## AWS Cloud Formation- Tutorial

A.N. Anil Kumar

CloudFormation is described as a JSON (JavaScript Object Notation) template. It's a model-driven template in that the AWS infrastructure is instantiated according to its own specification of proper order of execution. It is not a procedural language. If you are writing a CloudFormation template, you only need to follow the rules of **the CloudFormation external** **Domain Specific Language (DSL) in JSON notation.**

**In a template:**

Parameters are custom fields that users of the CloudFormation template enter to configure their environment. You can enter these parameters through the CloudFormation wizard available through the AWS Management Console, the command line interface or through the CloudFormation API. Within each parameter you can setup constraints for these parameters. Constraints may describe things like the Min and Max Size, whether or not to echo back what the user is entering in the parameter field, etc. There’s a syntax to describing a Parameter. First, you define the name Parameters in quotes followed by a colon. Then, you put an open curly brace. You use commas to delimit each parameter with the exception of the last parameter. Each parameter name is a custom name that you come up with. Each parameter name is in quotes. Then, you define each of the properties for the parameter. A parameter can be defined as aString, a Number of a CommaDelimitedList

**Create a Basic Amazon EC2 Instance Using CloudFormation**

We start with a basic template that defines a single Amazon EC2 instance with a security group that allows SSH traffic on port 22 and HTTP traffic on port 80. The template consists of five top-level JSON objects:

* **Description—**EC2\_Single\_Instance

EC2\_Single\_Instance specifies a single EC2 instance and a corresponding security group that allows SSH and HTTP access.

* **Parameters—**KeyName and InstanceType

The KeyName parameter specifies the EC2 keypair to use for SSH access, and the InstanceType parameter specifies the type of EC2 instance.

* **Mappings—**AWSInstanceType2Arch and AWSRegionArch2AMI

AWSInstanceType2Arch maps the proper architecture to the instance size so that the template user need specify only the instance size. AWSRegionArch2AMI maps AMI IDs to their specific region so that template users do not need to research which AMI IDs are available in each region.

* **Resources—**WebServer and WebServerSecurityGroup

The WebServer resource defines the Amazon EC2 instance and the WebServerSecurityGroup defines the security group that allows incoming traffic on ports 22 (SSH) and 80 (HTTP).

* **Outputs—**WebsiteURL

The WebsiteURL returns the public URL for the newly created website.

### Here is the template: Creating an Instance

{

"AWSTemplateFormatVersion" : "2010-09-09",

"Description" : "AWS CloudFormation Sample Template EC2\_Single\_Instance: Create a single EC2 instance. \*\*WARNING\*\* This template creates an Amazon EC2 instance. You will be billed for the AWS resources used if you create a stack from this template.",

"Parameters" : {

"KeyName" : {

"Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instances",

"Type" : "String",

"MinLength": "1",

"Default" : "cloudclass-new",

"MaxLength": "64",

"AllowedPattern" : "[-\_ a-zA-Z0-9]\*",

"ConstraintDescription" : "can contain only alphanumeric characters, spaces, dashes and underscores."

},

"InstanceType" : {

"Description" : "WebServer EC2 instance type",

"Type" : "String",

"Default" : "t1.micro",

"AllowedValues" : [ "t1.micro", "m1.small", "m1.large", "m1.xlarge", "m2.xlarge", "m2.2xlarge", "m2.4xlarge", "c1.medium", "c1.xlarge" ],

"ConstraintDescription" : "must be a valid EC2 instance type."

}

},

"Mappings" : {

"AWSInstanceType2Arch" : {

"t1.micro" : { "Arch" : "32" },

"t1.micro" : { "Arch" : "64" },

"t2.micro" : { "Arch" : "64" },

"m1.small" : { "Arch" : "32" },

"m1.small" : { "Arch" : "64" },

"m1.large" : { "Arch" : "64" },

"m1.xlarge" : { "Arch" : "64" },

"m2.xlarge" : { "Arch" : "64" },

"m2.2xlarge" : { "Arch" : "64" },

"m2.4xlarge" : { "Arch" : "64" },

"c1.medium" : { "Arch" : "32" },

"c1.xlarge" : { "Arch" : "64" }

},

"AWSRegionArch2AMI" : {

"us-east-1" : { "32" : "ami-7c356d2e", "64" : "ami-ac346cfe" },

"us-west-1" : { "32" : "ami-951945d0", "64" : "ami-971945d2" },

"us-west-2" : { "32" : "ami-16fd7026", "64" : "ami-10fd7020" },

"eu-west-1" : { "32" : "ami-24506250", "64" : "ami-20506254" },

"sa-east-1" : { "32" : "ami-3e3be423", "64" : "ami-3c3be421" },

"ap-southeast-1" : { "32" : "ami-74dda626", "64" : "ami-7edda62c" },

"ap-northeast-1" : { "32" : "ami-dcfa4edd", "64" : "ami-e8fa4ee9" }

}

},

"Resources" : {

"WebServer": {

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

"Properties": {

"ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" : "AWS::Region" },

{ "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" : "InstanceType" }, "Arch" ] } ] },

"InstanceType" : { "Ref" : "InstanceType" },

"SecurityGroups" : [ {"Ref" : "WebServerSecurityGroup"} ],

"KeyName" : { "Ref" : "KeyName" }

}

},

"WebServerSecurityGroup" : {

"Type" : "AWS::EC2::SecurityGroup",

"Properties" : {

"GroupDescription" : "Enable HTTP access via port 80",

"SecurityGroupIngress" : [

{"IpProtocol" : "tcp", "FromPort" : "80", "ToPort" : "80", "CidrIp" : "0.0.0.0/0"},

{"IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : "0.0.0.0/0"}

]

}

}

},

"PrivateBucket" : {

"Description" : "S3 bucket for storing credentials",

"Type" : "String",

"Default" : "jpalogbkt",

"ConstraintDescription" : "Must be a valid S3 Bucket"

},

"PublicBucket" : {

"Description" : "S3 bucket for storing build artifacts",

"Type" : "String",

"Default" : "www.cloud-b-lab.com",

"ConstraintDescription" : "Must be a valid S3 Bucket"

}

"Outputs" : {

"WebsiteURL" : {

"Value" : { "Fn::Join" : ["", ["http://", { "Fn::GetAtt" : [ "WebServer", "PublicDnsName" ]}]] },

"Description" : "URL for newly created EC2 Instance"

}

}

}

### Example 2: Installing a Tomcat server

{

"AWSTemplateFormatVersion" : "2010-09-09",

"Description" : "CloudFormation Template to create an Apache Tomcat server",

"Parameters" : {

"KeyName" : {

"Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instances",

"Type" : "String"

},

"InstanceType" : {

"Description" : "FormEngine EC2 instance type",

"Type" : "String",

"Default" : "m1.small"

}

},

"Mappings" : {

"AWSInstanceType2Arch" : {

"t1.micro" : { "Arch" : "64" },

"m1.small" : { "Arch" : "64" },

"m1.medium" : { "Arch" : "64" },

"m1.large" : { "Arch" : "64" },

"m1.xlarge" : { "Arch" : "64" },

"m2.xlarge" : { "Arch" : "64" },

"m2.2xlarge" : { "Arch" : "64" },

"m2.4xlarge" : { "Arch" : "64" },

"c1.medium" : { "Arch" : "64" },

"c1.xlarge" : { "Arch" : "64" },

"cc1.4xlarge" : { "Arch" : "64" }

},

"AWSRegionArch2AMI" : {

"us-east-1" : { "32" : "ami-7f418316", "64" : "ami-7341831a" },

"us-west-1" : { "32" : "ami-951945d0", "64" : "ami-971945d2" },

"us-west-2" : { "32" : "ami-16fd7026", "64" : "ami-10fd7020" },

"eu-west-1" : { "32" : "ami-24506250", "64" : "ami-20506254" },

"ap-southeast-1" : { "32" : "ami-74dda626", "64" : "ami-7edda62c" },

"ap-northeast-1" : { "32" : "ami-dcfa4edd", "64" : "ami-e8fa4ee9" }

}

},

"Resources" : {

"WebServerGroup" : {

"Type" : "AWS::EC2::SecurityGroup",

"Properties" : {

"GroupDescription" : "Enable SSH and HTTP access",

"SecurityGroupIngress" : [

{"IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : "0.0.0.0/0"},

{"IpProtocol" : "tcp", "FromPort" : "80", "ToPort" : "80", "CidrIp" : "0.0.0.0/0"}

]

}

},

"CfnUser" : {

"Type" : "AWS::IAM::User",

"Properties" : {

"Path": "/",

"Policies": [

{

"PolicyName": "Admin",

"PolicyDocument":

{ "Statement": [

{

"Effect":"Allow",

"Action":"\*",

"Resource":"\*"

}

]}

}

]

}

},

"HostKeys" : {

"Type" : "AWS::IAM::AccessKey",

"Properties" : {

"UserName" : {"Ref": "CfnUser"}

}

},

"WebServer": {

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

"Metadata" : {

"AWS::CloudFormation::Init" : {

"config" : {

"packages" : {

"yum" : {

"java-1.6.0-openjdk" : [],

"tomcat6" : [],

"httpd" : []

}

}

}

}

},

"Properties": {

"ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" : "AWS::Region" },

{ "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" : "InstanceType" }, "Arch" ] } ] },

"InstanceType" : { "Ref" : "InstanceType" },

"SecurityGroups" : [ {"Ref" : "WebServerGroup"} ],

"KeyName" : { "Ref" : "KeyName" },

"Tags" : [{ "Key" : "Name", "Value" : "WebServer" }],

"UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [

"#!/bin/bash -v\n",

"date > /home/ec2-user/starttime\n",

"yum update -y aws-cfn-bootstrap\n",

"## Error reporting helper function\n",

"function error\_exit\n",

"{\n",

" /opt/aws/bin/cfn-signal -e 1 -r \"$1\" '", { "Ref" : "WaitHandle" }, "'\n",

" exit 1\n",

"}\n",

"## Initialize CloudFormation bits\n",

"/opt/aws/bin/cfn-init -v -s ", { "Ref" : "AWS::StackName" }, " -r WebServer",

" --access-key ", { "Ref" : "HostKeys" },

" --secret-key ", {"Fn::GetAtt": ["HostKeys", "SecretAccessKey"]},

" --region ", { "Ref" : "AWS::Region" }, " > /tmp/cfn-init.log 2>&1 || error\_exit $(</tmp/cfn-init.log)\n",

"# Add Tomcat user to sudoers and disable tty\n",

"echo \"tomcat ALL=(ALL) NOPASSWD:ALL\" >> /etc/sudoers\n",

"echo \"Defaults:%tomcat !requiretty\" >> /etc/sudoers\n",

"echo \"Defaults:tomcat !requiretty\" >> /etc/sudoers\n",

"# Set JVM settings\n",

"echo \"JAVA\_OPTS='${JAVA\_OPTS} -Xms512m -Xmx512m -XX:PermSize=256m -XX:MaxPermSize=512m'\" >> /etc/tomcat6/tomcat6.conf\n",

"# Tomcat Setup\n",

"chown -R tomcat:tomcat /usr/share/tomcat6\n",

"service tomcat6 start\n",

"# Run Tomcat on server startup\n",

"chkconfig tomcat6 on\n",

"chkconfig --level 345 tomcat6 on\n",

"# Configure HTTPD\n",

"chkconfig httpd on\n",

"chkconfig --level 345 httpd on\n",

"# Proxy all requests to Tomcat\n",

"echo \"ProxyPass / ajp://localhost:8009/\" >> /etc/httpd/conf/httpd.conf\n",

"/etc/init.d/httpd start\n",

"/opt/aws/bin/cfn-signal", " -e 0", " '", { "Ref" : "WaitHandle" }, "'","\n",

"date > /home/ec2-user/stoptime"

]]}}

}

},

"IPAddress" : {

"Type" : "AWS::EC2::EIP"

},

"IPAssoc" : {

"Type" : "AWS::EC2::EIPAssociation",

"Properties" : {

"InstanceId" : { "Ref" : "WebServer" },

"EIP" : { "Ref" : "IPAddress" }

}

},

"WaitHandle" : {

"Type" : "AWS::CloudFormation::WaitConditionHandle"

},

"WaitCondition" : {

"Type" : "AWS::CloudFormation::WaitCondition",

"DependsOn" : "WebServer",

"Properties" : {

"Handle" : { "Ref" : "WaitHandle" },

"Timeout" : "1200"

}

}

},

"Outputs" : {

"InstanceIPAddress" : {

"Value" : { "Ref" : "IPAddress" }

},

"InstanceName" : {

"Value" : { "Fn::GetAtt" : [ "WebServer", "PublicDnsName" ] },

"Description" : "public DNS name of the new WebServer"

}

}

}

# Bootstrapping Applications using AWS CloudFormation

If you are installing and configuring your applications on EC2 dynamically at instance launch time, you will typically need to pull and install packages, deploy files and ensure services are started. AWS CloudFormation provides a set of helper scripts that, in conjunction with resource metadata defined in the template, can be used to install software and start services when you build your stack.

yum update -y aws-cfn-bootstrap

Refer: <http://docs.amazonwebservices.com/AWSCloudFormation/latest/UserGuide/cloudformation-waitcondition-article.html>

The sample template uses the *Fn::Base64* function to base64 encode the user data and to allow parameters and references from the template to be substituted in the script at runtime (in this case a reference to the WaitConditionHandle).

The example uses the *Fn::Join* function to concatenate the various pieces of the script.

"UserData" : { "Fn::Base64" : { "Fn::Join" : ["",[

"#!/bin/bash -ex","\n",

"yum -y install gcc-c++ make","\n",

"yum -y install mysql-devel sqlite-devel","\n",

"yum -y install ruby-rdoc rubygems ruby-mysql ruby-devel","\n",

"gem install --no-ri --no-rdoc rails","\n",

"gem install --no-ri --no-rdoc mysql","\n",

"gem install --no-ri --no-rdoc sqlite3","\n",

"rails new myapp","\n",

"cd myapp","\n",

"rails server -d","\n",

"curl -X PUT -H 'Content-Type:' --data-binary '{\"Status\" : \"SUCCESS\",",

"\"Reason\" : \"The application myapp is ready\",",

"\"UniqueId\" : \"myapp\",",

"\"Data\" : \"Done\"}' ",

"\"", {"Ref" : "WaitForInstanceWaitHandle"},"\"\n" ]]}}

So that the stack does not indicate *CREATE\_COMPLETE* until the packages have been installed and the application is running, we are using the new WaitCondition resource. In the Amazon EC2 instance resource definition above, you can see that the last line in the *UserData* script is a *CURL*command that signals the WaitCondition using the WaitConditionHandle resource called WaitForInstanceWaitHandle.

The WaitCondition itself is defined as follows:

"WaitForInstance" : {

"Type" : "AWS::CloudFormation::WaitCondition",

"DependsOn" : "Ec2Instance",

"Properties" : {

"Handle" : {"Ref" : "WaitForInstanceWaitHandle"},

"Timeout" : "600"

}

}

The WaitCondition definition uses the *DependsOn* construct. This ensures that the WaitForInstance WaitCondition resource is only created directly after the EC2 instance resource is created. Why is this important? The Timeout value specified in the WaitCondition (in this case 600 seconds) starts ticking when the WaitCondition object is put into the *CREATE\_IN\_PROGRESS* state. In this template, we want to give the Ruby application some time to start, but not too much time (in case something bad happened with the instance). By making the WaitCondition dependent on the Amazon EC2 instance, the WaitCondition resource will only be created after the EC2 instance enters the EC2 running state and the Cloud-init script starts. Using*DependsOn* ensures that the configuration script has 600 seconds to run. The stack creation will fail when the WaitCondition timeout triggers if the script does not signal via the *CURL* command.

NOTE: The WaitCondition resource can be used to synchronize creation of other resources in the template, not just stack creation. For example, you might chose not to associate the instance with an Elastic IP address until the application is running. By adding a DependsOn clause in other resources in the template that refer to the WaitCondition, you ensure that the resources that depend on the WaitCondition are not created until the WaitCondition is signaled. Watch for more articles about passing data back from the application in the template, as well as using WaitCondition objects to wait for multiple instances to be up and running before the application is considered healthy.

# cfn-signal

**Description**

You can use the cfn-signal helper script to pause the stack creation process. Use the cfn-signal script in conjunction with two AWS CloudFormation resources:

**Syntax**

cfn-signal --success|-s *signal.to.send* \

--reason|-r *resource.status.reason* \

--data|-d *data* \

--id|-i *unique.id* \

--exit-code|-e *exit.code* \

*waitconditionhandle.url*

**Options**

| **Name** | **Description** | **Required** |
| --- | --- | --- |
| -s, --success | if true, signal SUCCESS, else FAILURE.  Type: Boolean  *Default*: true | No |
| -r, --reason | A status reason for the resource event (currently only used on failure) - defaults to 'Configuration failed' if success is false.  *Type*: String | No |
| -d, --data | Data to send back with the waitConditionHandle. Defaults to blank.  *Type*: String  *Default*: blank | No |
| -i, --id | The unique id to send back with the WaitConditionHandle.  *Type*: String  *Default*: The ID of the Amazon EC2 instance. If the ID cannot be resolved, the machine's Fully Qualified Domain Name (FQDN) is returned. | No |
| -e, --exit-code | The error code from a process that can be used to determine success or failure. If specified, the *--success* option is ignored.  *Type*: String  *Examples*: -e $? (for Linux), -e %ERRORCODE% (for Windows) | No |
| waitconditionhandle.url | A pre-signed URL that you can use to signal success or failure to an associated WaitCondition  *Type*: String | Yes |

**Examples**

**Example 1**

A common usage pattern is to use cfn-init and cfn-signal together. The cfn-signal call uses the return status of the call to cfn-init (using the $? shell construct). If the application fails to install, the WaitCondition will fail to create and the stack will rollback.

"MyInstance": {

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

"Metadata": {

:

},

"Properties": {

"ImageId" : "ami-12345678",

"UserData" : {

"Fn::Base64" : {

"Fn::Join" : ["", [

"#!/bin/bash\n",

"/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" },

" -r MyInstance ",

" --region ", { "Ref" : "AWS::Region" },

"\n",

"/opt/aws/bin/cfn-signal -e $? '", { "Ref" : "MyHandle" }, "'\n",

] ]

}

}

}

},

**Example 2**

The following snippet shows another common usage pattern in which a function calls cfn-signal. Wrapping cfn-signal in a function makes it easier to call cfn-signal from multiple places.

"# Helper function\n",

"function error\_exit\n",

"{\n",

" /opt/aws/bin/cfn-signal -e 1 -r \"$1\" '", { "Ref" : "WaitHandle" }, "'\n",

" exit 1\n",

"}\n",

The following snippet shows the error\_exit function called in two commands.

"# Setup MySQL, create a user and a database\n",

"mysqladmin -u root password '", { "Ref" : "DBRootPassword" },

"' || error\_exit 'Failed to initialize root password'\n",

"mysql -u root --password='", { "Ref" : "DBRootPassword" },

"' < /tmp/setup.mysql || error\_exit 'Failed to initialize database'\n",

# cfn-init

## Description

The cfn-init helper script reads template metadata from the AWS::CloudFormation::Init key and acts accordingly to:

* Fetch and parse metadata from CloudFormation
* Install packages
* Write files to disk
* Enable/disable and start/stop services

**Note**

If you use cfn-init to update an existing file, it creates a backup copy of the original file in the same directory with a .bak extension. For example, if you update /*path*/*to*/*file\_name*, the action produces two files: /*path*/*to*/*file\_name*.bak contains the original file's contents and /*path*/*to*/*file\_name* contains the updated contents.

For information about the template metadata, see [AWS::CloudFormation::Init](http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-init.html).

**Note**

cfn-init does not require credentials, so you do not need to use the --access-key, --secret-key, --role, or --credential-fileoptions.

## Syntax

cfn-init --stack|-s *stack.name.or.id*

--resource|-r *logical.resource.id* \

--region *region*

--access-key *access.key* \

--secret-key *secret.key* \

--role *rolename*\

--credential-file|-f *credential.file* \

--configsets|-c *config.sets* \

--url|-u *service.url* \

-v

**Options**

| **Name** | **Description** | **Required** |
| --- | --- | --- |
| -s, --stack | Name of the Stack.  *Type*: String  *Default*: None  *Example*: -s { "Ref" : "AWS::StackName" }, | Yes |
| -r, --resource | The logical resource ID of the resource that contains the metadata.  *Type*: String  *Example*: -r WebServerHost | Yes |
| --region | The region to derive the CloudFormation URL from.  *Type*: String  *Default*: None  *Example*: --region ", { "Ref" : "AWS::Region" }, | No |
| --access-key | AWS Access Key for an account with permission to call DescribeStackResource on CloudFormation.  *Type*: String  Condition: The credential file parameter supersedes this parameter. | Conditional |
| --secret-key | AWS Secret Key that corresponds to the specified AWS Access Key.  *Type*: String  Condition: The credential file parameter supersedes this parameter. | Conditional |
| --role | The IAM role name.  *Type*: String  Condition: The credential file parameter supersedes this parameter. |  |
| -f, --credential-file | A file that contains both a secret key and an access key.  *Type*: String  Condition: The credential file parameter supersedes the --role, --access-key, and --secret-key parameters. | Conditional |
| -c, --configsets | A comma-separated list of configsets to run (in order).  *Type*: String  *Default*: default | No |
| -u, --url | The CloudFormation service URL to hit.  *Type*: String | No |
| -v | Verbose output. This is useful for debugging cases where cfn-init is failing to initialize.  **Note**  To debug initialization events, you should turn DisableRollback on. You can do this by using the CloudFormation console, selecting *Show Advanced Options*, and then setting "Rollback on failure" to "No". You can then SSH into the console and read the logs at /var/log/cfn-init.log. | No |

**Examples**

The following snippet is associated with a resource named WebServer.

"/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" },

" -r WebServer ",

" --region ", { "Ref" : "AWS::Region" }, \n",

The **Apache JServ Protocol** (**AJP**) is a binary protocol that can proxy inbound requests from a web server through to an application server that sits behind the web server.

It also supports some monitoring in that the web server can ping the application server. Web implementors typically use AJP in a load-balanced deployment where one or more front-end web servers feed requests into one or more application servers. Sessions are redirected to the correct application server using a routing mechanism wherein each application server instance gets a name (called a *route*). In this scenario the web server functions as a reverse proxy for the application server.

# Refer <http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html> for The Apache Tomcat Connector - AJP Protocol Reference