Tutrial

Automated AWS Provisioning and Deployment Using CloudFormation and CodeDeploy

Norman Wilde, Brian Eddy, Khyati Patel, Nathan Cooper, Keenal Shah, Valeria Gamboa, Bhavyansh Mishra

If you are using DevOps practices such as continuous deployment, a change to your application should trigger a deployment script. That script would provision virtual hardware and deploy the new code to it. Amazon Web Services (AWS) provides the CloudFormation service for automatic provisioning and the CodeDeploy service for automatic application deployment. This tutorial illustrates how to script these two services using their Command Line Interfaces (CLIs) (See note [1]). Careful setup is needed, but the result can be a straightforward reproducible deployment process.

To aid you if you wish to step through this tutorial yourself, we provide a zip with all the files used.

Last updated Feb. 7, 2016

Table of Contents

1 Intended Audience .................................................................................................................. 1
2 CloudFormation / CodeDeploy Background ........................................................................ 2
  2.1 The Basic Mechanism .................................................................................................. 2
  2.2 Authentication and Authorization Issues ...................................................................... 3
3 Setting Up Authentication and Authorization ..................................................................... 4
  3.1 Using the Management Consoles from your Workstation ..................................... 4
    3.1.1 Setting up the IAM EC2 Role and IAM instance profile .................................... 5
    3.1.2 Setting up the IAM Service Role ....................................................................... 6
    3.1.3 Setting Up the IAM User .................................................................................. 6
    3.1.4 Choose a Tag for identification ........................................................................... 8
    3.1.5 Create a Key Pair ............................................................................................... 8
  3.2 Setting Up on Your Build Server .................................................................................. 8
    4.1 Provisioning a Build Server with AWS CLI .............................................................. 8
    4.2 Setting Up Your Application .................................................................................... 9
    4.3 Setting Up CloudFormation ..................................................................................... 9
5 The Scripts and How To Use Them ................................................................................... 10
  5.1 The deploy.sh Script ................................................................................................. 10
  5.2 The cleanup.sh Script and Timing Problems .............................................................. 11
  5.3 Using the Scripts ......................................................................................................... 12
6 Successful Deployment ........................................................................................................ 13
7 Conclusions .......................................................................................................................... 13
8 Notes .................................................................................................................................. 13

1 INTENDED AUDIENCE

This is an intermediate level tutorial that presumes some knowledge of AWS and Unix. You should have some familiarity with the process for launching instances in EC2 and with EC2 terms such as key pairs, security groups, and virtual private clouds. You should know how to log in to
an EC2 instance and how to transfer files to it. We also assume some background with Unix directory structure and the use of the command line.

As you go through the tutorial you will be using Amazon's Identity and Access Management (IAM) service to manage authentication and authorization, so you should probably have some previous exposure to IAM.

The main services we will be using are CloudFormation, CodeDeploy and S3. We describe how to use these services in our scenario, but if you have not encountered them before it might be useful to first watch the very short introductory videos that Amazon provides on each service's web site:

- CloudFormation: https://aws.amazon.com/cloudformation/
- CodeDeploy: https://aws.amazon.com/codedeploy/
- S3: https://aws.amazon.com/s3/

## 2 CloudFormation / CodeDeploy Background

### 2.1 The Basic Mechanism

Figure 1 shows the main components of the CloudFormation / CodeDeploy mechanism. On the left is your workstation, such as a Windows PC or a Mac. The deployment process is controlled from a build server, which in this tutorial will be a Linux instance launched from the Amazon Linux AMI. Such instances have the AWS Command Line Interface (CLI) ready installed[2].

![Figure 1 - The CloudFormation / CodeDeploy Mechanism](image)

In the top layer of the figure, on the build server we have:

1) A Bash master script, here named deploy.sh. This contains the commands:
a. To zip up the application and put it in S3
b. To tell CloudFormation to provision an EC2 instance
c. To tell CodeDeploy about the application and about the tag which identifies the instance.

2) A CloudFormation template file, here named cfTemplate.json. This contains directives to the CloudFormation service which describe the instance and its network environment. The template also contains a very important UserData section with another Bash script. This script will run on the instance as it starts up and will install and start the CodeDeploy Agent.

3) A directory, here called myApp, which contains everything needed by the application we are going to install. For the tutorial, the application will be a simple web site served using python's HTTP server. The web site consists of a single index.html test page. However to use CodeDeploy we have to add a startApp.sh script to start the server and an appspec.yml file with instructions for the CodeDeploy Agent.

The second layer of the figure shows the AWS services we will be using:

1) CloudFormation is an AWS service that automatically provisions and configures EC2 virtual hardware, such as instances, load balancers, and virtual networks.
2) CodeDeploy is a deployment service that consists of two parts, a service and an agent. The build server tells the CodeDeploy service when an application is available for deployment. The CodeDeploy Agent on the instance periodically queries the CodeDeploy service to see if there is deployment work to be done.
3) The S3 storage service holds the package needed to install the CodeDeploy Agent and, as well, holds the zip file for the application.

In the third layer of the figure we see the provisioned instance, now running in the EC2 cloud. It has the CodeDeploy Agent installed and has been tagged so that the CodeDeploy service can identify it.

2.2 Authentication and Authorization Issues

If CloudFormation and CodeDeploy can provision hardware and deploy applications, then obviously a potential attacker can use these same services to deploy malicious code. To prevent such attacks, credentials have to be provided and verified at each step of the deployment process. Amazon provides the Identity and Access Management (IAM) service and the Simple Token Service (STS) to help generate and manage the necessary credentials.

Figure 2 shows the main points where authentication / authorization happens:

1) At (1), the instance needs to access S3, first to get the CodeDeploy Agent package and then for that Agent to read the user's application zip file. The solution here is for CloudFormation to attach an IAM instance profile to the new instance. The profile contains an IAM EC2 role that grants S3 access, and has a trust relationship that allows EC2 services to assume the role. Any process running on the instance can then use that EC2 role to call STS and get a temporary token to use in calling S3 [3].

2) At (2), the CodeDeploy service needs to scan the user's EC2 instances and check their tags, so as to locate the specific instance where the application should be deployed. To provide this access we create an IAM service role that allows listing instances and reading their tags. The service role's trust relationship says that the CodeDeploy service can assume the role.

3) At (3), the deploy.sh script runs with credentials for CloudFormation, CodeDeploy and S3. Its credentials are also inherited by these services, for example to pass the IAM instance profile to the new instance. The solution is to create an IAM user with the right
policies, and then to supply the IAM user's credentials (secret access key and access key id) when starting the script.

There are two security principles to keep in mind. First, credentials should be closely held, so that as few people as possible can create and access them. The IAM roles and the IAM user are set up by an administrator and not modified during normal operations (though credentials should be periodically rotated). The generated credentials should not be hard-coded into scripts, since they would then be accessible to anybody having read access to the code repository.

Second, the credentials should grant least privilege, that is, they should not allow actions that are not essential for the job. For example the IAM user running the deploy.sh script needs authorizations to access the instance profile and the service role, but not to modify them or create new roles. [4]

3 SETTING UP AUTHENTICATION AND AUTHORIZATION

3.1 Using the Management Consoles from your Workstation
While everything can be done using the AWS command line, much of the setup is a one-time process so it is easier to use the management consoles for S3, IAM and EC2. From your workstation, log in to the top level AWS Console using either the password for your master account or else the password for an IAM user with full administrative access. Then select the appropriate console for the setup tasks described in the following sections [5].
We strongly suggest that you do all your work in this tutorial in the AWS us-east-1 region (N. Virginia). Some items you create, such as key pairs, and some we refer to, such as AMI id's, are specific to one region.

Each time you use one of the management consoles, check at the upper right to make sure you are in the correct region (Figure 3).

If you are using this tutorial as a starting point for your own deployments, you will want to change some of the names we use to identify resources such as S3 buckets, key pairs, etc. etc. In the spreadsheet CodeDeployTutorialSymbols.xlsx we have listed the names we have specified and where they may need to be changed.

You will also find a few places where you cannot use the same names as the ones we show you. Pick a different name and then use it consistently.

- The name of the S3 bucket (cdtutorial-uwf). S3 bucket names must be globally unique
- The name of the key pair (cdtutorialKeyPair). Pick one of your existing key pairs or create a new one
- The Amazon Resource Name (ARN) of the service role, which is used in the deploy.sh script. Amazon Resource Names are generated by AWS and are globally unique.

To identify these names in the text, they are underlined and in red.

### 3.1.1 Setting up the IAM EC2 Role and IAM instance profile

The objective here is to provide the provisioned instance with access to S3, but use an inline policy that only grants read access to buckets containing the CodeDeploy agent and the application zip files [6]. The steps are as follows:

1) Open the S3 Management Console and create a bucket (bucket name cdtutorial-uwf). It is probably best to specify the US-east (US Standard) region.
2) Open the IAM Management Console
3) Select the Roles tab
4) Select Create New Role and enter the name (cdtutorialEC2Role)
5) For role type, select AWS Service Role >> Amazon EC2. IAM will create the trust relationship to let EC2 instances use the role.
6) For now, do not add any policies. Just finish the wizard to create the role.
7) Go back to the Roles tab and select the new role.
8) Under the Permissions tab, add a custom Inline Policy (name cdtutorialPolicyEC2S3) using the policy editor. Enter the policy rules as shown in the table below.

The result is a Role and an Instance Profile, both named cdtutorialEC2Role, and having the following policies:

<table>
<thead>
<tr>
<th>Permissions Policy (cdtutorialPolicyEC2S3)</th>
<th>Trust Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>&quot;Version&quot;: &quot;2012-10-17&quot;,</td>
<td>&quot;Version&quot;: &quot;2012-10-17&quot;,</td>
</tr>
<tr>
<td>&quot;Statement&quot;: [</td>
<td>&quot;Statement&quot;: [</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>&quot;Effect&quot;: &quot;Allow&quot;,</td>
<td>&quot;Effect&quot;: &quot;Allow&quot;,</td>
</tr>
<tr>
<td>&quot;Action&quot;: [</td>
<td>&quot;Principal&quot;: {</td>
</tr>
<tr>
<td>&quot;s3:Get*&quot;,</td>
<td>&quot;Service&quot;:</td>
</tr>
<tr>
<td>&quot;s3:List*&quot;</td>
<td>&quot;ec2.amazonaws.com&quot;</td>
</tr>
<tr>
<td>]</td>
<td>},</td>
</tr>
<tr>
<td>},</td>
<td>&quot;Action&quot;:</td>
</tr>
<tr>
<td>&quot;Resource&quot;: [</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>
3.1.2 Setting up the IAM Service Role
The IAM Service Role allows the CodeDeploy service to list the user's EC2 instances and to read their tags, so as to locate which instances may need to be updated. The steps are as follows:

1) Open the IAM Management Console
2) Select the Roles tab
3) Select Create New Role and enter the name (cdtutorialServiceRole)
4) For role type, select AWS Service Role >> AWS CodeDeploy.
5) On the next screen, attach the AWSCodeDeployRole. This is an AWS managed role so IAM populates the policy for you.
6) Finish the wizard to create the role.

The result is a Role named cdtutorialServiceRole, and having the following policies:

<table>
<thead>
<tr>
<th>Permissions Policy (AWSCodeDeployRole)</th>
<th>Trust Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{</td>
<td>`{</td>
</tr>
<tr>
<td>&quot;Version&quot;: &quot;2012-10-17&quot;,</td>
<td>&quot;Version&quot;: &quot;2012-10-17&quot;,</td>
</tr>
<tr>
<td>&quot;Statement&quot;: [</td>
<td>&quot;Statement&quot;: [</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>&quot;Effect&quot;: &quot;Allow&quot;,</td>
<td>&quot;Sid&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td>&quot;Action&quot;: [</td>
<td>&quot;Effect&quot;: &quot;Allow&quot;,</td>
</tr>
<tr>
<td>&quot;autoscaling:CompleteLifecycleAction&quot;,</td>
<td>&quot;Principal&quot;: {</td>
</tr>
<tr>
<td>&quot;autoscaling:DeleteLifecycleHook&quot;,</td>
<td>&quot;Service&quot;:</td>
</tr>
<tr>
<td>&quot;autoscaling:DescribeAutoScalingGroups&quot;,</td>
<td>&quot;codedeploy.amazonaws.com&quot;</td>
</tr>
<tr>
<td>&quot;autoscaling:DescribeLifecycleHooks&quot;,</td>
<td>},</td>
</tr>
<tr>
<td>&quot;autoscaling:PutLifecycleHook&quot;,</td>
<td>&quot;Action&quot;:</td>
</tr>
<tr>
<td>&quot;autoscaling:RecordLifecycleActionHeartbeat&quot;,</td>
<td>&quot;sts:AssumeRole&quot;</td>
</tr>
<tr>
<td>&quot;ec2:DescribeInstances&quot;,</td>
<td>}</td>
</tr>
<tr>
<td>&quot;ec2:DescribeInstanceStatus&quot;,</td>
<td>}</td>
</tr>
<tr>
<td>&quot;tag:GetTags&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;tag:GetResources&quot;</td>
<td></td>
</tr>
<tr>
<td>},</td>
<td></td>
</tr>
<tr>
<td>&quot;Resource&quot;: &quot;*&quot;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Note that the AWS managed role includes autoscaling permissions, although we will not use them in this tutorial.

3.1.3 Setting Up the IAM User
The IAM User will be used to run the deploy.sh master script on the build server. The steps to create this user are:

1) Open the IAM Management Console
2) Select the Users tab and then Create New Users
3) Enter the user name (cdtutorialUser) and take the option to generate an access key. (This user will not need a password)
4) On the next screen, download the generated credentials for cdtutorialUser. You get a csv file containing the user name, the access key id, and a secret access key. Save this file for later use (file cdtutorialCredentials.csv).
5) Return to the Users tab of the IAM Management Console.
6) Select the new user and on the Permissions tab attach the following managed policies:
   - AmazonEC2FullAccess
   - AWSCodeDeployFullAccess
7) Return to the IAM Management Console and again select the new User
8) Under the Permissions tab, add three custom Inline Policies using the policy editor. The policies are as follows and their rules are shown in the table below:
   - Inline Policy: cdtutorialS3BucketFull - Allow full access to our S3 bucket
   - Inline Policy: cdtutorialCloudFormation - Allow full access to CloudFormation
   - Inline Policy: cdtutorialIAMRoles - Allow limited access to IAM to pass the roles we have created to the services that need them.

<table>
<thead>
<tr>
<th>Inline Policy</th>
<th>Rules</th>
</tr>
</thead>
</table>
| cdtutorialS3BucketFull | ```json
  { "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "s3:PutObject",
          "s3:GetObject",
          "s3:DeleteObject"
        ],
        "Resource": "arn:aws:s3:::cdtutorial-uwf/*"
      }
    ]
  }
``` |
| cdtutorialCloudFormation | ```json
  { "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "cloudformation:*"
        ],
        "Resource": "*"
      }
    ]
  }
``` |
| cdtutorialIAMRoles | ```json
  { "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "iam:PassRole",
          "iam:ListInstanceProfiles"
        ],
        "Resource": "*"
      }
    ]
  }
``` |
3.1.4 Choose a Tag for identification
As previously mentioned, CodeDeploy scans your instances to see which ones need to have your application installed. The right instance (technically the right Deployment Group of instances) is identified by a tag consisting of a key / value pair. We picked:
   - Tag key: gTag
   - Tag value: myApp

3.1.5 Create a Key Pair
While not strictly necessary, it is desirable to have a key pair for use in logging in to the instances you create in case you need to debug. To create a key pair:
   1) Open the EC2 Management Console and select the Key Pairs tab.
   2) Click Create Key Pair and give the pair a name. We used cdtutorialKeyPair.
   3) Download the cdtutorialKeyPair.pem file containing the private key and put it in a safe place.
   4) If you will be using PuTTY you may want to convert the key to xxx.ppk format using PuTTYgen or a similar program.

4 Setting Up Your Build Server

4.1 Provisioning a Build Server with AWS CLI
As a build server we suggest provisioning an EC2 Linux instance starting from the Amazon Linux AMI [7]. Such instances have the AWS Command Line Interface (CLI) ready installed. If you choose to use a different build server you may need to install the CLI yourself following Amazon's documentation [2]. Give your build server instance a meaningful name so you can distinguish it when using the EC2 Management Console.

Get the zip file that accompanies this tutorial and transfer it to your build server using a secure FTP file manager such as WinSCP [8]. Log in to the build server and unzip the archive using the unzip command.

    unzip tutorial.zip

Verify that you have created the following directory structure:

<table>
<thead>
<tr>
<th>tutorial/cfTemplate.json</th>
</tr>
</thead>
<tbody>
<tr>
<td>tutorial/cleanup.sh</td>
</tr>
<tr>
<td>tutorial/deploy.sh</td>
</tr>
<tr>
<td>tutorial/iamPolicies</td>
</tr>
<tr>
<td>tutorial/iamPolicies/cdtutorialCloudFormation.json</td>
</tr>
<tr>
<td>tutorial/iamPolicies/cdtutorialIAMRoles.json</td>
</tr>
<tr>
<td>tutorial/iamPolicies/cdtutorialPolicyEC2S3.json</td>
</tr>
<tr>
<td>tutorial/iamPolicies/cdtutorialPolicyEC2S3trust.json</td>
</tr>
<tr>
<td>tutorial/iamPolicies/cdtutorialS3BucketFull.json</td>
</tr>
<tr>
<td>tutorial/myApp</td>
</tr>
<tr>
<td>tutorial/myApp/appData</td>
</tr>
<tr>
<td>tutorial/myApp/appData/index.html</td>
</tr>
<tr>
<td>tutorial/myApp/appspec.yml</td>
</tr>
<tr>
<td>tutorial/myApp/scripts</td>
</tr>
<tr>
<td>tutorial/myApp/scripts/startApp.sh</td>
</tr>
</tbody>
</table>
4.2 Setting Up Your Application
As previously mentioned, our application is a web site that consists of a single index.html test page. However to use CodeDeploy we have to add a startApp.sh script to start the server and an appspec.yml file with instructions for the CodeDeploy Agent. These files are found in the zip file.

The startApp.sh script is as follows:

```bash
#!/bin/bash
# Start python HTTP server to show test page on port 8000
# Run in background discarding output
cd /var/myApp/appData/
nohup python -m SimpleHTTPServer &>/dev/null &
```

The appspec.yml file is as follows [9]:

```yaml
# appspec.yml file to install myApp using CodeDeploy
version: 0.0
os: linux
# Copy all files to /var/myApp
files:
  - source: /
    destination: /var/myApp
hooks:
  ## After files have been copied, run the startApp.sh script
  ApplicationStart:
    - location: scripts/startApp.sh
      timeout: 300
```

4.3 Setting Up CloudFormation
The AWS CloudFormation service provisions "stacks" of virtual hardware based on an input template. Our template, shown below, is in file cfTemplate.json. It creates a stack containing a single instance generated from the Amazon Linux AMI for the us-east region. The security group allows access from anywhere to port 22 (for debugging) and to port 8000 (for the Python HTTP server). The Tags section specifies that the instance is named tutorialInstance and is tagged gTag:myApp.

Note particularly the "UserData" property of the instance. This is a Bash script that will run on the instance as it starts up. It gets the CodeDeploy Agent package for the us-east region and installs the Agent.

```json
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "cfTemplate.json - create a stack for CodeDeploy tutorial",
  "Resources" : {
    "EC2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "InstanceType" : "t2.micro",
        "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
        "KeyName" : "cdtutorialKeyPair",
        "IamInstanceProfile" : "cdtutorialEC2Role",
        "ImageId" : "ami-60b6c60a",
        "UserData" : { "Fn::Base64" : { "Fn::Join" : [""] },
          "#!/bin/bash -xe
```

```bash
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
5 THE SCRIPTS AND HOW TO USE THEM

5.1 The deploy.sh Script

Now that everything has been set up, one master script can do the provisioning and the installation. The deploy.sh script is shown below. Comments explain what each line does. The
commands beginning "aws" are the calls to the AWS Command Line Interface (CLI) that do most of the work. Note the names **underlined in red** that will need to be changed.

```bash
#!/bin/bash
# deploy.sh - master script to provision an EC2 instance and
# deploy a test application to it using AWS CloudFormation and CodeDeploy
# See CodeDeployTutorial.doc for a full explanation.

# Usage:
# (1) Run the "aws configure" command and, when prompted, provide
#     the AWS Access Key ID and AWS Secret Access Key of an IAM user
#     with sufficient permissions as described in CodeDeployTutorial.doc
# (2) cd to the "tutorial" directory containing this script and
#     give the command "./deploy.sh"

# Expected directory structure:
# tutorial/
#   deploy.sh - this script
#   cleanup.sh - script to cleanup AWS resources
#   cfTemplate.json - CloudFormation template
#   myApp/ - directory with files for the test application

# Cleanup any resources left from previous runs (prevents error msgs.)
./cleanup.sh

# Create a zip archive for myApp with no encompassing directory
echo "Creating zip archive for myApp"
cd myApp
zip -qr ../myApp.zip *
cd ..

# Upload the archive to S3
aws s3 cp myApp.zip s3://cdtutorial

# Run CloudFormation to create the stack and instance
templatebody="file://`pwd`/cfTemplate.json"
aws cloudformation create-stack --stack-name "cdtutorialStack" --template-body $templatebody

echo "Sleeping 3 minutes to allow instance to start"
sleep 180

# Run CodeDeploy to create a CodeDeploy Application for myApp
aws deploy create-application --application-name "cdtutorialMyApp"

# Run CodeDeploy to create a CodeDeploy DeploymentGroup for the instances
aws deploy create-deployment-group --application-name "cdtutorialMyApp" --deployment-group-name "cdtutorialDG" --ec2-tag-filters Key=gTag,Value=myApp,Type=KEY_AND_VALUE --service-role-arn arn:aws:iam::987839724330:role/cdtutorialServiceRole

# Run CodeDeploy to create a CodeDeploy Deployment for this revision of myApp
aws deploy create-deployment --application-name "cdtutorialMyApp" --deployment-config-name "cdtutorialDefault.OneAtATime" --deployment-group-name "cdtutorialDG" --description "cdtutorial deployment" --s3-location bucket="cdtutorial-uwf",bundleType=zip,key=myApp.zip

echo "Deployment complete. To test get instance's IP and point your browser"
```

5.2 The cleanup.sh Script and Timing Problems
Running deploy.sh creates a CloudFormation stack including the provisioned instance. It also creates a CodeDeploy application, a CodeDeploy deployment group, and a CodeDeploy deployment. Before trying to deploy a second time, or after any run that had an error, it is important to delete these resources. If you run a second time and some of these exist, you will get
error messages that may prevent successful deployment. For convenience, we provide a cleanup.sh script as shown below.

```bash
#!/bin/bash
# cleanup.sh - cleanup the CloudFormation stack and the CodeDeploy application after a successful or failed test.
# Cleanup removes all stack and CodeDeploy resources
#
# Usage:
# (1) Run the "aws configure" command and, when prompted, provide
#     the AWS Access Key ID and AWS Secret Access Key of an IAM user
#     with sufficient permissions as described in CodeDeployTutorial.doc
# (2) cd to the "tutorial" directory containing this script and
#     give the command "./cleanup.sh"
# echo "Starting cleanup of old resources"
# Clean up any previous CloudFormation stack from this tutorial
aws cloudformation delete-stack --stack-name "cdtutorialStack"
# Clean up any previous CodeDeploy artifacts from this tutorial
aws deploy delete-application --application-name cdtutorialMyApp
# echo "Sleeping 3 minutes to allow time for cleanup to complete"
sleep 180
# echo "End of cleanup script"
```

The cleanup.sh script is also called from within deploy.sh so accidental repeat runs should not cause problems.

Both deploy.sh and cleanup.sh include 3 minute waits to allow time for stack creation and tear down. This normally seems to be sufficient, but AWS operation times can vary so there may be occasional failures if something is working very slowly.

### 5.3 Using the Scripts

Before running either script, you need to provide the Access Key ID and the Secret Access Key of the IAM user that we set up previously. We stored these credentials in the file cdtutorialCredentials.csv. There are several ways to provide the credentials [10]. The easiest is to use the "aws configure" command on your build server. This prompts you to enter the credentials and stores them in file ~/.aws/credentials. In a production environment both this file and the csv file should be closely held.

The figure below shows the aws configure dialog with our actual credentials blanked out.

```bash
$ aws configure
AWS Access Key ID [***************XAZQ]: AKI...DA
AWS Secret Access Key [***************Xt6G]: n3...1et7
Default region name [us-east-1]: us-east-1
Default output format [json]: json
$
```

Once the credentials have been configured, first make sure that the scripts are executable. To do this you can cd to the tutorial directory and give the command:

    chmod a+x *.sh

Then to run the desired script give the command:

    ./deploy.sh

or

    ./cleanup.sh
If either script gives errors or seems to hang, you can look in the management consoles for CodeDeploy or CloudFormation to see what happened.

Further background information on the commands used in each script are given in file CodeDeployTutorialScriptBackground.

6 SUCCESSFUL DEPLOYMENT

After a successful run of the deploy.sh script, you should have a provisioned instance running in EC2. Use the EC2 Management Console to look for the instance. It should be named "tutorialInstance" and, if you check its tags in the lower part of the console window, you should find a tag with key "gTag" and value "myApp".

If you want to check that the Python application started correctly, get the public IP of the provisioned instance from the EC2 console, and point your browser to port 8000 at that IP. You should see something like Figure 4.

![Figure 4 - The Python Application Running](image)

To avoid additional AWS charges, don't forget to run the cleanup.sh script and terminate your build server when finished.

7 CONCLUSIONS

As we have seen, setting up CloudFormation and CodeDeploy for command line use is fairly complicated and there are many things that can go wrong. However once setup is complete it becomes easy to provision virtual hardware and deploy an application to it by running a single, fairly simple, script. Your script, with its associated configuration files, may be tested, inspected for correctness, and then kept under configuration management control. You then have a secure and reproducible way of deploying your software.

8 NOTES

[1] In this tutorial we will use CodeDeploy to produce a heavily baked (immutable) instance, which has all the necessary hardware and software for an application. Such instances are not maintained. Instead, if necessary, the instance would be re-created by rerunning the deploy script. While not covered in this tutorial, CodeDeploy may also be used on partially baked instances to deploy revisions of an application when needed. We also confine ourselves to deploying a single application on a single instance. CodeDeploy can manage groups of instances behind a load balancer, but this use is not discussed here.


The permissions we set in this tutorial are only moderately restrictive. For a production environment tighter limitations are probably desirable. See Amazon, *AWS CodeDeploy User Access Permissions Reference*, [http://docs.aws.amazon.com/codedeploy/latest/userguide/access-permissions.html](http://docs.aws.amazon.com/codedeploy/latest/userguide/access-permissions.html), [Link accessed December 2015].

In IAM you can use either AWS Managed Policies created by Amazon or Inline Policies you create yourself, or a mixture of the two. In this tutorial we will use a mixture. The advantages and drawbacks of each approach are described in Amazon, *Managed Policies and Inline Policies*, [http://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html](http://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html), [Link accessed December 2015].


