specify input parameters in your template because there is no logic for it. For example, when you want to select the instance type.

Parameters section

Parameters enable you to input custom values to your template each time you create or update a stack.

```
Parameters:
  InstanceTypeParameter:
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - m1.small
      - m1.large
    Description: Enter t2.micro, m1.small, or m1.large. Default is t2.micro.
```

This results in a drop down menu in the cloudformation console. To call the selected value in the template we can use the Ref intrinsic function again.

```
Ec2Instance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType:
      Ref: InstanceTypeParameter
   ..
```

There are different types of input parameters. Above is a simple parameter of the type String with the allowed values. But there are also AWS specific parameter types, for example AWS::EC2::KeyPair::KeyName. This will call the specific resource in AWS to pull the correct values. In this case, the drop down menu presents the key pairs in your account within the region you are deploying a stack.

Another interesting parameter type is the SSM parameter. These types correspond to existing parameters in Systems Manager Parameter Store. You specify a Systems Manager parameter key as the value of the SSM parameter, and AWS CloudFormation fetches the latest value from Parameter Store to use for the stack. You can store data such as passwords, database strings, and license codes as parameter values. You can store values as plain text or encrypted data.

Outputs section

This section declares output values that you can import into other stacks, return in response or view in the console. For example, you can output a custom description to the output section in the CloudFormation console.

```
Outputs:
  OutputVariableName:
    Value: !GetAtt
    - Ec2Instance
    - PublicDnsName
```

Now you'll find in the output section of the Cloudformation console a key value pair of OutputVariableName – dns name of the EC2 instance.

Writing templates can be challenging when you try to deploy multiple resources which are all related to each other. For example, an instance with multiple security groups, a second interface and possibly a role may be attached to it. After deploying your stack from the command line, you'll get feedback in the AWS console. When your deployment fails, the complete stack will be rolled back. The output in the CloudFormation console gives some usable feedback about the possible cause for this. Also you find a lot of information in the CloudFormation reference about resource and property types.

Bootstrapping an EC2 instance with User Data

Bootstrapping an EC2 instance with User Data

The cloud-init package is an open-source application built by Canonical that is used to bootstrap Linux images in a cloud computing environment. EC2 instances contains a customized version of cloud-init. It enables you to specify actions that should happen to your instance at boot time. You can pass desired actions to cloud-init through the user data fields when launching an instance.

You can use AWS CloudFormation to automatically install, configure, and start applications on Amazon EC2 instances. Doing so enables you to easily duplicate deployments and update existing installations without connecting directly to the instance, which can save you a lot of time and effort. UserData is a property of the EC2 instance resource type.

```
Ec2Instance:
  Type: AWS::EC2::Instance
  Properties:
    UserData:
      !Base64 |
      #!/bin/bash
      yum -y update
      ..
```

A basic bootstrap script runs only on the first boot of the instance and is specifically useful to make sure your freshly installed instance has the latest updates, required packages or configurations etc. The UserData properties needs to be Base64 encoded and starts with #! and the interpreter. It's basically shell scripting with some limitations. It's not interactive so there is no direct feedback. You can redirect your output to an file like this.

```
exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console) 2>&1
```

This line has to be at the top of your UserData section. This way you have your debug logging available on the instance you just deployed.

Another drawback of this approach is that the UserData section can become a little messy after a while. To solve this CloudFormation provides a metadata section where you can describe any implementation details regarding your resource. Additionally, Python based helper scripts helps you interact with CloudFormation and access the specific meta data of a resource declared in the template.

CloudFormation helper scripts

CloudFormation includes a set of helper scripts (cfn-init, cfn-signal, cfn-get-metadata, and cfn-hup) that are based on cloud-init. You call these helper scripts from your AWS CloudFormation templates to install, configure, and update applications on Amazon EC2 instances that are in the same template.

cfn-init

Reads and interprets Metadata to execute AWS::CloudFormation::Init. This script is called in your UserData section.

cfn-signal

The cfn-signal helper script signals AWS CloudFormation to indicate whether Amazon EC2 instances have been successfully created or updated. If you install and configure software applications on instances, you can signal AWS CloudFormation when those software applications are ready.

cfn-get-metadata

Can be used to retrieve Metadata based on a specific key.

cfn-hup

The cfn-hup helper is a daemon that detects changes in resource Metadata and runs user-specified actions when a change is detected. This allows you to make configuration updates on your running Amazon EC2 instances through the UpdateStack API action. We've already seen this helper script in action in Elastic beanstalk.

Metadata

CloudFormation Metadata

Additional to bootstrap scripts you can include meta data on an EC2 instance. In comparison to the User Data, where basic shell scripting is used, meta data will follow a declarative approach to setting up the EC2 instances. Use the AWS::CloudFormation::Init type to include meta data on an Amazon EC2 instance for the cfn-init helper script. When your template calls the cfn-init script, the script looks for resource Metadata rooted in the AWS::CloudFormation::Init Metadata key. The cfn-init command can be found in UserData property. Basically the previous UserData example will be extended with this extra line:

```
/opt/aws/bin/cfn-init -v --stack ${AWS::StackName} \
--resource MyInstance --region ${AWS::Region}
```

More about this later. The Metadata configuration is separated into sections. The following example shows how to attach Metadata for cfn-init to an Amazon EC2 instance resource within the template.

```
Resources:
  MyInstance:
    Type: AWS::EC2::Instance
    Metadata:
      AWS::CloudFormation::Init:
        config:
          packages:
            :
          groups:
            :
          users:
            :
          sources:
            :
          files:
            :
          commands:
            :
          services:
            :
    Properties:
```

Metadata is organized into config keys, which you can group into config sets. You can specify a config set when you call cfn-init in your template. If you don't specify a config set, cfn-init looks for a single config key named config. The cfn-init helper script processes these configuration sections in the following order: packages, groups, users, sources, files, commands, and then services. If you require a different order, separate your sections into different config keys, and then use a config set that specifies the order in which the config keys should be processed.

```
AWS::CloudFormation::Init:
  configSets:
    ascending:
      - "config1"
      - "config2"
    descending:
      - "config2"
      - "config1"
  config1:
    commands:
      test:
        command: "echo \"$CFNTEST\" > test.txt"
        env:
          CFNTEST: "I come from config1."
        cwd: "~"
  config2:
    commands:
      test:
        command: "echo \"$CFNTEST\" > test.txt"
        env:
          CFNTEST: "I come from config2"
        cwd: "~"
```

With cfn-init -c <configSet> you can call a specific set and force an ordering. Also, using configSets allows you to combine multiple configSets within your template.

```
AWS::CloudFormation::Init:
  1:
    commands:
      test:
        command: "echo \"$MAGIC\" > test.txt"
        env:
          MAGIC: "I come from the environment!"
        cwd: "~"
  2:
    commands:
      test:
        command: "echo \"$MAGIC\" >> test.txt"
        env:
          MAGIC: "I am test 2!"
        cwd: "~"
  configSets:
    test1:
      - "1"
    test2:
      - ConfigSet: "test1"
      - "2"
    default:
      - ConfigSet: "test2"
```

Let's dive into more detail and explore each section keys. Every key contains a set of (sub) keys. Some of them are required.

Commands key

The only required key in this section is command. This can either be a string or an array. Commands are processed in alphabetical order by name. Optionally you can specify environment variables, working directory and test-runs before being executed by cfn-init (dry run). Additionally you can ignore errors by setting ignoreErrors to true.

Files key

You can use the files key to create files on the EC2 instance. The content can be specified inline in the template or pulled from a URL.

```
files:
  /tmp/setup.mysql:
    content: !Sub |
      CREATE DATABASE ${DBName};
      CREATE USER '${DBUsername}'@'localhost' IDENTIFIED BY '${DBPassword}';
      GRANT ALL ON ${DBName}.* TO '${DBUsername}'@'localhost';
      FLUSH PRIVILEGES;
    mode: "000644"
    owner: "root"
    group: "root"
```

Creating a symlink can easily done by specify the symlink target in the content key. The mode key uses the first three digits for symlinks and the last three digits for setting permissions. To create a symlink, specify 120000. To specify permissions for a file, use the last three digits, such as 000644

```
files:
  /tmp/file1:
    content: "/tmp/file2"
    mode: "120644"
```

Sources key

Instead of declaring the content inside the template, you can use the source key to specify a specific URL, like GitHub etc. This may also be a S3 bucket. Additionally you can configure your access keys in the authentication key to successfully pull the content from S3. You need to configure the type key correctly, and tell CloudFormation about the kind of authentication. Use "s3" to authenticate to S3 buckets and "basic" to authenticate to GitLab or another site. The type of authentication determines which properties to use. "s3" requires secretKey, accessorized and bucket. "Basic" on the other hand requires an url, user name and password.

Packages key

This is very similar to configuration management. It allows you to install packages and specify the version. Version is not required, leaving this blank, CloudFormation assumes the latest version.

```
yum:
  httpd: []
```

In this example CloudFormation leverages the yum repository to install httpd with the latest version.

Services key

This key manages the services, but has some extra keys, which you may not expect at first sight. The following code snippet shows two examples of services managed by CloudFormation init.

```
services:
  sysvinit:
    nginx:
      enabled: "true"
      ensureRunning: "true"
      files:
        - "/etc/nginx/nginx.conf"
    php-fastcgi:
      enabled: "true"
      ensureRunning: "true"
      packages:
        yum:
          - "php"
          - "spawn-fcgi"
```

Besides declaring the run- and startup-state, you can also manage the files, sources and packages that are required to run the services. Any changes to this will trigger a restart of the service.

User and Group keys

These keys will manage your user and groups during creation.

```
groups:
  groupOne:
    gid: "45"
users:
  myUser:
    groups:
      - "groupOne"
    uid: "50"
    homeDir: "/tmp"
```

As already mentioned the Metadata is accessed by cfn-init via UserData. With Fn::Sub you can substitute the stackname and region pseudo parameters by the actual stackname and region at run-time. Always update aws-cfn-bootstrap to have to latest version installed.

```
Ec2Instance:
  Type: AWS::EC2::Instance
  Metadata:
    AWS::CloudFormation::Init:
      config:
      ..
  Properties:
    UserData:
      Fn::Base64:
        !Sub |
          #!/bin/bash -xe
          yum update -y aws-cfn-bootstrap
          /opt/aws/bin/cfn-init -v --stack ${AWS::StackName} \
          --resource MyInstance --region ${AWS::Region}
```

In this example cfn-init will look for the meta data of the resource MyInstance within the CloudFormation template and executes the configuration (search for config by default).

As described above, cfn-init only applies to the bootstrap process of your EC2 instance. Updating the configuration of your instances can be done with the cfn-hup daemon. cfn-hup runs user defined actions when a change is detected in the resource Metadata.

Updating EC2 instance with cfn-hup

Updating EC2 instance with cfn-hup

During the creation process in CloudFormation the cfn-init helper script enables you to manage the configuration of the AWS resources and their corresponding meta data. This applies only to the bootstrap of your resource. Through the AWS Management Console, AWS CloudFormation update-stack command, or the UpdateStack API call you can update your resources. The stack update can be a simple change to a parameter value or a more complex update that updates, adds, or removes resources. AWS CloudFormation updates resource properties, adds new resources, or removes unwanted resources. These changes may affect the applications running on instances in one of two ways:

Changing a resource in the template may require an update to the configuration of an instance. For example, if you add a database to the template for scaling, the application on an instance must be provided with the new connection string, and the instance may need a restart.

The meta data on the instance may have been updated. For example, you can update the version of a package that is deployed, add files or packages, or run additional commands.

To facilitate this, CloudFormation provides the cfn-hup helper to reconfigure, restart, or update an application on an instance as part of the stack update process. The cfn-hup helper is a daemon that performs the actions specified in the resource's Metadata after it detects changes in these Metadata. You can use the daemon to make configuration updates on your running Amazon EC2 instances through UpdateStack.

The cfn-hup helper must be configured to inspect the correct stack. This configuration is stored in the cfn-hup configuration file cfn-hup.conf. The cfn-hup helper uses the AWS credentials from the IAM role to retrieve the meta data. The IAM role is passed to the instance profile when the Amazon EC2 instance is created.

By default, every 10 minutes cfn-hup checks for changes in each configured resource path. When a change to the requested meta data is detected, the user action is triggered. User actions (also known as hooks) are defined in a hook configuration file. Hooks have a uniquely name. Each hook is configured in a separate section.

To support composition of several applications deploying change notification hooks, cfn-hup uses a directory /hooks.d which is located in the hooks configuration directory. All files in this directory are parsed and loaded using the same layout as hooks.conf. Hooks in /hooks.d with the same name as a hook in hooks.conf, are merged, possibly overwriting values from hooks.conf.

The hooks configurations are loaded when the cfn-hup daemon starts up, so new hooks require the daemon to be restarted. A cache of previous Metadata values is stored at /var/lib/cfn-hup/data/metadata_db (not human readable). This cache can be deleted by forcing cfn-hup to run all post.add actions again.

As described previously, the cfn-hup helper is a small daemon that you can use to execute hooks when the meta data on a resource are changed. The cfn-init function takes the packages and files that are defined in the Metadata and installs them on your Amazon EC2 instance. By combining cfn-hup hooks with the cfn-init script, you can automatically install new versions of software when you change the Metadata by updating the stack template. The following example is a hook file that you can install by using the files section in the AWS::CloudFormation::Init Metadata in your template:

```
"/etc/cfn/hooks.d/cfn-auto-reloader.conf" :
{  "  content": {
  "[cfn-auto-reloader-hook]\n",
  "triggers=post.update\n",
  "path=Resources.MyResource.Metadata.AWS::CloudFormation::Init\n",
  "action=/opt/aws/bin/cfn-init "
      "  --stack ", { "Ref" : "AWS::StackName" },
      "  --resource MyResource",
      "  --region     ", { "Ref" : "AWS::Region" }, "\n",
      "runas=root\n"
  }
}
```

In this file, we define a cfn-hup hook that looks for changes to the Metadata, defined in the MyResource resource (an EC2 instance for example) in the stack and calls cfn-init if there is a change. When the Metadata changes, cfn-init looks at all the versions of the packages that are defined for the MyResource resource and, if there was a change, installs the version from the new template. Because the templates are text files, you can version-control them just like any other application artifact. By doing so, you can version-control not only your AWS infrastructure configuration, but also the set of packages installed on your instances. If you specify a version of a package in the template, cfn-init attempts to install that version even if a newer version of the package is already installed on the instance.

If you do not specify a version and a version of the package is already installed, cfn-init does not install a new version; it assumes that you want to keep the existing version.

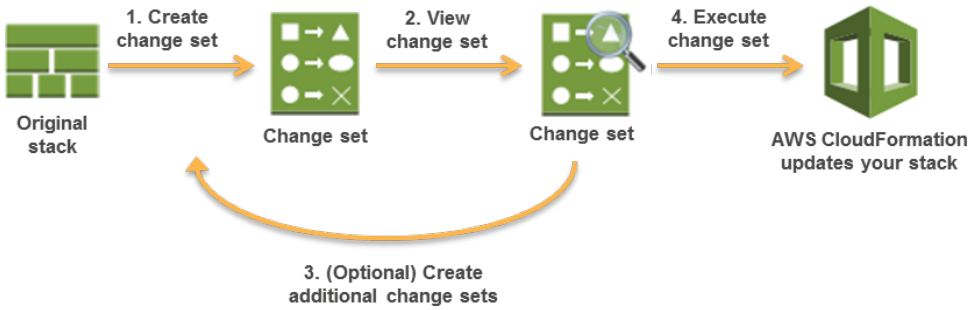
Updating the configuration of your application or OS is handled by the cfn-hup daemon running on the instance. Updating your AWS resources in your stack can be done with Change sets.

Updating AWS resources with Change sets

Updating AWS resources with Change sets

When you need to change your stack resources, this can be done via Change sets. Change sets allow you to preview how the proposed changes may impact the running resources. They don't indicate whether CloudFormation will successfully update a stack. For example, a Change set is unaware of any insufficient permissions on certain resources. In such a caste, CloudFormation will attempt to rollback your resources to their original state.

Here is an overview of how a change set will update your resources.



Creating a Change set can be done by the following command. You can use the same template as before and just change the parameters for example. This will deploy the Change set.

```
aws cloudformation create-change-set --stack-name Demo--change-set-name DemoChangeset
--use-previous-template --parameters ParameterKey=InstanceName,ParameterValue=somevalue
```

The output can be found in the CloudFormation console or via an API call from the cli. For example this:

```
aws cloudformation describe-change-set --change-set-name DemoChangeset
--stack-name demo
arn:aws:cloudformation:us-east-
1:xxxxxxxxxx:changeSet/DemoChangeset/6b0951dd-3a0d-4287-893b-
03f5e450db22      DemoChangeset   2019-09-
27T13:14:10.608Z        None    AVAILABLE       arn:aws:cloudformation:
us-east-1:xxxxxxxxx:stack/demo/f1d47e80-e127-11e9-947d-
1262c1c6cf8e      demo    CREATE_COMPLETE None    None
CHANGES Resource
RESOURCECHANGE  Modify  EC2Instance     i-
08076d71e01fc64d4     False   AWS::EC2::Instance
DETAILS         DirectModification      Dynamic
TARGET  Tags    Never
DETAILS InstanceName    ParameterReference      Static
TARGET  Tags    Never
SCOPE   Tags
PARAMETERS      InstanceName    somevalue
PARAMETERS      KeyName demo
PARAMETERS      SSHLocation     0.0.0.0/0
PARAMETERS      InstanceType    t2.micro
```

It describes what will be changed, when executing this Changeset. Executing a Changeset from the command line is quite similar to the previous command.

```
aws cloudformation execute-change-set \
--change-set-name DemoChangeset --stack-name demo
```

Working with Changesets gives you more control over the potential impact of changes. In addition, it opens the door to additional control over updates. IAM can be used to control access to specific CloudFormation functions (UpdateStack, CreateChangeSet etc). You could allow developers to create and view change sets and restrict execution to more experienced administrators.