Alfresco Enterprise on AWS: Implementation Guide

October 2013

(Visit **http://aws.amazon.com/whitepapers/** for the latest version of this paper.)

Abstract

Amazon Web Services (AWS) provides a complete set of services and tools for deploying business-critical enterprise workloads on its highly reliable and secure cloud infrastructure. Alfresco is an enterprise content management system (ECM) useful for document and case management, project collaboration, web content publishing and compliant records management. Few classes of business-critical applications touch more enterprise users than enterprise content management (ECM) and collaboration systems.

This implementation guide is built on the Alfresco Enterprise on AWS: Reference Architecture whitepaper and is intended for IT infrastructure administrators and DevOps personnel. This guide provides the specific steps and techniques used to configure, deploy, and run an Alfresco server cluster (version 4.1) on AWS as described in the Reference Architecture whitepaper. The included AWS CloudFormation template and information in this guide can be modified to suit your specific business requirements or used as-is.

Introduction

Enterprises need to grow and manage their global computing infrastructures rapidly and efficiently while simultaneously optimizing and managing capital costs and expenses. The computing and storage services from AWS meet this need by providing a global computing infrastructure as well as services that simplify managing infrastructure, storage, and databases. With the AWS infrastructure, companies can rapidly provision compute capacity or quickly and flexibly extend existing on-premises infrastructure into the cloud.

Alfresco is an enterprise content management (ECM) platform for use by organizations interested in managing business-critical processes that are related to document management, collaboration, secure mobile and desktop access to vital files. The flexible compute, storage, and database services that AWS offers make it an ideal platform on which to run an Alfresco deployment.

Implementing Alfresco Enterprise on AWS

This implementation guide provides a walkthrough of the sample template and describes both the AWS and Alfresco implementation details used in the template to help create a secure, scalable, and highly available Alfresco implementation on AWS. You can customize the template and accompanying scripts as needed to best meet your business, IT, and security requirements. This guide is broken into three separate steps:

- Step 1: Sign up for an AWS account.
- Step 2: Perform prerequisite steps.
  - Creating a key-pair to be used for instances that are launched
  - Creating an Amazon Simple Storage Service (Amazon S3) bucket to be used as the storage repository
- Step 3: Launch the AWS CloudFormation template.
  - Creating the required AWS infrastructure
  - Creating a temporary setup instance to perform the initial Alfresco installation.
  - Modifying the Alfresco configuration to enable clustering.
  - Creating a new Amazon Machine Image (AMI) to be used as part of the auto scaling environment.
  - Configuring the Auto Scaling and Elastic Load Balancing (ELB) services from AWS.

The completed environment implements the architecture represented in the following diagram.

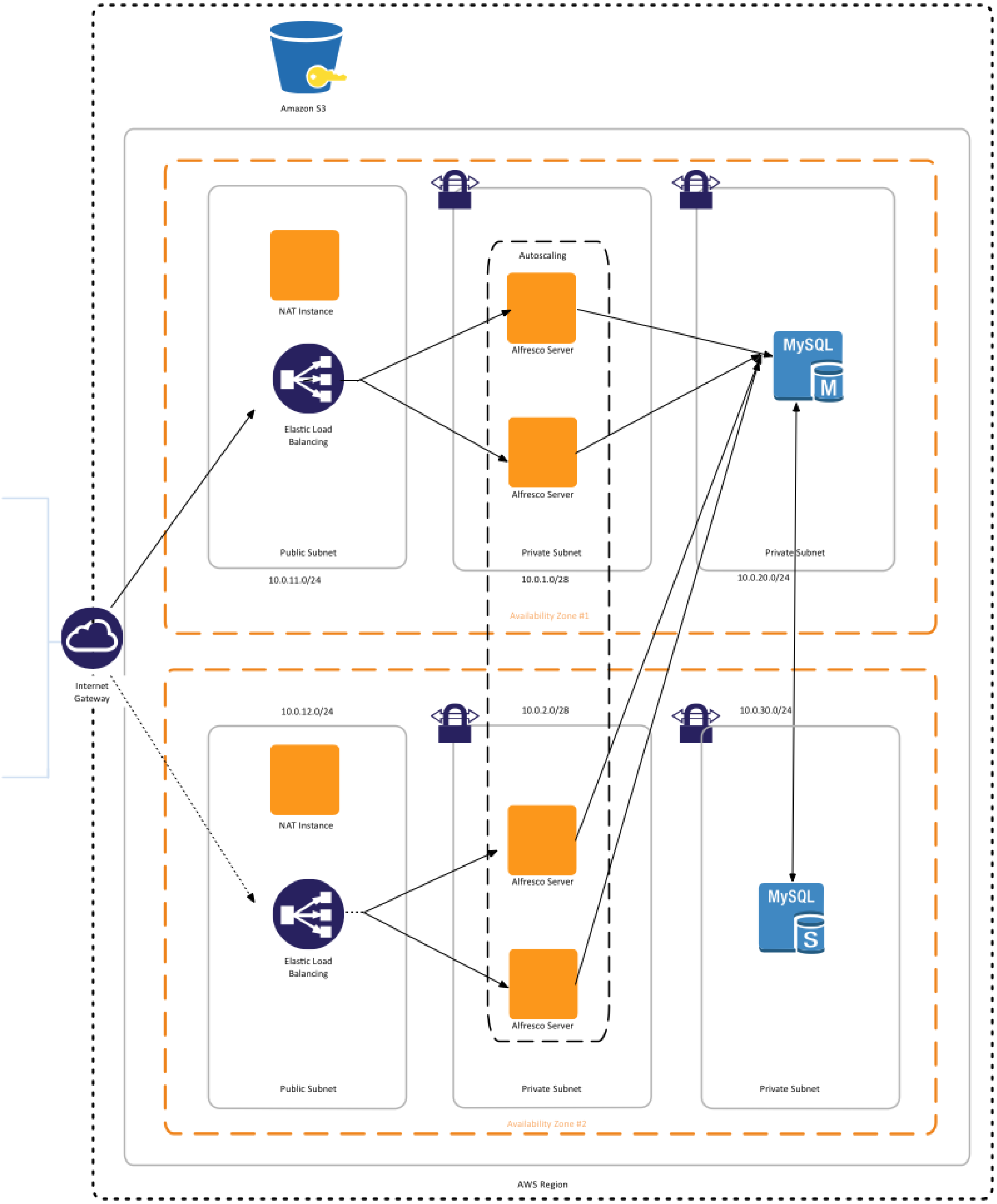

Figure 1: Alfresco Enterprise Reference Architecture

Step 1: Sign Up for an AWS Account

If you already have an AWS account, skip to the next step. If you don't already have an AWS account, use the following procedure to create one.

To create an AWS account, go to http://aws.amazon.com, and click **Sign Up Now**. Follow the on-screen instructions. Part of the sign-up process involves receiving a phone call and entering a PIN using the phone keypad.

When you create an AWS account, AWS signs up the account for all AWS services, including Amazon Elastic Compute Cloud (Amazon EC2), which you will use in the next step. You are charged only for the services that you use.

Step 2: Perform Prerequisite Steps

There are two items in the deployment that are not created automatically by the AWS CloudFormation template. You must create them before you launch the template:

- An Amazon EC2 key pair
- The Amazon S3 bucket used for shared storage

The Amazon EC2 key pair is used to provide SSH access to the instances that the AWS CloudFormation template creates. If you already have an Amazon EC2 key pair that you want to use, you can skip this step. If you need to create a key pair you can find instructions for doing so here: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

The deployment uses Amazon S3 as a shared storage repository in the Alfresco deployment. Amazon S3 buckets must be globally unique, so it's not a good practice to dynamically create the bucket in the template. The bucket to be used is a required parameter in the next step, when you launch the AWS CloudFormation template. The bucket must already exist or the Alfresco installation will fail.

Information on how to create a new Amazon S3 bucket can be found here:
http://docs.aws.amazon.com/AmazonS3/latest/gsg/CreatingABucket.html

Step 3: Launch the AWS CloudFormation Template

The AWS CloudFormation template creates the AWS infrastructure needed to deploy the Alfresco Enterprise cluster and installs and configures the Alfresco Enterprise software. This section walks through all the steps in this process. You can download the AWS CloudFormation template and follow along as the steps below describe the components of the template.

Template Parameters
In order to customize the alfresco deployment based on your needs, the AWS CloudFormation template requires a number of input parameters. The following table lists the parameters and describes their use within the template.

Parameter	Default	Description
KeyName	<User Provided>	Name of an existing Amazon EC2 key pair. All instances launch with this key pair.
SSHFrom	0.0.0.0/0	Lockdown SSH access to a known IP or CIDR block. (The default allows SSH to be accessed from

		anywhere.)
AZ1	<User Provided>	First Availability Zone to deploy into.
AZ2	<User Provided>	Second Availability Zone to deploy into
RDSInstanceType	db.m1.small	Type of Amazon EC2 instance for the Amazon RDS instance
AlfrescoInstanceType	m1.xlarge	Type of Amazon EC2 instance for the Alfresco instances
NATInstanceType	m1.small	Type of Amazon EC2 instance for the NAT instances
RDSUsername	<User Provided>	User name for the Amazon RDS database
RDSPassword	<User Provided>	Password for the Amazon RDS database
AlfrescoPassword	<User Provided>	Password for the Alfresco admin user
S3Bucket	<User Provided>	Name of the Amazon S3 bucket that Alfresco should use to store data. Note: This bucket must already exist before you launch the template.

Table 1: Template Parameters

Template Mappings

This template also makes use of the AWS CloudFormation Mappings feature to define some fixed parameters that can be referenced as the template is being executed. There are three mappings. The first two are for the AMIs that are used for the Alfresco servers and the NAT instances. The default settings are to use the Amazon Linux AMI in the region the template is being launched in for the Alfresco and the Amazon NAT Instance AMIs for the NAT instances. If you have an alternate preferred AMI, you can modify this mapping, but the NAT mapping must be to an AMI that is configured to be a NAT instance. The third mapping is for the IP ranges. The default mappings in the table correspond to the mappings depicted in the preceding architecture diagram. You can also modify this mapping, but it's important that the IP CIDR blocks for the two Alfresco subnets be /28.

Key	Value
us-east-1	ami-3275ee5b
us-west-1	ami-66d1fc23
us-west-2	ami-ecbe2adc
eu-west-1	ami-44939930
ap-southeast-1	ami-aa9ed2f8
ap-southeast-2	ami-363eaf0c
ap-northeast-1	ami-173fbf16
sa-east-1	ami-dd6bb0c0

Table 2: ALINUXAMI Mappings

Key	Value
us-east-1	ami-c6699baf
us-west-1	ami-3bcc9e7e

us-west-2	ami-52ff7262
eu-west-1	ami-0b5b6c7f
ap-southeast-1	ami-02eb9350
ap-southeast-2	ami-ab990e91
ap-northeast-1	ami-14d86d15
sa-east-1	ami-0439e619

Table 2: AWSNATAMI Mappings

Key	Value
VPC	10.0.0.0/16
NAT1	10.0.11.0/24
NAT2	10.0.12.0/24
Alfresco1	10.0.1.0/28
Alfresco2	10.0.2.0/28
RDS1	10.0.20.0/24
RDS2	10.0.30.0/24

Table 3: SubnetConfig Mappings

Amazon VPC and Subnet Setup

The first step that the AWS CloudFormation service takes when it executes the template is to create a virtual private cloud (VPC) and the subnets. (All the following steps are in the Resources section of the template.)

This launches a resource of the type AWS::EC2::VPC.

```
"VPC" : {
      "Type" : "AWS::EC2::VPC",
      "Properties" : {
        "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "VPC", "CIDR" ]},
        "Tags" : [
          {"Key" : "Application", "Value" : "Alfresco Cluster behind ELB with S3 for
shared storage and RDS (mySQL) for database"  }
        ]
      }
    }
```

Notice that for the “CidrBlock” property the template uses a built-in function called FindInMap that allows the AWS CloudFormation service to look up the value at runtime from the Mappings section.

Next the template creates the Internet Gateway and attaches it to the VPC. This is required to allow instances inside the VPC to access the Internet and to allow outside users to access resources within the VPC.

```
   "InternetGateway" : {
          "Type" : "AWS::EC2::InternetGateway"
        },

        "AttachGateway" : {
           "Type" : "AWS::EC2::VPCGatewayAttachment",
           "Properties" : {
             "VpcId" : { "Ref" : "VPC" },
             "InternetGatewayId" : { "Ref" : "InternetGateway" }
           }
        }
```

Next the template creates the subnets. Six subnets are created, three in each of the two Availability Zones.

```
    "AlfrescoSubnet" : {
          "Type" : "AWS::EC2::Subnet",
          "Properties" : {
            "VpcId" : { "Ref" : "VPC" },
            "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco1", "CIDR" ]},
            "Tags" : [
              {"Key" : "Application", "Value" : "Alfresco" },
              {"Key" : "Network", "Value" : "Private" }
            ],
            "AvailabilityZone" : { "Ref" : "AZ1"}
          }
        },
```

Notice in the above subnet code that we again leverage the FindInMap function to determine the CidrBlock associated with this subnet. This allows us to change the IP CIDR block ranges easily without needing to ensure that all references to the previous setting were changed. Additionally we use the “Ref” feature to reference a parameter that was supplied by the user to determine which Availability Zone this subnet should be placed in. Lastly we add some Tags that allow us to add information about this subnet to its metadata. These tags can be leveraged by management applications used in conjunction with Resource-Level Permissions for EC2. The remaining subnets have the same structure as the one above and are shown below.

```
    "AlfrescoSubnet2" : {
          "Type" : "AWS::EC2::Subnet",
          "Properties" : {
            "VpcId" : { "Ref" : "VPC" },
            "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco2", "CIDR" ]},
            "Tags" : [
              {"Key" : "Application", "Value" : "Alfresco" },
              {"Key" : "Network", "Value" : "Private" }
            ],
            "AvailabilityZone" : { "Ref" : "AZ2"}
          }
        },
    "NATSubnet" : {
          "Type" : "AWS::EC2::Subnet",
          "Properties" : {
            "VpcId" : { "Ref" : "VPC" },
            "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "NAT1", "CIDR" ]},
            "Tags" : [
              {"Key" : "Application", "Value" : "NAT" },
              {"Key" : "Network", "Value" : "Public" }

            ],
            "AvailabilityZone" : { "Ref" : "AZ1"}
          }
        },
    "NATSubnet2" : {
          "Type" : "AWS::EC2::Subnet",
          "Properties" : {
            "VpcId" : { "Ref" : "VPC" },
            "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "NAT2", "CIDR" ]},
            "Tags" : [
              {"Key" : "Application", "Value" : "NAT" },
              {"Key" : "Network", "Value" : "Public" }
            ],
            "AvailabilityZone" : { "Ref" : "AZ2"}
          }
        },
    "RDSSubnet" : {
          "Type" : "AWS::EC2::Subnet",
          "Properties" : {
            "VpcId" : { "Ref" : "VPC" },
            "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "RDS1", "CIDR" ]},
            "Tags" : [
              {"Key" : "Application", "Value" : "RDS" },
              {"Key" : "Network", "Value" : "Private" }
            ],
            "AvailabilityZone" : { "Ref" : "AZ1"}
          }
        },
      "RDSSubnet2" : {
          "Type" : "AWS::EC2::Subnet",
          "Properties" : {
            "VpcId" : { "Ref" : "VPC" },
            "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "RDS2", "CIDR" ]},
            "Tags" : [
              {"Key" : "Application", "Value" : "RDS" },
              {"Key" : "Network", "Value" : "Private" }
            ],
            "AvailabilityZone" : { "Ref" : "AZ2"}
          }
        }
```

NAT Instances

The template creates two NAT instances, one for each Availability Zones. The NAT instances allow instances in private subnets to access the Internet. In the AWS CloudFormation template, the NAT instances are described using several different components. The first is an Elastic IP (EIP):

```
"NATEIP" :
      {
        "Type" : "AWS::EC2::EIP",
        "Properties" :
             { "Domain" : "vpc" }
      }
```

This is followed by the security group that will be used:

```
"NATSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable internal access to the NAT device",
        "VpcId" : { "Ref" : "VPC" },
         "SecurityGroupIngress" : [
           { "IpProtocol" : "tcp", "FromPort" : "80",  "ToPort" : "80",  "CidrIp" : {
"Fn::FindInMap" : [ "SubnetConfig", "VPC", "CIDR" ]} } ,
           { "IpProtocol" : "tcp", "FromPort" : "443", "ToPort" : "443", "CidrIp" : {
"Fn::FindInMap" : [ "SubnetConfig", "VPC", "CIDR" ]} } ,
           { "IpProtocol" : "tcp", "FromPort" : "22",  "ToPort" : "22",  "CidrIp" : {
"Ref" : "SSHFrom" }} ],
        "SecurityGroupEgress" : [
           { "IpProtocol" : "tcp", "FromPort" : "80",  "ToPort" : "80",  "CidrIp" :
"0.0.0.0/0" } ,
           { "IpProtocol" : "tcp", "FromPort" : "443", "ToPort" : "443", "CidrIp" :
"0.0.0.0/0" } ,
           { "IpProtocol" : "tcp", "FromPort" : "22",  "ToPort" : "22",  "CidrIp" : {
"Fn::FindInMap" : [ "SubnetConfig", "Alfresco2", "CIDR" ]} } ,
           { "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : {
"Fn::FindInMap" : [ "SubnetConfig", "Alfresco1", "CIDR" ]} } ]
               }
             }
```

Notice that the above security group uses both ingress and egress rules. This is done to control what type of traffic the instance will act as a NAT instance for. Specifically, the above rules only allow HTTP and HTTPS traffic from within the VPC to be forwarded by the NAT instance. Additionally, the NAT instances act as bastion hosts in this deployment and are used when an administrator needs to SSH to one of the Alfresco servers.

Next we create the Elastic Network Interface (ENI) to use for the NAT instance and associate the previously created EIP with this interface.

```
"NATInterface" :
      {
          "Type" : "AWS::EC2::NetworkInterface",
          "Properties" : {
            "SubnetId" : { "Ref" : "NATSubnet" },
            "Description" :"External interface for the NAT instance in AZ1",
            "GroupSet" : [ {"Ref" : "NATSecurityGroup"} ],
            "SourceDestCheck" : "false",
            "Tags" : [ {"Key" : "Network", "Value" : "Public"}]
          }
        }
    "AssociateInterfaceNAT1" :
      {
        "Type" : "AWS::EC2::EIPAssociation",
        "Properties" :
        {
          "AllocationId" : { "Fn::GetAtt" : [ "NATEIP", "AllocationId" ]},
          "NetworkInterfaceId" : { "Ref" : "NATInterface" }
        }
      }
```

One key step that is taken in the creation of the ENI for NAT instances is disabling the source/destination check. This is needed because by default an Amazon EC2 instance must be either the source or the destination of any traffic it sends or receives. Since a NAT instance sends and receives traffic for which it is not the source or destination, this check must be disabled.

Now, with the dependencies created, we can create the NAT instances:

```
             "NATInstance" : {
```

```
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "Tags" :
        [
          {"Key" : "Name" ,"Value" : "NAT Instance" }

        ],
        "InstanceType" : { "Ref" : "NATInstanceType" },
        "KeyName" : { "Ref" : "KeyName" },
        "NetworkInterfaces" : [ { "NetworkInterfaceId" : {"Ref" :
"NATInterface"}, "DeviceIndex" : "0" } ],
        "ImageId" : { "Fn::FindInMap" : [ "AWSNATAMI", { "Ref" : "AWS::Region" },
"AMI" ]}


      }
    }
```

This process is repeated for the second NAT instance, though they will both share the same security group.

```
"NATEIP2" :
        {
        "Type" : "AWS::EC2::EIP",
        "Properties" :
           { "Domain" : "vpc" }
      },
"NATInterface2" :
      {
          "Type" : "AWS::EC2::NetworkInterface",
          "Properties" : {
            "SubnetId" : { "Ref" : "NATSubnet2" },
            "Description" :"External interface for the NAT instance in AZ1",
            "GroupSet" : [ {"Ref" : "NATSecurityGroup"} ],
            "SourceDestCheck" : "false",
            "Tags" : [ {"Key" : "Network", "Value" : "Public"}]
          }
        },
"AssociateInterfaceNAT2" :
      {
        "Type" : "AWS::EC2::EIPAssociation",
        "Properties" :
        {
          "AllocationId" : { "Fn::GetAtt" : [ "NATEIP2", "AllocationId" ]},
          "NetworkInterfaceId" : { "Ref" : "NATInterface2" }
        }
      },
"NATInstance2" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "Tags" :
        [

          {"Key" : "Name" ,"Value" : "NAT Instance" }

        ],
        "InstanceType" : { "Ref" : "NATInstanceType" },
        "KeyName" : { "Ref" : "KeyName" },
        "NetworkInterfaces" : [ { "NetworkInterfaceId" : {"Ref" :
"NATInterface2"}, "DeviceIndex" : "0" } ],
        "ImageId" : { "Fn::FindInMap" : [ "AWSNATAMI", { "Ref" : "AWS::Region" },
"AMI" ]}
```

```
               }
             }
```

Elastic Load Balancer

An Elastic Load Balancer (ELB) is used to distribute traffic amongst instances in the Alfresco cluster. Creating the ELB using the template consists of two steps. First we create the security group that defines which traffic the ELB should accept, and then we create the ELB itself.

```
"ELBSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Allow access to the ELB",
                "VpcId" : { "Ref" : "VPC" },
        "SecurityGroupIngress" :
                [
                    { "IpProtocol" : "tcp", "FromPort" : "80", "ToPort" : "80", "CidrIp" :
"0.0.0.0/0"},
                    { "IpProtocol" : "tcp", "FromPort" : "8080", "ToPort" : "8080",
"CidrIp" : "0.0.0.0/0"}

                ],
        "SecurityGroupEgress" :
                [
                    {"IpProtocol" : "tcp", "FromPort" : "8080", "ToPort" : "8080",
"CidrIp" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco1", "CIDR" ]} },
                    {"IpProtocol" : "tcp", "FromPort" : "8080", "ToPort" : "8080",
"CidrIp" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco2", "CIDR" ]} },
                    {"IpProtocol" : "tcp", "FromPort" : "7070", "ToPort" : "7070",
"CidrIp" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco1", "CIDR" ]} },
                    {"IpProtocol" : "tcp", "FromPort" : "7070", "ToPort" : "7070",
"CidrIp" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco2", "CIDR" ]} }
                ]
              }
    }
```

Notice again we are using both ingress and egress rules, but also that we have different ports for ingress than egress. This is because we are accepting traffic on 80 and 8080 externally, but as shown in the following ELB configuration, we forward the traffic to alternate ports. This is because the TCP/7070 port that is used by Alfresco for the SharePoint Protocol might be blocked as a non-standard port by many enterprise firewalls.

```
  "ElasticLoadBalancer" : {
   "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
   "Properties" : {
       "Subnets" : [ { "Ref" : "NATSubnet" } , { "Ref" : "NATSubnet2" } ],
       "SecurityGroups" : [ { "Ref" : "ELBSecurityGroup"}],
      "AppCookieStickinessPolicy" : [ { "CookieName" : "jsessionid", "PolicyName" :
"AlfrescoCluster" }],
      "Listeners" :
      [
        {
         "LoadBalancerPort" : "80",
         "InstancePort" :  "8080" ,
         "Protocol" : "HTTP",
         "PolicyNames" : [ "AlfrescoCluster"]
```

```
          },
          {
          "LoadBalancerPort" : "8080",
          "InstancePort" :  "7070" ,
          "Protocol" : "HTTP",
          "PolicyNames" : [ "AlfrescoCluster"]
          }
       ],


       "HealthCheck" : {
          "Target" : { "Fn::Join" : [ "", ["HTTP:", "8080" ,
"/alfresco/faces/jsp/dashboards/container.jsp"]]},

          "HealthyThreshold" : "2",
          "UnhealthyThreshold" : "3",
          "Interval" : "30",
          "Timeout" : "3"
              }
       }
}
```

In the preceding code you can see we have only enabled HTTP and not HTTPS. To enable HTTPS on the ELB you must provide your SSL certificate and private key. Do this using the Identity and Access Management (IAM) service, which is beyond the scope of this guide. Information on enabling SSL for ELB can be found here. In addition to the listeners, we also configured both a cookie stickiness policy and a health check. The cookie stickiness policy instructs the ELB to use the "jsessionid" cookie that is created by the Tomcat server to stick a particular client to a single Alfresco instance, though if that instance fails, traffic is forwarded to an alternate server. The health check is used as part of the Auto Scaling service to identify failed instances. Based on the settings above, the ELB requests the "/alfresco/faces/jsp/dashboards/container.jsp" page on port 8080 every 30 seconds. A request is considered successful if the server returns an HTTP 200 (OK). If an instance returns anything other than an HTTP 200, or if the timeout of three seconds is reached, it's considered an unsuccessful request. Instances that fail the health check three consecutive times are considered unhealthy and removed from the load balancer.

Route tables and Network ACLs

In VPC route tables and network ACLs are associated with subnets to provide routing information and stateless filtering of traffic into and out of a subnet. By default, a VPC will be created with a route table that has a single rule defining the entire VPC address space as a "local" route, meaning that all subnets within a VPC are reachable from one another. We add three additional route tables. Their use is described below.

First, we need a route table for our public subnets. Because the VPC Internet Gateway is a regional construct, we can use this same table for both of our public subnets.

```
"PublicRouteTable" : {
          "Type" : "AWS::EC2::RouteTable",
          "Properties" : {
            "VpcId" : {"Ref" : "VPC"},
            "Tags" :[ {"Key" : "Application", "Value" : "ELB and NAT Instance"  }]
          }
        },
        "PublicRoute" : {
          "Type" : "AWS::EC2::Route",
          "Properties" : {
            "RouteTableId" : { "Ref" : "PublicRouteTable" },
            "DestinationCidrBlock" : "0.0.0.0/0",
            "GatewayId" : { "Ref" : "InternetGateway" }
          }
        }
```

The following route tables are used for the Alfresco instances. Each Availability Zone has its own route table, because we direct outbound Internet traffic to the NAT instance in the same Availability Zone as the Alfresco instance.

```
"AlfrescoRouteTable" : {
          "Type" : "AWS::EC2::RouteTable",
          "Properties" : {
            "VpcId" : {"Ref" : "VPC"},
            "Tags" :[ {"Key" : "Application", "Value" : "Alfresco"  }]
          }
        },
"AlfrescoRoute" : {
          "Type" : "AWS::EC2::Route",
          "Properties" : {
            "RouteTableId" : { "Ref" : "AlfrescoRouteTable" },
            "DestinationCidrBlock" : "0.0.0.0/0",
            "InstanceId" : { "Ref" : "NATInstance" }
          }
        },
"AlfrescoRouteTable2" : {
          "Type" : "AWS::EC2::RouteTable",
          "Properties" : {
            "VpcId" : {"Ref" : "VPC"},
            "Tags" :[ {"Key" : "Application", "Value" : "Alfresco"  }]
          }
        },
"AlfrescoRoute2" : {
          "Type" : "AWS::EC2::Route",
          "Properties" : {
            "RouteTableId" : { "Ref" : "AlfrescoRouteTable2" },
            "DestinationCidrBlock" : "0.0.0.0/0",
            "InstanceId" : { "Ref" : "NATInstance2" }
          }
        }
```

The above route tables are then associated with the different subnets using the following code:

```
      "PublicSubnetRouteTableAssociation" : {
          "Type" : "AWS::EC2::SubnetRouteTableAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "NATSubnet" },
            "RouteTableId" : { "Ref" : "PublicRouteTable" }
          }
        },
         "PublicSubnetRouteTableAssociation2" : {
          "Type" : "AWS::EC2::SubnetRouteTableAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "NATSubnet2" },
            "RouteTableId" : { "Ref" : "PublicRouteTable" }
          }
        },
          "AlfrescoSubnetRouteTableAssociation" : {
          "Type" : "AWS::EC2::SubnetRouteTableAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "AlfrescoSubnet" },
            "RouteTableId" : { "Ref" : "AlfrescoRouteTable" }
          }
        },
         "AlfrescoSubnetRouteTableAssociation2" : {
          "Type" : "AWS::EC2::SubnetRouteTableAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "AlfrescoSubnet2" },
            "RouteTableId" : { "Ref" : "AlfrescoRouteTable2" }
          }
        }
```

Creating a network ACL adds an optional layer of security in addition to security groups. ACLs are used to restrict the type of traffic that one subnet can send to another. While we use security groups for most filtering, the Amazon RDS subnet has been configured with an ACL to only accept MySQL traffic (TCP/3306) from the two Alfresco subnets and not from the public subnet.

```
"RDSNetworkAcl" : {
     "Type" : "AWS::EC2::NetworkAcl",
     "Properties" : {
       "VpcId" : {"Ref" : "VPC"},
       "Tags" : [
         {"Key" : "Application", "Value" : "RDS Instance (mySQL)"  },
         {"Key" : "Network", "Value" : "Private" }
       ]
     }
   },


    "InboundRDSNetworkAclEntry" : {
     "Type" : "AWS::EC2::NetworkAclEntry",
     "Properties" : {
       "NetworkAclId" : {"Ref" : "RDSNetworkAcl"},
       "RuleNumber" : "100",
       "Protocol" : "6",
       "RuleAction" : "allow",
       "Egress" : "false",
       "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco1", "CIDR" ]},
       "PortRange" : {"From" : "3306", "To" : "3306"}
     }
   },
         "InboundRDSNetworkAclEntry2" : {
     "Type" : "AWS::EC2::NetworkAclEntry",
     "Properties" : {
       "NetworkAclId" : {"Ref" : "RDSNetworkAcl"},
       "RuleNumber" : "101",
       "Protocol" : "6",
       "RuleAction" : "allow",
       "Egress" : "false",
       "CidrBlock" : { "Fn::FindInMap" : [ "SubnetConfig", "Alfresco2", "CIDR" ]},
       "PortRange" : {"From" : "3306", "To" : "3306"}
     }
   },

   "OutBoundRDSNetworkAclEntry" : {
     "Type" : "AWS::EC2::NetworkAclEntry",
     "Properties" : {
       "NetworkAclId" : {"Ref" : "RDSNetworkAcl"},
       "RuleNumber" : "101",
       "Protocol" : "6",
       "RuleAction" : "allow",
       "Egress" : "true",
       "CidrBlock" : "0.0.0.0/0",
       "PortRange" : {"From" : "0", "To" : "65535"}
     }
   },

   "RDSSubnetNetworkAclAssociation" : {
     "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
     "Properties" : {
       "SubnetId" : { "Ref" : "RDSSubnet" },
       "NetworkAclId" : { "Ref" : "RDSNetworkAcl" }
     }
   },

   "RDSSubnetNetworkAclAssociation2" : {
     "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
     "Properties" : {
       "SubnetId" : { "Ref" : "RDSSubnet2" },
       "NetworkAclId" : { "Ref" : "RDSNetworkAcl" }
     }
   }
```

The Alfresco and public subnets primarily use security groups for network filtering. At the network ACL layer we are only imposing the restriction that traffic be TCP. Further ingress and egress filtering takes place using security groups.

```
      "PublicNetworkAcl" : {
          "Type" : "AWS::EC2::NetworkAcl",
          "Properties" : {
            "VpcId" : {"Ref" : "VPC"},
            "Tags" : [
              {"Key" : "Application", "Value" : "ELB and NAT Instance"  },
              {"Key" : "Network", "Value" : "Public" }
            ]
          }
        },
     "InboundPublicNetworkAclEntry" : {
          "Type" : "AWS::EC2::NetworkAclEntry",
          "Properties" : {
            "NetworkAclId" : {"Ref" : "PublicNetworkAcl"},
            "RuleNumber" : "100",
            "Protocol" : "6",
            "RuleAction" : "allow",
            "Egress" : "false",
            "CidrBlock" : "0.0.0.0/0",
            "PortRange" : {"From" : "0", "To" : "65535"}
          }
        },

        "OutBoundPublicNetworkAclEntry" : {
          "Type" : "AWS::EC2::NetworkAclEntry",
          "Properties" : {
            "NetworkAclId" : {"Ref" : "PublicNetworkAcl"},
            "RuleNumber" : "101",
            "Protocol" : "6",
            "RuleAction" : "allow",
            "Egress" : "true",
            "CidrBlock" : "0.0.0.0/0",
            "PortRange" : {"From" : "0", "To" : "65535"}
          }
        },
        "AlfrescoSubnetNetworkAclAssociation" : {
          "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "AlfrescoSubnet" },
            "NetworkAclId" : { "Ref" : "PublicNetworkAcl" }
          }
        },
             "AlfrescoSubnetNetworkAclAssociation2" : {
          "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "AlfrescoSubnet2" },
            "NetworkAclId" : { "Ref" : "PublicNetworkAcl" }
          }
        },
        "PublicSubnetNetworkAclAssociation" : {
          "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "NATSubnet" },
            "NetworkAclId" : { "Ref" : "PublicNetworkAcl" }
          }
        },
        "PublicSubnetNetworkAclAssociation2" : {
          "Type" : "AWS::EC2::SubnetNetworkAclAssociation",
          "Properties" : {
            "SubnetId" : { "Ref" : "NATSubnet2" },
            "NetworkAclId" : { "Ref" : "PublicNetworkAcl" }
          }
        }
```

Identity and Access Management (IAM) Configuration

Several points within the deployment require AWS APIs. An example is accessing the Amazon S3 bucket by the Alfresco instances to store and retrieve objects or configure the Auto Scaling service. To provide access to the instances, we use two mechanisms: IAM Roles for EC2 Instances and IAM user credentials. We also create two separate IAM Roles, one used during the setup and configuration steps, and one used by the resulting instances. This is because, during setup, we need to perform some configuration steps that are only needed during the initial setup.

The Setup Role is associated with the initial setup instance that is created later. This instance is used to build a custom AMI that has Alfresco installed and configured for the cluster. It is also used to create the Auto Scaling configuration as well as some Amazon CloudWatch metrics that can be used to monitor the deployment.

```
          "SetupRole" : {
              "Type" : "AWS::IAM::Role",
                     "Properties": {
               "AssumeRolePolicyDocument": {
                              "Statement": [ {
                                 "Effect": "Allow",
                                 "Principal": {
                                    "Service": [ "ec2.amazonaws.com" ] },
                                    "Action": [ "sts:AssumeRole" ]
                                   } ]
                           },

                      "Path": "/",
               "Policies": [ {
                            "PolicyName": "AlfrescoSetup",
                            "PolicyDocument": {
                                  "Statement": [{
                                       "Effect"   : "Allow",
                                       "Action"   : "cloudformation:DescribeStackResource",
                                       "Resource" : "*"
                                     },
                                     {
                                       "Effect"   : "Allow",
                                       "Action"   :[ "EC2:Describe*", "EC2:CreateImage", 
"ec2:TerminateInstances" ],
                                       "Resource" : "*"
                                     },
                                  {
                                       "Effect"   : "Allow",
                                       "Action"   : 
"elasticloadbalancing:DescribeLoadBalancers",
                                       "Resource" : "*"
                                     },
                                   {
                                       "Effect"   : "Allow",
                                       "Action"   :[ "autoscaling:create*", 
"autoscaling:put*", "autoscaling:DescribePolicies"  ],
                                       "Resource" : "*"
                                     },
                                   {
                                       "Effect"   : "Allow",
                                       "Action"   : "iam:PassRole",
                                       "Resource" : {"Fn::GetAtt" : ["AlfrescoRole", "Arn"] 
}
                                     },
                            {
                                    "Effect"   : "Allow",
                                     "Action"   : 
["cloudwatch:PutMetricData","cloudwatch:EnableAlarmActions", "cloudwatch:PutMetricAlarm" 
],
                                    "Resource" : "*"
                                  },
                                     {
                                       "Effect" : "Allow",
                                       "Action" : [ "s3:GetObject", "s3:PutObject", 
"s3:DeleteObject", "s3:ListBucket", "s3:Get*","s3:List*"],
                                       "Resource" : { "Fn::Join" : [ "", ["arn:aws:s3:::", 
{"Ref" : "S3Bucket"}, "/*" ] ]}


                                     },
                                     {
                                       "Effect" : "Allow",
                                       "Action" : [ "s3:List*"],
                                       "Resource" : "*"
                                     }

                                     ]
                                   }
                 }


                  ]}
      },

      "SetupRoleProfile" : {
             "Type": "AWS::IAM::InstanceProfile",
                      "Properties": {
                         "Path": "/",
                         "Roles": [ { "Ref": "SetupRole"} ]
                      }
      }
```

After the cluster is built and running, the Alfresco instances requires access to some APIs, but not to the extent that was required during the setup phase, so a second role with fewer permissions is created and used in the Auto Scaling launch configuration.

```
 "AlfrescoRole" : {
       "Type" : "AWS::IAM::Role",
              "Properties": {
               "AssumeRolePolicyDocument": {
                             "Statement": [ {
                                "Effect": "Allow",
                                "Principal": {
                                   "Service": [ "ec2.amazonaws.com" ] },
                                   "Action": [ "sts:AssumeRole" ]
                                  } ]
                          },

              "Path": "/",
               "Policies": [ {
                           "PolicyName": "AlfrescoCluster",
                           "PolicyDocument": {
                                  "Statement": [{
                                       "Effect"   : "Allow",
                                       "Action"   : "cloudformation:DescribeStackResource",
                                       "Resource" : "*"
                                     },
                                     {
                                       "Effect"   : "Allow",
                                       "Action"   : "EC2:Describe*",
                                       "Resource" : "*"
                                     },
                                                         {
                                                   "Effect"   : "Allow",
                                            "Action"   : "cloudwatch:PutMetricData",
                                                   "Resource" : "*"
                                                   },
                                     {
                                       "Effect" : "Allow",
                                       "Action" : [ "s3:GetObject", "s3:PutObject",
"s3:DeleteObject", "s3:ListBucket", "s3:Get*", "s3:CreateBucket", "s3:List*"],
                                       "Resource" : { "Fn::Join" : [ "", ["arn:aws:s3:::",
{"Ref" : "S3Bucket"}, "/*" ] ]}
                                     },
                                     {
                                       "Effect" : "Allow",
                                       "Action" : [ "s3:List*"],
                                       "Resource" : "*"
                                     }
                                     ]
                                   }
                           }
                  ]}
      },

      "AlfrescoRoleProfile" : {
              "Type": "AWS::IAM::InstanceProfile",
                       "Properties": {
                          "Path": "/",
                          "Roles": [ { "Ref": "AlfrescoRole"} ]
                       }
      }
```

Finally, some of the components used in the deployment do not support IAM Roles for Amazon EC2 Instances and must have a specific IAM user access key and secret access key provided. For these components, an IAM user and associated API credentials are created.

```
"AlfrescoUser" : {
          "Type" : "AWS::IAM::User",
          "Properties" : {
            "Policies": [{
              "PolicyName": "cfn-and-s3",
              "PolicyDocument" : {
                "Statement": [{
                  "Effect"   : "Allow",
                  "Action"   : "cloudformation:DescribeStackResource",
                  "Resource" : "*"
                },
                {
                  "Effect"   : "Allow",
                  "Action"   : "EC2:Describe*",
                  "Resource" : "*"
                },
                {
                  "Effect"   : "Allow",
                  "Action"   : "cloudwatch:PutMetricData",
                  "Resource" : "*"
                },
                {
                  "Effect" : "Allow",
                  "Action" : [ "s3:GetObject", "s3:PutObject", "s3:DeleteObject",
"s3:ListBucket", "s3:Get*", "s3:List*"],
                  "Resource" : { "Fn::Join" : [ "", ["arn:aws:s3:::",  {"Ref" :
"S3Bucket"}, "/*" ] ]}


                },
                {
                  "Effect" : "Allow",
                  "Action" : [ "s3:List*"],
                  "Resource" : "*"
                }
                ]
              }
            }
            ]
          }
        },
          "CFNKeys" : {
              "Type" : "AWS::IAM::AccessKey",
              "Properties" : {
                "UserName" : { "Ref": "AlfrescoUser" }
              }
            }
```

Amazon Relational Database Service (RDS) Configuration

Alfresco requires a database to store metadata about the objects and configuration data. The availability of this database is critical to the availability of the overall system. As such we have enabled the Multi-AZ feature on the MySQL Amazon RDS database. Deploying an Amazon RDS database with AWS CloudFormation requires three steps: Setting up the Amazon RDS instance, the Amazon RDS security group, and a subnet group.

The subnet group below defines which subnets should be used when deploying an Amazon RDS instance. We use the two subnets we created earlier.

```
 "RDSDBSubnetGroup" :
        {
                 "Type" : "AWS::RDS::DBSubnetGroup",
                  "Properties" : {
                      "DBSubnetGroupDescription" : "description",
                      "SubnetIds" :[{"Ref" : "RDSSubnet"}, {"Ref" : "RDSSubnet2"}]
                  }
            },
```

The Amazon RDS DBSecurityGroup is similar to an Amazon EC2 security group in that it defines from which instances or IP addresses the Amazon RDS instance should accept traffic. Since we know all our Alfresco instances have the same security group, we reference that group.

```
"RDSSecurityGroup" :
      {
              "Type" : "AWS::RDS::DBSecurityGroup",
              "Properties" : {
                 "EC2VpcId" : { "Ref" : "VPC" },
                  "GroupDescription" : "Allow AlfrescoSecurityGroup access to RDS DB",
                  "DBSecurityGroupIngress" : [ {
                  "EC2SecurityGroupId" : { "Ref" : "AlfrescoSecurityGroup" }

               }
               ]
             }
      }
```

Finally, we create the Amazon RDS instances using some of the parameters that were supplied by the user initially and the subnet and security groups we defined above.

```
"RDSDBinstance" :
       {
         "Type" : "AWS::RDS::DBInstance",
         "Properties" : {
            "MultiAZ" : "true",
            "DBSecurityGroups" : [ {"Ref" : "RDSSecurityGroup"} ],
            "AllocatedStorage" : "5",
            "DBInstanceClass" : {"Ref" : "RDSInstanceType"},
            "Engine" : "MySQL",
            "MasterUsername" : {"Ref": "RDSUsername"},
            "MasterUserPassword" : {"Ref":"RDSPassword"},
            "DBSubnetGroupName" : {"Ref" : "RDSDBSubnetGroup"},
            "DBName" : "alfresco"
         },
         "DeletionPolicy" : "Snapshot"
      }
```

The Amazon RDS instance created has a database of 5 GB. This should be sufficient for most deployments because the database is only storing metadata about the objects. If you find that you are nearing the capacity of your database you can easily resize it using the steps outlined here.

Alfresco Setup Instance

One key decision in how an environment in AWS is set up is to determine how much of the configuration is performed dynamically (often referred to as bootstrapping) and how much is pre-configured as part of the AMI. The complete set of steps to create a new instance for the cluster, including the installation of the Alfresco binaries, takes approximately 12-15 minutes to complete and result in a new node that is ready to accept requests. While this process can be scripted and performed in an automated fashion after a new instance is created, it takes too much time to install and configure the new cluster node to be effective in an Auto Scaling environment. For this reason the AWS CloudFormation template does not include the specific steps to configure Auto Scaling. Instead a setup instance is created that we use to create a new base AMI with Alfresco preconfigured for this cluster. We configure the Auto Scaling service from within this instance.

The process of creating this setup instance and ensuring that it has completely set up Auto Scaling takes 12-15 minutes. We need to make sure that the AWS CloudFormation service waits for this process to complete before continuing, which we do using WaitConditions, as shown below.

```
  "WaitForAlfrescoInstall" :
      {
            "Type" : "AWS::CloudFormation::WaitCondition",
            "DependsOn" : "AlfrescoInstance",
            "Properties" : {
              "Handle" : {"Ref" : "WaitForAlfrescoInstallWaitHandle"},
              "Timeout" : "1700"
            }
          },
       "WaitForAlfrescoInstallWaitHandle" :
        {
          "Type" : "AWS::CloudFormation::WaitConditionHandle",
          "Properties" : { }
        }
```

This pauses processing until the Alfresco instance has completed the installation and configuration of the Alfresco software and signaled a success back to the AWS CloudFormation service. If no signal is returned in 1700 seconds then the stack fails and rolls back the deployment.

The setup instance is the largest of all the resources created. This is because we are leveraging both EC2 user-data and CloudFormation Init Metadata to dynamically create or update the configuration files. Due to the length of this resource, it has been broken into smaller pieces to make it easier to describe what is being done in each block.

```
"AlfrescoInstance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "NetworkInterfaces" : [ { "NetworkInterfaceId" : {"Ref" : "AlfrescoInterface"},
"DeviceIndex" : "0" } ],
        "KeyName" : { "Ref" : "KeyName" },
        "ImageId" : { "Fn::FindInMap" : [ "ALINUXAMI", { "Ref" : "AWS::Region" }, "AMI"
]},
        "IamInstanceProfile" : { "Ref" : "SetupRoleProfile"  },
        "BlockDeviceMappings" : [ { "DeviceName" : "/dev/sdh", "VirtualName" :
"ephemeral0" }],
        "UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
           "#!/bin/bash -v\n",
            "yum update -y aws* \n",
            "yum install -y python-boto*\n" ,
            "echo #!/bin/bash -v >> /tmp/configSetup.sh \n",
            "echo export ELB_NAME=", { "Ref" : "ElasticLoadBalancer"  },"  >>
/tmp/configSetup.sh \n",
            "echo export AWS_REGION=", {"Ref" : "AWS::Region"  }, " >> /tmp/configSetup.sh
\n",
            "echo export INSTANCE_ID=$(curl http://169.254.169.254/latest/meta-
data/instance-id) >> /tmp/configSetup.sh \n",
            "echo export KEY_NAME=", { "Ref" : "KeyName"  },"  >> /tmp/configSetup.sh \n",
            "echo export SEC_GROUP=", { "Ref" : "AlfrescoSecurityGroup"  },"  >>
/tmp/configSetup.sh \n",
            "echo export INSTANCE_TYPE=", { "Ref" : "AlfrescoInstanceType"}," >>
/tmp/configSetup.sh \n",
            "echo export AZLIST=", { "Ref" : "AZ1"},",",  {"Ref" : "AZ2"},"  >>
/tmp/configSetup.sh \n",
            "echo export VPC_ZONE=",  { "Ref" : "AlfrescoSubnet" }, ","  , { "Ref" :
"AlfrescoSubnet2" } , " >> /tmp/configSetup.sh \n",
            "echo export ROLE=",  { "Ref" : "AlfrescoRoleProfile" },  " >>
/tmp/configSetup.sh \n",
            "cur=$(hostname  | sed 's/-/./g' | cut -c4-18)\n",
            "cur1=$(hostname) \n",
            "echo export CUR=\"$cur\" >> /tmp/configSetup.sh \n",
            "echo export CUR1=\"$cur1\" >> /tmp/configSetup.sh \n",
            "echo sed -i \"s/alfresco.ehcache.rmi.hostname=\"$CUR\"/ /g\"
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties >> /tmp/configSetup.sh \n",
            "echo sed -i \"s/alfresco.jgroups.bind_address=\"$CUR\"/ /g\"
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties >> /tmp/configSetup.sh \n",
            "echo sed -i \"s/$CUR1/ /g\" /etc/hosts \n",
            "echo service alfresco stop  >> /tmp/configSetup.sh \n",
            "echo killall java >> /tmp/configSetup.sh \n",

            "echo python /tmp/setupAS.py \\$ELB_NAME \\$AWS_REGION \\$INSTANCE_ID
\\$KEY_NAME \\$SEC_GROUP \\$INSTANCE_TYPE \\$AZLIST \\$VPC_ZONE \\$ROLE>>
/tmp/configSetup.sh \n",
            "chmod 700 /tmp/configSetup.sh \n",
            "/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" },
             " -r AlfrescoInstance",
             " --role=", { "Ref" : "SetupRole"  },
             " --region ", { "Ref" : "AWS::Region" }, "\n",
            "/home/ec2-user/installAlf.sh\n",
            "mkdir /opt/alfresco/tomcat/webapps/WEB-INF\n",
            "mkdir /opt/alfresco/tomcat/webapps/WEB-INF/lib\n",
            "mv /home/ec2-user/hazelcast-cloud-1.9.4.6.jar
/opt/alfresco/tomcat/webapps/WEB-INF/lib\n",
            "cd /opt/alfresco/tomcat/webapps/\n",
            "zip -u alfresco.war WEB-INF/lib/hazelcast-cloud-1.9.4.6.jar\n",
            "zip -u share.war WEB-INF/lib/hazelcast-cloud-1.9.4.6.jar\n",
            "cd /home/ec2-user\n",
            "sed -i 's/@@HZ_CLUSTER_NAME@@Share/AlfrescoS3Cluster/g' custom-slingshot-
application-context.xml\n",
            "sed -i 's/@@HZ_CLUSTER_PASSWORD@@/AlfrescoS3Cluster/g' custom-slingshot-
application-context.xml\n",
            "sed -i 's/@@AWS_ACCESS_KEY@@/", { "Ref" : "CFNKeys"}, "/g' custom-slingshot-
application-context.xml\n",
            "sed -i 's/@@AWS_SECRET_KEY@@/", {"Fn::GetAtt": ["CFNKeys",
"SecretAccessKey"]}, "/g' custom-slingshot-application-context.xml\n",
            "sed -i 's/@@AWS_REGION@@/", {"Ref" : "AWS::Region"},"/g' custom-slingshot-
application-context.xml\n",
            "sed -i 's/@@AWS_SECURITY_GROUP@@/",{ "Ref" : "AlfrescoSecurityGroup"} , "/g'
custom-slingshot-application-context.xml\n",
            "mv /home/ec2-user/hazelcast-ec2.xml
/opt/alfresco/tomcat/shared/classes/alfresco/extension\n",
            "mv /home/ec2-user/cache-context.xmll
/opt/alfresco/tomcat/shared/classes/alfresco/extension\n",
            "mv /home/ec2-user/ehcache-custom.xml
/opt/alfresco/tomcat/shared/classes/alfresco/extension\n",
            "mv /home/ec2-user/custom-slingshot-application-context.xml
/opt/alfresco/tomcat/shared/classes/alfresco/web-extension\n",

            "wget https://s3.amazonaws.com/publisher-bucket/bin/aws-
publisher_install.sh\n",
            "chmod u+x /home/ec2-user/aws-publisher_install.sh\n",
            "/home/ec2-user/aws-publisher_install.sh\n",
            "echo product.code   AlfrescoEnterprise > /etc/aws-publisher/cw.conf\n",
            "service aws-publisher stop \n",
           "service alfresco start\n",
            "sleep 330\n",
             "/tmp/configSetup.sh\n",
             "curl -X PUT -H 'Content-Type:' --data-binary '{\"Status\" : \"SUCCESS\",",
                                                           "\"Reason\" : \"The application
Alfresco Enterprise has been installed and started\",",
                                                           "\"UniqueId\" :
\"AlfrescoEnterprise\",",
                                                           "\"Data\" : \"Done\"}' ",
                  "\"", {"Ref" : "WaitForAlfrescoInstallWaitHandle"},"\"\n"



        ]]}},

        "InstanceType" : { "Ref" : "AlfrescoInstanceType" }
      },
```

The first portion above describes many of the basic features of the instance, such as the AMI ID to use, the Amazon EC2 key pair and the IAM Role to associate with the instance. We also enabled a BlockDeviceMapping for ephemeral0. The ephemeral storage devices provide non-persistent local storage to the instance. This storage is used for caching within the Alfresco cluster and is local to the host and is provided without any additional cost. The bulk of the code above is in the UserData section. We dynamically build out a bash script that will be executed later to perform the Auto Scaling setup, perform the installation of Alfresco, update configuration documents with both user-provided parameters and dynamically created API credentials, call the cfn-init service to perform additional bootstrapping, and then finally run the script to configure Auto Scaling. We do this at runtime because many of the objects being referenced didn't exist or were unknown until after the AWS CloudFormation template began to execute.

```
    "Metadata" : {
           "AWS::CloudFormation::Init" : {

          "config" : {
            "packages" :
              {

              },


            "files" : {
              "/home/ec2-user/installAlf.sh" :
                {
```

```
                "content" : { "Fn::Join" : ["", [
                  "echo enable-
components=alfrescosharepoint,javaalfresco,alfrescowcmqs,openofficecomponent >> /home/ec2-
user/option_file \n",
                  "echo disable-components=postgres >> /home/ec2-user/option_file \n",
                  "echo prefix=/opt/alfresco >> /home/ec2-user/option_file \n",
                  "echo alfresco_admin_password=", {"Ref" : "AlfrescoPassword"}, " >>
/home/ec2-user/option_file\n",
                  "echo jdbc_driver=org.gjt.mm.mysql.Driver >> /home/ec2-
user/option_file\n",
                  "echo \"jdbc_url=jdbc:mysql://", {"Fn::GetAtt" : [ "RDSDBinstance",
"Endpoint.Address" ] }, "/alfresco?useUnicode=yes&characterEncoding=UTF-8\"  >> /home/ec2-
user/option_file\n",
                  "echo jdbc_database=alfresco >> /home/ec2-user/option_file\n",
                  "echo jdbc_username=", { "Ref" : "RDSUsername"}, " >> /home/ec2-
user/option_file\n",
                  "echo jdbc_password=", { "Ref" : "RDSPassword"}, " >> /home/ec2-
user/option_file\n",
                  "echo baseunixservice_install_as_service=1 >> /home/ec2-
user/option_file\n",
                  "echo installer-language=en >> /home/ec2-user/option_file\n",
                  "echo mode=unattended >> /home/ec2-user/option_file\n",
                  "cur=$(hostname  | sed 's/-/./g' | cut -c4-18)\n",
                  "cur1=$(hostname)\n",
                  "echo \"$cur $cur1\" >> /etc/hosts\n",
                  "mkdir /mnt/alfresco\n",
                  "mkdir /mnt/alfresco/cache\n",
                  "chown -R ec2-user:ec2-user /mnt/alfresco\n",
                  "mount /dev/sdh /mnt/alfresco/cache\n",
                  "cd /home/ec2-user\n",
                  "unzip /home/ec2-user/alfresco-s3-connector-1.0.0-5.zip\n",
                  "/home/ec2-user/alfresco-enterprise-4.1.3-installer-linux-x64.bin --
optionfile /home/ec2-user/option_file \n",
                  "cp /home/ec2-user/alfresco-s3-connector-1.0.0-5.amp /opt/alfresco/amps
\n",
                  "cp /home/ec2-user/mysql-connector-java-5.1.22-bin.jar
/opt/alfresco/tomcat/lib \n",
                  "sed -i 's/read RESP/ /g' /opt/alfresco/bin/apply_amps.sh \n",
                  "sed -i 's/read DUMMY/ /g' /opt/alfresco/bin/apply_amps.sh \n",
                  "/opt/alfresco/bin/apply_amps.sh\n",
                  "echo s3.accessKey=", { "Ref" : "CFNKeys"}, " >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n",
                  "echo s3.secretKey=", {"Fn::GetAtt": ["CFNKeys", "SecretAccessKey"]}, "
>> /opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n",
                  "echo s3.bucketName=", {"Ref" : "S3Bucket"}, " >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n",
                         "echo \"alfresco.jgroups.bind_address=$cur\"   >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n",
                                      "echo \"alfresco.ehcache.rmi.hostname=$cur\"  >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n",
      "echo alfresco.cluster.name=AlfrescoS3Cluster >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
             "echo index.reindexMissingContent.cronExpression=0 \"*\" \"*\" \"*\" \"*\" ?
>> /opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
                  "echo alfresco.jgroups.bind_interface=eth0 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
                  "echo alfresco.jgroups.defaultProtocol=TCP >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
                  "echo
alfresco.tcp.initial_hosts=10.0.2.5[7800],10.0.2.6[7800],10.0.2.7[7800],10.0.2.8[7800],10.
0.2.9[7800],10.0.2.10[7800],10.0.2.11[7800],10.0.2.12[7800],10.0.2.13[7800],10.0.2.14[7800
],10.0.1.5[7800],10.0.1.6[7800],10.0.1.7[7800],10.0.1.8[7800],10.0.1.9[7800],10.0.1.10[780
0],10.0.1.11[7800],10.0.1.12[7800],10.0.1.13[7800],10.0.1.14[7800] >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n",
                    "echo dir.cachedcontent=/mnt/alfresco/cache >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
"echo system.content.caching.cacheOnInbound=true >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo system.content.caching.maxDeleteWatchCount=1 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo system.content.caching.contentCleanup.cronExpression=0 0 3 \"*\" \"*\"  ? >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.timeToLiveSeconds=0 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.timeToIdleSeconds=60 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.maxElementsInMemory=5000 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.maxElementsOnDisk=10000 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.minFileAgeInMillis=2000 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.maxUsageMB=10240 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo system.content.caching.maxFileSizeMB=0 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo alfresco.hazelcast.configLocation=classpath:alfresco/extension/hazelcast-
ec2.xml >> /opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo alfresco.hazelcast.password=AlfrescoS3Cluster >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo alfresco.hazelcast.protocol=ec2 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo alfresco.hazelcast.ec2.accesskey=", { "Ref" : "CFNKeys"}, "  >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
         "echo alfresco.hazelcast.ec2.secretkey=", {"Fn::GetAtt": ["CFNKeys",
"SecretAccessKey"]}, "  >> /opt/alfresco/tomcat/shared/classes/alfresco-
global.properties\n",
       "echo alfresco.hazelcast.ec2.region=", {"Ref" : "AWS::Region"}, " >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo alfresco.hazelcast.ec2.securitygroup=",{ "Ref" : "AlfrescoSecurityGroup"} , "
>> /opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo alfresco.hazelcast.ec2.tagkey=Name >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo alfresco.hazelcast.ec2.tagvalue=AlfrescoServer >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo alfresco.host=", { "Fn::GetAtt" : [ "ElasticLoadBalancer" , "DNSName"] }, "
>> /opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo alfresco.port=80 >> /opt/alfresco/tomcat/shared/classes/alfresco-
global.properties\n",
       "echo alfresco.protocol=http >> /opt/alfresco/tomcat/shared/classes/alfresco-
global.properties\n",
       "echo share.host=", { "Fn::GetAtt" : [ "ElasticLoadBalancer" , "DNSName"] }, "  >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo share.port=80 >> /opt/alfresco/tomcat/shared/classes/alfresco-
global.properties\n",
       "echo share.protocol=http >> /opt/alfresco/tomcat/shared/classes/alfresco-
global.properties\n",
       "echo vti.server.external.host=", { "Fn::GetAtt" : [ "ElasticLoadBalancer" ,
"DNSName"] }, "  >> /opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo vti.server.external.port=8080 >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n",
       "echo vti.server.external.protocol=http >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties\n"

                  ]]
```

```
              },
               "mode"    : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"
            },
              "/home/ec2-user/alfresco-enterprise-4.1.3-installer-linux-x64.bin" : {
"source" : "http://eu.dl.alfresco.com/release/enterprise/4.1/build-00097/alfresco-
enterprise-4.1.3-installer-linux-x64.bin",  "mode"   : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user" },
              "/home/ec2-user/alfresco-s3-connector-1.0.0-5.zip" : { "source" :
"http://eu.dl.alfresco.com/release/enterprise/4.1/build-00097/alfresco-s3-connector-1.0.0-
5.zip",  "mode"   : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
              "/home/ec2-user/mysql-connector-java-5.1.22-bin.jar" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/mysql-connector-java-5.1.22-bin.jar",  "mode"
: "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
              "/home/ec2-user/hazelcast-cloud-1.9.4.6.jar" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/hazelcast-cloud-1.9.4.6.jar",  "mode"   :
"000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
             "/home/ec2-user/cache-context.xml" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/cache-context.xml",  "mode"   : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
             "/home/ec2-user/custom-slingshot-application-context.xml" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/custom-slingshot-application-context.xml",
"mode"   : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
             "/home/ec2-user/ehcache-custom.xml" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/ehcache-custom.xml",  "mode"   : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
             "/home/ec2-user/hazelcast-ec2.xml" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/hazelcast-ec2.xml",  "mode"   : "000700",
                              "owner"  : "ec2-user",
                              "group"  : "ec2-user"},
            "/tmp/setupAS.py" : { "source" :
"https://s3.amazonaws.com/alfrescoInstaller/setupAS.py",  "mode"   : "000700",
                              "owner"  : "root",
                              "group"  : "root"}


          }
        }
      }
    }
}
```

The above section uses CloudFormation Init files section to download the needed installation files and dynamically update all the required Alfresco configuration files to enable clustering and use Amazon S3 as the shared storage. In addition to the Alfresco installer, we download several additional components:

- Hazelcast – An in-memory open source clustering implementation for Java
- Alfresco MySQL connector – Allows Alfresco to use MySQL as its database
- Ehcache – Java-based cache

- setupAS.py – Python script used to configure Auto Scaling

The setupAS.py script is run as the final step before signaling success back to CloudFormation. The script below first creates a new AMI from the current running instance. This AMI has Alfresco already installed and configured to act as part of the cluster. In the event that the cluster is under heavy load, a new instance can be added and ready to accept requests in only the amount of time required to boot the new instance.

Next the script configures the different components of the Auto Scaling and Amazon CloudWatch services to trigger the addition and removal of cluster nodes. Additionally the Auto Scaling launch configuration includes information about which security group new instances will use, as well as the ELB that they will be associated with.

```
import os
import boto
import sys
import logging
import time
import string
import random
from boto.ec2.connection import EC2Connection
from boto.ec2.autoscale import AutoScaleConnection
from boto.ec2.autoscale import LaunchConfiguration
from boto.ec2.autoscale import AutoScalingGroup
from boto.ec2.autoscale import ScalingPolicy
from boto.ec2.autoscale import Tag
from boto.ec2.cloudwatch import MetricAlarm

# Get values from cmd line
ELB_NAME        = sys.argv[1]
REGION          = sys.argv[2]
INSTANCE        = sys.argv[3]
KEY             = sys.argv[4]
SECGRP          = sys.argv[5]
TYPE            = sys.argv[6]
AZLIST          = sys.argv[7]
VPC_ZONE        = sys.argv[8]
ROLE            = sys.argv[9]

#generate random string to append to launch config and AS group names to prevent collisions
randomStr = ''.join(random.choice(string.ascii_uppercase + string.digits) for x in range(6))
asLCstr = 'AlfrescoLC-' + randomStr
asGrpStr = 'AlfrescoGrp-'+ randomStr
#connect to region
conn = boto.ec2.connect_to_region(REGION)
conn_as = boto.ec2.autoscale.connect_to_region(REGION)

#create new image from this running instance
AMIID = conn.create_image(INSTANCE, 'AlfrescoClusterAMI-'+ randomStr, 'Alfreco Cluster AMI for
Autoscaling Group', True)
logging.debug( ' AMIID=%s', AMIID)
time.sleep(20)

#setup ephemeral0 for local cache
blockDeviceMap = []
blockDeviceMap.append( {'DeviceName':'/dev/sdh', 'VirtualName' : 'ephemeral0'})

#create user-data string
userData = '#!/bin/bash \n cur=$(hostname  | sed \'s/-/./g\' | cut -c4-18) \n echo
\"alfresco.jgroups.bind_address=$cur\"   >> /opt/alfresco/tomcat/shared/classes/alfresco-
global.properties \n echo \"alfresco.ehcache.rmi.hostname=$cur\"  >>
/opt/alfresco/tomcat/shared/classes/alfresco-global.properties \n cur1=$(hostname)\n echo \"$cur $cur1\"
>> /etc/hosts\n'
#create launch configuration and AS group
launchConfig = LaunchConfiguration(name=asLCstr, image_id=AMIID, key_name=KEY, security_groups=[SECGRP],
instance_type=TYPE, instance_monitoring=True, instance_profile_name=ROLE,
block_device_mappings=blockDeviceMap, user_data=userData)
conn_as.create_launch_configuration(launchConfig)
```

```
        time.sleep(20)
        autoscaleGroup = AutoScalingGroup(group_name=asGrpStr , load_balancers=[ELB_NAME],
        availabilty_zones=[AZLIST], launch_config=launchConfig, vpc_zone_identifier=VPC_ZONE, min_size=2,
        max_size=6, health_check_period='360', health_check_type='ELB')
        conn_as.create_auto_scaling_group(autoscaleGroup)

        #setup tagging for the instances

        # create a Tag for the austoscale group
        as_tag = Tag(key='Name', value = 'Alfresco Server', propagate_at_launch=True, resource_id=asGrpStr)

        # Add the tag to the autoscale group
        conn_as.create_or_update_tags([as_tag])

        #create scale up and scale down policies for the autoscale group
        scaleUpPolicy = ScalingPolicy(name='alfrescoScaleUp-'+randomStr, adjustment_type='ChangeInCapacity',
        as_name=autoscaleGroup.name, scaling_adjustment=2, cooldown=400)
        scaleDownPolicy = ScalingPolicy(name='alfrescoScaleDown-'+randomStr, adjustment_type='ChangeInCapacity',
        as_name=autoscaleGroup.name, scaling_adjustment=-1, cooldown=400)
        conn_as.create_scaling_policy(scaleUpPolicy)
        conn_as.create_scaling_policy(scaleDownPolicy)

        #redeclare policies to populate the ARN fields
        policyResults = conn_as.get_all_policies(as_group=autoscaleGroup.name,
        policy_names=[scaleUpPolicy.name])
        scaleUpPolicy = policyResults[0]

        policyResults = conn_as.get_all_policies(as_group=autoscaleGroup.name,
        policy_names=[scaleDownPolicy.name])
        scaleDownPolicy = policyResults[0]

        #connect to Cloud Watch
        cw_conn = boto.ec2.cloudwatch.connect_to_region(REGION)

        #create the following alarms: ScaleUp @ Avg CPU >60% over 2 periods OR ELB latency >= 0.5sec.  ScaleDown
        @ Avg CPU <30% over 2 periods

        dimensions = {"AutoScalingGroupName" : autoscaleGroup.name}
        dimensions_elb = {"LoadBalancerName" : ELB_NAME}
        scaleUpAlarmCPU = MetricAlarm(name='Alfresco-HighCPU', namespace='AWS/EC2',metric='CPUUtilization',
        statistic='Average', comparison='>', threshold='60', evaluation_periods=2, period=60, unit='Percent' ,
        alarm_actions=[scaleUpPolicy.policy_arn], dimensions=dimensions)

        scaleDownAlarmCPU = MetricAlarm(name='Alfresco-LowCPU', namespace='AWS/EC2',metric='CPUUtilization',
        statistic='Average', comparison='<', threshold='30', evaluation_periods=2, period=60, unit='Percent',
        alarm_actions=[scaleDownPolicy.policy_arn], dimensions=dimensions)

        scaleUpAlarmLatency = MetricAlarm(name='Alfresco-HighLatency', namespace='AWS/ELB', metric='Latency',
        statistic='Average', comparison='>', threshold='1', evaluation_periods=2, period=60, unit='Seconds',
        alarm_actions=[scaleUpPolicy.policy_arn],dimensions=dimensions_elb)

        cw_conn.create_alarm(scaleUpAlarmCPU)
        cw_conn.create_alarm(scaleDownAlarmCPU)
        cw_conn.create_alarm(scaleUpAlarmLatency)

        #Terminate this setup instance now that auto-scaling is configured
        conn.terminate_instances(INSTANCE)

        #done
```

As the setup instance is terminated, the Auto Scaling service creates new instances based on the AMI just created and joined to the ELB. Shortly after the instances are launched, the ELB will be in-service and ready to accept requests. The output parameter of the template shown below includes the URL to your new Alfresco cluster.

```
"Outputs": {
        "AlfrescoServerOutput": {
            "Description": "URL to the ELB serving the Alfresco cluster",
            "Value": {"Fn::Join" : [ "", ["http://", { "Fn::GetAtt" : [ "ElasticLoadBalancer" , "DNSName"] }
,"/alfresco"] ] }
```

```
        }
    }
```

This completes the creation of the Alfresco cluster. After you click on the output link, you can log in to the Alfresco web interface using the password provided as a parameter when you launched the stack.

Security Group and Network ACL Configuration

The deployment in this implementation guide uses four different security groups and three Network ACLs that were described above. The table below describes each of the rules.

Elastic Load Balancing Security Group

Direction	Source or Destination	Protocol/Port	Description
Inbound	0.0.0.0/0	TCP/80	Allow inbound HTTP requests to the elastic load balancer.
Inbound	0.0.0.0/0	TCP/8080	Allow inbound SharePoint traffic on 8080.
Outbound	10.0.1.0/28	TCP/7070	SharePoint listener on Alfresco Instances in Availability Zone 1
Outbound	10.0.2.0/28	TCP/7070	SharePoint listener on Alfresco Instances in Availability Zone 2.
Outbound	10.0.1.0/28	TCP/8080	HTTP listener on Alfresco instances in Availability Zone 1.
Outbound	10.0.2.0/28	TCP/8080	HTTP listener on Alfresco instances in Availability Zone 2.

Alfresco Security Group

Direction	Source or Destination	Protocol/Port	Description
Inbound	Elastic load balancer	TCP/8080	Allow inbound HTTP requests from the elastic load balancer.
Inbound	Elastic load balancer	TCP/7070	Allow inbound SharePoint traffic from the elastic load

			balancer.
Inbound	10.0.1.0/28 10.0.2.0/28	TCP/5700-5710	Allow Hazelcast traffic.
Inbound	10.0.1.0/28 10.0.2.0/28	TCP/5800-5810	Alfresco RMI.
Inbound	10.0.1.0/28 10.0.2.0/28	TCP/7800	JGroups cluster port.
Inbound	<NAT Instances>	TCP/22	Allow SSH only from either of the two NAT instances.
Outbound	0.0.0.0	TCP/0-65535	All outbound.

NAT Instances Security Group

Direction	**Source or Destination**	**Protocol/Port**	**Description**
Inbound	<SSH From Parameter>	TCP/22	Allow SSH from IP range specified.
Inbound	10.0.0.0/16	TCP/80	Accept HTTP traffic from instances in the Amazon VPC.
Inbound	10.0.0.0/16	TCP/443	Accept HTTPS traffic from instances in the Amazon VPC.
Outbound	0.0.0.0/0	TCP/80	Outbound HTTP traffic.
Outbound	0.0.0.0/0	TCP/443	Outbound HTTPS traffic.
Outbound	10.0.1.0/28 10.0.2.0/28	TCP/22	Outbound SSH to Alfresco instances.

Amazon RDS Security Group

Direction	**Source or Destination**	**Protocol/Port**	**Description**
Inbound	Alfresco Security Group	TCP/3306	Allow MySQL traffic from Alfresco instances

Outbound	0.0.0.0/0	ALL	Allow outbound

Amazon RDS Subnet Network ACL

Direction	Source or Destination	Protocol/Port	Description
Inbound	10.0.1.0/28 10.0.2.0/28	TCP/3306	Allow MySQL traffic from Alfresco subnets.
Inbound	0.0.0.0/0	ALL	Deny all.
Outbound	0.0.0.0/0	TCP	Allow all TCP.

Alfresco and NAT Subnet Network ACL

Direction	Source or Destination	Protocol/Port	Description
Inbound	0.0.0.0/0	TCP	Allow all TCP.
Outbound	0.0.0.0/0	TCP	Allow all TCP.

Further Reading

1. Alfresco Enterprise on AWS: Reference Architecture Whitepaper: http://media.amazonwebservices.com/AWS_Alfresco_Enterprise_Reference_Architecture.pdf

2. AWS Alfresco Partner Page: http://www.aws-partner-directory.com/PartnerDirectory/PartnerDetail?id=7609

3. Alfresco on AWS: http://www.alfresco.com/aws

4. AWS CloudFormation: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/CHAP_Intro.html