Using Chef with AWS CloudFormation

Whenever you use AWS CloudFormation to provision Amazon Elastic Cloud Compute (EC2) instances, you have a number of options for deploying and configuring software applications to those instances. You can build a custom AMI, run user data scripts, or use a configuration management tool, such as Chef or Puppet. Each option can affect instance start time and software maintenance issues on those instances. The following walkthrough focuses on using Chef, a configuration management tool, with AWS CloudFormation. For more information about custom AMIs or user data scripts, see Amazon Machine Images (AMI) or Launching Instances with User Data in the Amazon EC2 user guide.

With Chef, you can automate the deployment of your software applications on Amazon EC2 instances instead of manually building various scripts. By combining Chef with AWS CloudFormation, you can consistently deploy and configure your AWS resources along with the software applications that run on top of it, all from a single AWS CloudFormation template.

For example, imagine you want to deploy a WordPress blog with a back-end database in AWS. You can use an AWS CloudFormation template to provision a web server and database. Using the same template, you can also pass in metadata and commands to your web server, with which you can install and run Chef. By running Chef, you can use a recipe to install and configure WordPress on your web server. With this single AWS CloudFormation template, you can quickly and easily get a WordPress blog running in AWS. You can also consistently deploy the same setup by reusing the same template. These templates also provide a history of your setup as you iterate your templates.

Before you read the walkthrough, note the following items:

- If you start with a bare-bones AMI and have a lot of software applications to install, bootstrapping your instances can take time. If you need to decrease the bootstrapping time, create a custom AMI that already includes many of the software applications that you require.
- The following walkthrough assumes familiarity with Chef and is not intended to be a tutorial on using Chef. Also, to demonstrate how to use Chef, we use a WordPress recipe. However, the recipe is intended for demonstration purposes only and is not maintained by AWS.

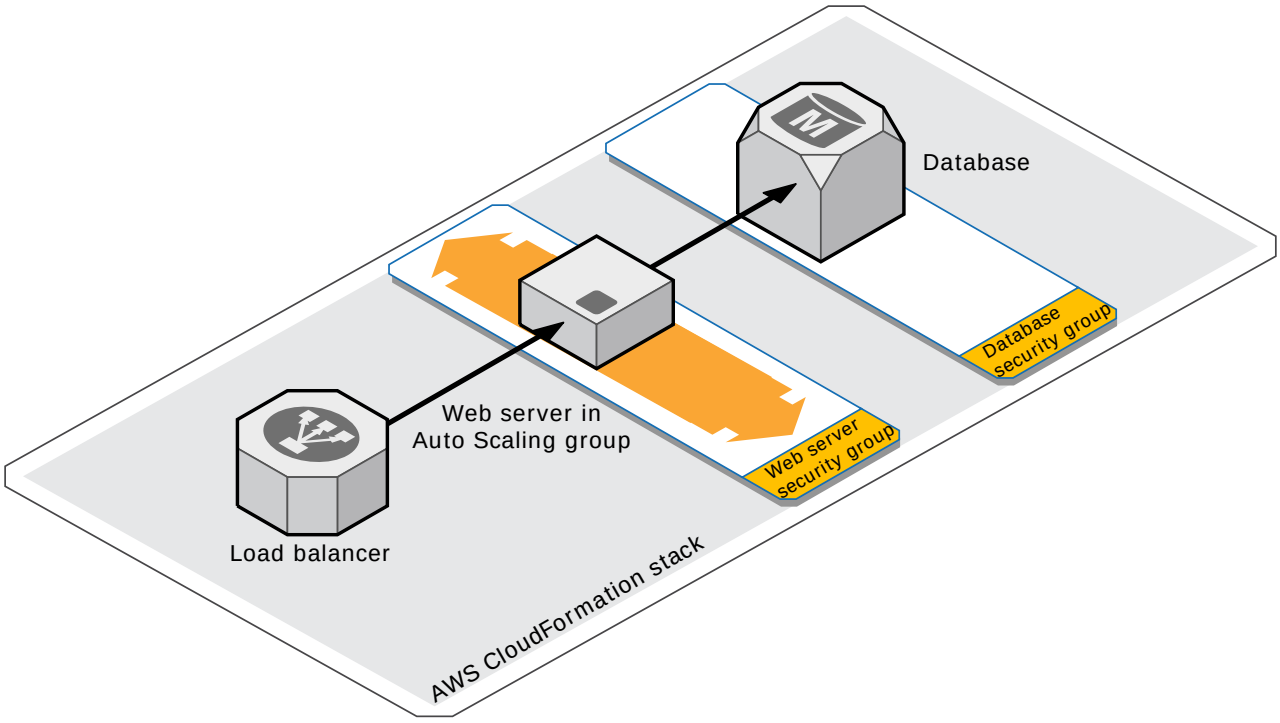
Sample Template Overview

Before diving into the full sample template, consider the overall goal of the template: The template builds a fault-tolerant, load-balancing web server that runs a WordPress blog with a back-end database.

The web server is an Amazon EC2 instance in an Auto Scaling group. Auto Scaling maintains the health of the web server by replacing the web server if it doesn't pass a health check. The Auto Scaling group sits behind a load balancer, which gives the WordPress blog a single endpoint. If we didn't have a load balancer, each time an unhealthy web server is replaced, you would have to use a new endpoint.

The Amazon Relational Database Service (Amazon RDS) database provides the back-end storage that is required for the WordPress blog. It is used to keep persistent data such as blog posts and user comments so that it is not lost if the web server is ever replaced.

The web server and database are in their own security groups, which implement networking rules that filter traffic accordingly. The following diagram outlines the resources that are provisioned by the template:



After the Amazon EC2 instance is started, AWS CloudFormation installs and configures Chef on the instance. It then runs Chef to install and configure WordPress. Several different versions of Chef are available, such as Enterprise Chef and Open Source Chef, but for simplicity, the sample template uses Chef local mode. Chef local mode uses a local Chef repository so that you can use recipes and Chef without a Chef server.

The following template shows the Chef-WordPress sample. The sections that follow break down and describe parts of the template. You can download the template from the following location: https://s3.amazonaws.com/cloudformation-templates-us-east-1/WordPress_Chef.template

Note: If you're interested in how to use Chef with AWS CloudFormation, you can skip to the Web Server section.

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Description" : "AWS CloudFormation Sample Template for WordPress_Chef: WordPress is web software you can use to create a beautiful
website or blog. This template installs a highly available, scalable WordPress deployment using a multi-AZ (Availability Zone) Amazon RDS
database instance for storage. It demonstrates using the AWS CloudFormation bootstrap scripts to deploy the Chef client and using Chef-
client in local mode to deploy WordPress. **WARNING** This template creates an Amazon EC2 instance, an Elastic Load Balancing load
balancer, and an Amazon RDS database instance. You will be billed for the AWS resources used if you create a stack from this template.",

  "Parameters" : {

    "KeyName": {
      "Description" : "Name of an existing EC2 key pair to enable SSH access to the instances",
      "Type": "String",
      "MinLength": "1",
      "MaxLength": "255",
      "AllowedPattern" : "[\\x20-\\x7E]*",
      "ConstraintDescription" : "Can contain only ASCII characters."
    },

    "InstanceType" : {
      "Description" : "Web Server EC2 instance type",
      "Type" : "String",
      "Default" : "m1.small",
      "AllowedValues" : [ "t1.micro", "t2.micro", "t2.small", "t2.medium", "m1.small", "m1.medium", "m1.large", "m1.xlarge", "m2.xlarge",
"m2.2xlarge", "m2.4xlarge", "m3.medium", "m3.large", "m3.xlarge", "m3.2xlarge", "c1.medium", "c1.xlarge", "c3.large", "c3.xlarge",
"c3.2xlarge", "c3.4xlarge", "c3.8xlarge", "g2.2xlarge", "r3.large", "r3.xlarge", "r3.2xlarge", "r3.4xlarge", "r3.8xlarge", "i2.xlarge",
"i2.2xlarge", "i2.4xlarge", "i2.8xlarge", "hi1.4xlarge", "hs1.8xlarge", "cr1.8xlarge", "cc2.8xlarge", "cg1.4xlarge"],
      "ConstraintDescription" : "Must be a valid EC2 instance type."
    },

    "SSHLocation": {
      "Description": " The IP address range that can be used to SSH to the EC2 instances",
      "Type": "String",
      "MinLength": "9",
      "MaxLength": "18",
      "Default": "0.0.0.0/0",
      "AllowedPattern": "(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
      "ConstraintDescription": "Must be a valid IP CIDR range of the form x.x.x.x/x."
    },

    "DBClass" : {
      "Default" : "db.m1.small",
      "Description" : "Database instance class",
      "Type" : "String",
      "AllowedValues" : [ "db.t1.micro", "db.m1.small", "db.m1.medium", "db.m1.large", "db.m1.xlarge", "db.m2.xlarge", "db.m2.2xlarge",
"db.m2.4xlarge", "db.m3.medium", "db.m3.large", "db.m3.xlarge", "db.m3.2xlarge", "db.r3.large", "db.r3.xlarge", "db.r3.2xlarge",
"db.r3.4xlarge", "db.r3.8xlarge", "db.m2.xlarge", "db.m2.2xlarge", "db.m2.4xlarge", "db.cr1.8xlarge"],
      "ConstraintDescription" : "Must select a valid database instance type."
    },

    "DBName" : {
      "Default": "wordpressdb",
      "Description" : "The WordPress database name",
      "Type": "String",
      "MinLength": "1",
      "MaxLength": "64",
      "AllowedPattern" : "[a-zA-Z][a-zA-Z0-9]*",
      "ConstraintDescription" : "Mmust begin with a letter and contain only alphanumeric characters."
    },

    "DBUsername" : {
      "Default": "admin",
      "NoEcho": "true",
      "Description" : "The WordPress database admin account user name",
      "Type": "String",
      "MinLength": "1",
      "MaxLength": "16",
      "AllowedPattern" : "[a-zA-Z][a-zA-Z0-9]*",
      "ConstraintDescription" : "Must begin with a letter and contain only alphanumeric characters."
    },

    "DBPassword" : {
      "Default": "password",
      "NoEcho": "true",
      "Description" : "The WordPress database admin account password",
      "Type": "String",
      "MinLength": "8",
      "MaxLength": "41",
      "AllowedPattern" : "[a-zA-Z0-9]*",
      "ConstraintDescription" : "Must contain only alphanumeric characters."
    },
```

```
    "MultiAZDatabase": {
      "Default": "false",
      "Description" : "Create a multi-AZ MySQL Amazon RDS database instance",
      "Type": "String",
      "AllowedValues" : [ "true", "false" ],
      "ConstraintDescription" : "Must be either true or false."
    },

    "WebServerCapacity": {
      "Default": "1",
      "Description" : "The initial number of web server instances",
      "Type": "Number",
      "MinValue": "1",
      "MaxValue": "5",
      "ConstraintDescription" : "Must be between 1 and 5 EC2 instances."
    },

    "DBAllocatedStorage" : {
      "Default": "5",
      "Description" : "The size of the database (GB)",
      "Type": "Number",
      "MinValue": "5",
      "MaxValue": "1024",
      "ConstraintDescription" : "Must be between 5 and 1024 GB."
    }
  },

  "Mappings" : {
    "AWSInstanceType2Arch" : {
      "t1.micro"    : { "Arch" : "PV64"   },
      "t2.micro"    : { "Arch" : "HVM64"  },
      "t2.small"    : { "Arch" : "HVM64"  },
      "t2.medium"   : { "Arch" : "HVM64"  },
      "m1.small"    : { "Arch" : "PV64"   },
      "m1.medium"   : { "Arch" : "PV64"   },
      "m1.large"    : { "Arch" : "PV64"   },
      "m1.xlarge"   : { "Arch" : "PV64"   },
      "m2.xlarge"   : { "Arch" : "PV64"   },
      "m2.2xlarge"  : { "Arch" : "PV64"   },
      "m2.4xlarge"  : { "Arch" : "PV64"   },
      "m3.medium"   : { "Arch" : "PV64"   },
      "m3.large"    : { "Arch" : "PV64"   },
      "m3.xlarge"   : { "Arch" : "PV64"   },
      "m3.2xlarge"  : { "Arch" : "PV64"   },
      "c1.medium"   : { "Arch" : "PV64"   },
      "c1.xlarge"   : { "Arch" : "PV64"   },
      "c3.large"    : { "Arch" : "PV64"   },
      "c3.xlarge"   : { "Arch" : "PV64"   },
      "c3.2xlarge"  : { "Arch" : "PV64"   },
      "c3.4xlarge"  : { "Arch" : "PV64"   },
      "c3.8xlarge"  : { "Arch" : "PV64"   },
      "g2.2xlarge"  : { "Arch" : "HVMG2"  },
      "r3.large"    : { "Arch" : "HVM64"  },
      "r3.xlarge"   : { "Arch" : "HVM64"  },
      "r3.2xlarge"  : { "Arch" : "HVM64"  },
      "r3.4xlarge"  : { "Arch" : "HVM64"  },
      "r3.8xlarge"  : { "Arch" : "HVM64"  },
      "i2.xlarge"   : { "Arch" : "HVM64"  },
      "i2.2xlarge"  : { "Arch" : "HVM64"  },
      "i2.4xlarge"  : { "Arch" : "HVM64"  },
      "i2.8xlarge"  : { "Arch" : "HVM64"  },
      "hi1.4xlarge" : { "Arch" : "PV64"   },
      "hs1.8xlarge" : { "Arch" : "PV64"   },
      "cr1.8xlarge" : { "Arch" : "HVM64"  },
      "cc2.8xlarge" : { "Arch" : "HVM64"  },
      "cg1.4xlarge" : { "Arch" : "HVMGPU" }
    },

    "AWSRegionArch2AMI" : {
      "us-east-1"      : { "PV64" : "ami-7c807d14", "HVM64" : "ami-76817c1e", "HVMG2" : "ami-9c13ecf4",  "HVMGPU" : "ami-c6867bae"  },
      "us-west-2"      : { "PV64" : "ami-1b3b462b", "HVM64" : "ami-d13845e1", "HVMG2" : "ami-6d8cf15d",  "HVMGPU" : "NOT_SUPPORTED" },
      "us-west-1"      : { "PV64" : "ami-a8d3d4ed", "HVM64" : "ami-f0d3d4b5", "HVMG2" : "ami-84494fc1",  "HVMGPU" : "NOT_SUPPORTED" },
      "eu-west-1"      : { "PV64" : "ami-672ce210", "HVM64" : "ami-892fe1fe", "HVMG2" : "ami-1138f166",  "HVMGPU" : "ami-972fe1e0"  },
      "ap-southeast-1" : { "PV64" : "ami-56b7eb04", "HVM64" : "ami-a6b6eaf4", "HVMG2" : "ami-7a1a4528",  "HVMGPU" : "NOT_SUPPORTED" },
      "ap-northeast-1" : { "PV64" : "ami-25dd9324", "HVM64" : "ami-29dc9228", "HVMG2" : "ami-4d3b734c",  "HVMGPU" : "NOT_SUPPORTED" },
      "ap-southeast-2" : { "PV64" : "ami-6bf99c51", "HVM64" : "ami-d9fe9be3", "HVMG2" : "ami-010c683b",  "HVMGPU" : "NOT_SUPPORTED" },
      "sa-east-1"      : { "PV64" : "ami-c7e649da", "HVM64" : "ami-c9e649d4", "HVMG2" : "NOT_SUPPORTED", "HVMGPU" : "NOT_SUPPORTED" },
      "us-gov-west-1"  : { "PV64" : "ami-ab4a2d88", "HVM64" : "ami-a54a2d86", "HVMG2" : "NOT_SUPPORTED", "HVMGPU" : "NOT_SUPPORTED" },
      "cn-north-1"     : { "PV64" : "ami-cab82af3", "HVM64" : "ami-ccb82af5", "HVMG2" : "NOT_SUPPORTED", "HVMGPU" : "NOT_SUPPORTED" }
    }
  },

  "Resources" : {
```

```
"ElasticLoadBalancer" : {
  "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
  "Metadata" : {
    "Comment1" : "Configure the Load Balancer with a simple health check and cookie-based stickiness",
    "Comment2" : "Use install path for healthcheck to avoid redirects - ELB healthcheck does not handle 302 return codes"
  },
  "Properties" : {
    "AvailabilityZones" : { "Fn::GetAZs" : "" },
    "LBCookieStickinessPolicy" : [ {
      "PolicyName" : "CookieBasedPolicy",
      "CookieExpirationPeriod" : "30"
    } ],
    "Listeners" : [ {
      "LoadBalancerPort" : "80",
      "InstancePort" : "80",
      "Protocol" : "HTTP",
      "PolicyNames" : [ "CookieBasedPolicy" ]
    } ],
    "HealthCheck" : {
      "Target" : "HTTP:80/wp-admin/install.php",
      "HealthyThreshold" : "2",
      "UnhealthyThreshold" : "5",
      "Interval" : "10",
      "Timeout" : "5"
    }
  }
},

"WebServerGroup" : {
  "Type" : "AWS::AutoScaling::AutoScalingGroup",
  "Properties" : {
    "AvailabilityZones" : { "Fn::GetAZs" : "" },
    "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },
    "MinSize" : "1",
    "MaxSize" : "5",
    "DesiredCapacity" : { "Ref" : "WebServerCapacity" },
    "HealthCheckType" : "ELB",
    "LoadBalancerNames" : [ { "Ref" : "ElasticLoadBalancer" } ]
  }
},

"LaunchConfig": {
  "Type" : "AWS::AutoScaling::LaunchConfiguration",
  "Metadata" : {
    "AWS::CloudFormation::Init" : {
      "configSets" : {
        "wordpress_install" : ["install_cfn", "install_chefdk", "install_chef", "install_wordpress", "run_chef"]
      },
      "install_cfn" : {
        "files": {
          "/etc/cfn/cfn-hup.conf": {
            "content": { "Fn::Join": [ "", [
              "[main]\n",
              "stack=", { "Ref": "AWS::StackId" }, "\n",
              "region=", { "Ref": "AWS::Region" }, "\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          },
          "/etc/cfn/hooks.d/cfn-auto-reloader.conf": {
            "content": { "Fn::Join": [ "", [
              "[cfn-auto-reloader-hook]\n",
              "triggers=post.update\n",
              "path=Resources.LaunchConfig.Metadata.AWS::CloudFormation::Init\n",
              "action=/opt/aws/bin/cfn-init ",
                      "         --stack ", { "Ref" : "AWS::StackName" },
                      "         --resource LaunchConfig ",
                      "         --configsets wordpress_install ",
                      "         --region ", { "Ref" : "AWS::Region" }, "\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          }
        },
        "services" : {
          "sysvinit" : {
            "cfn-hup" : {
              "enabled" : "true",
              "ensureRunning" : "true",
              "files" : ["/etc/cfn/cfn-hup.conf", "/etc/cfn/hooks.d/cfn-auto-reloader.conf"]
```

```
            }
          }
        }
      },

      "install_chef" : {
        "sources" : {
          "/var/chef/chef-repo" : "http://github.com/opscode/chef-repo/tarball/master"
        },
        "files" : {
          "/tmp/install.sh" : {
            "source" : "https://www.opscode.com/chef/install.sh",
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          },
          "/var/chef/chef-repo/.chef/knife.rb" : {
            "content" : { "Fn::Join": [ "", [
              "cookbook_path [ '/var/chef/chef-repo/cookbooks' ]\n",
              "node_path [ '/var/chef/chef-repo/nodes' ]\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          },
          "/var/chef/chef-repo/.chef/client.rb" : {
            "content" : { "Fn::Join": [ "", [
              "cookbook_path [ '/var/chef/chef-repo/cookbooks' ]\n",
              "node_path [ '/var/chef/chef-repo/nodes' ]\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          }
        },
        "commands" : {
          "01_make_chef_readable" : {
            "command" : "chmod +rx /var/chef"
          },
          "02_install_chef" : {
            "command" : "bash /tmp/install.sh",
            "cwd"  : "/var/chef"
          },
          "03_create_node_list" : {
            "command" : "chef-client -z -c /var/chef/chef-repo/.chef/client.rb",
            "cwd" : "/var/chef/chef-repo",
            "env" : { "HOME" : "/var/chef" }
          }
        }
      },

      "install_chefdk" : {
        "packages" : {
          "rpm" : {
            "chefdk" : "https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chefdk-0.2.0-2.el6.x86_64.rpm"
          }
        }
      },

      "install_wordpress" : {
        "files" : {
          "/var/chef/chef-repo/.chef/knife.rb" : {
            "content" : { "Fn::Join": [ "", [
              "cookbook_path [ '/var/chef/chef-repo/cookbooks/wordpress/berks-cookbooks' ]\n",
              "node_path [ '/var/chef/chef-repo/nodes' ]\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          },
          "/var/chef/chef-repo/.chef/client.rb" : {
            "content" : { "Fn::Join": [ "", [
              "cookbook_path [ '/var/chef/chef-repo/cookbooks/wordpress/berks-cookbooks' ]\n",
              "node_path [ '/var/chef/chef-repo/nodes' ]\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          },
          "/var/chef/chef-repo/cookbooks/wordpress/attributes/aws_rds_config.rb" : {
            "content": { "Fn::Join": [ "", [
              "normal['wordpress']['db']['pass'] = '", {"Ref" : "DBPassword"}, "'\n",
              "normal['wordpress']['db']['user'] = '", {"Ref" : "DBUsername"}, "'\n",
```

```
              "normal['wordpress']['db']['host'] = '", {"Fn::GetAtt" : ["DBInstance", "Endpoint.Address"]}, "'\n",
              "normal['wordpress']['db']['name'] = '", {"Ref" : "DBName"}, "'\n"
            ]]},
            "mode"  : "000400",
            "owner" : "root",
            "group" : "root"
          }
        },
        "commands" : {
          "01_get_cookbook" : {
            "command" : "knife cookbook site download wordpress",
            "cwd" : "/var/chef/chef-repo",
            "env" : { "HOME" : "/var/chef" }
          },
          "02_unpack_cookbook" : {
            "command" : "tar xvfz /var/chef/chef-repo/wordpress*",
            "cwd" : "/var/chef/chef-repo/cookbooks"
          },
          "03_init_berkshelf": {
            "command" : "berks init /var/chef/chef-repo/cookbooks/wordpress --skip-vagrant --skip-git",
            "cwd" : "/var/chef/chef-repo/cookbooks/wordpress",
            "env" : { "HOME" : "/var/chef" }
          },
          "04_vendorize_berkshelf" : {
            "command" : "berks vendor",
            "cwd" : "/var/chef/chef-repo/cookbooks/wordpress",
            "env" : { "HOME" : "/var/chef" }
          },
          "05_configure_node_run_list" : {
            "command" : "knife node run_list add -z `knife node list -z` recipe[wordpress]",
            "cwd" : "/var/chef/chef-repo",
            "env" : { "HOME" : "/var/chef" }
          }
        }
      },

      "run_chef" : {
        "commands" : {
          "01_run_chef_client" : {
            "command" : "chef-client -z -c /var/chef/chef-repo/.chef/client.rb",
            "cwd" : "/var/chef/chef-repo",
            "env" : { "HOME" : "/var/chef" }
          }
        }
      }
    }
  },
  "Properties": {
    "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" : "AWS::Region" },
                    { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" : "InstanceType" }, "Arch" ] } ] },
    "InstanceType"   : { "Ref" : "InstanceType" },
    "SecurityGroups" : [ {"Ref" : "WebServerSecurityGroup"} ],
    "KeyName"        : { "Ref" : "KeyName" },
    "UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
                   "#!/bin/bash -xe\n",
                   "yum update aws-cfn-bootstrap\n",

                   "/opt/aws/bin/cfn-init ",
                   "         --stack ", { "Ref" : "AWS::StackName" },
                   "         --resource LaunchConfig ",
                   "         --configsets wordpress_install ",
                   "         --region ", { "Ref" : "AWS::Region" }, "\n",

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

"WebServerWaitHandle" : {
  "Type" : "AWS::CloudFormation::WaitConditionHandle"
},

"WebServerWaitCondition" : {
  "Type" : "AWS::CloudFormation::WaitCondition",
  "DependsOn" : "WebServerGroup",
  "Properties" : {
    "Handle" : {"Ref" : "WebServerWaitHandle"},
    "Timeout" : "900"
  }
},

"DBInstance" : {
  "Type": "AWS::RDS::DBInstance",
```

```
      "Properties": {
        "DBName"            : { "Ref" : "DBName" },
        "Engine"            : "MySQL",
        "MultiAZ"           : { "Ref": "MultiAZDatabase" },
        "MasterUsername"    : { "Ref" : "DBUsername" },
        "DBInstanceClass"   : { "Ref" : "DBClass" },
        "DBSecurityGroups"  : [{ "Ref" : "DBSecurityGroup" }],
        "AllocatedStorage"  : { "Ref" : "DBAllocatedStorage" },
        "MasterUserPassword": { "Ref" : "DBPassword" }
      }
    },

    "DBSecurityGroup": {
      "Type": "AWS::RDS::DBSecurityGroup",
      "Properties": {
        "DBSecurityGroupIngress": { "EC2SecurityGroupName": { "Ref": "WebServerSecurityGroup"} },
        "GroupDescription"      : "Front-end access"
      }
    },


    "WebServerSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable HTTP access via port 80 locked down to the load balancer + SSH access",
        "SecurityGroupIngress" : [
          {"IpProtocol" : "tcp", "FromPort" : "80", "ToPort" : "80",
"SourceSecurityGroupOwnerId" : {"Fn::GetAtt" : ["ElasticLoadBalancer", "SourceSecurityGroup.OwnerAlias"]},"SourceSecurityGroupName" :
{"Fn::GetAtt" : ["ElasticLoadBalancer", "SourceSecurityGroup.GroupName"]}},
          {"IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : { "Ref" : "SSHLocation"}}
        ]
      }
    }
  },

  "Outputs" : {
    "WebsiteURL" : {
      "Value" : { "Fn::Join" : ["", ["http://", { "Fn::GetAtt" : [ "ElasticLoadBalancer", "DNSName" ]}]]},
      "Description" : "WordPress website"
    }
  }
}
```

Before Launching the Template

Before you can launch the sample template, you must have a valid Amazon EC2 key pair in the region that you are launching the template in. The key pair is used to authenticate users when they use SSH to access instances. For more information about creating an Amazon EC2 key pair, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.[1]

Parameters

In some cases, you might have resource properties that can change from stack to stack, or you might have sensitive information that you don't want to include in a template. For example, you might specify different key pair names for different regions, or you might specify a database password that you don't want visible in a template. Instead of hard coding these values in the template, you can use input parameters, which you specify when you create a stack. These input parameters are declared in the Parameters section of the template. For example, the following snippet describes a database password parameter:

[1] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

```
        "DBPassword": {
            "Default": "password",
            "NoEcho": "true",
            "Description": "The WordPress database admin account password",
            "Type": "String",
            "MinLength": "8",
            "MaxLength": "41",
            "AllowedPattern": "[a-zA-Z0-9]*",
            "ConstraintDescription": "Must contain only alphanumeric characters."
        }
```

In the parameter's properties, we declare the parameter type, a description, and constraints that the password must follow. The sample also sets the `NoEcho` property to `true` so that the value is obfuscated whenever the parameter value is displayed. Later in the template, the `DBPassword` parameter is used to set the password for the Amazon RDS database instance.

Mappings

You can use the Mappings section to pick values that depend on the stack's context. For example, the following snippet uses the stack's region and the instance architecture to select the correct AMI ID:

```
    "AWSRegionArch2AMI" : {
      "us-east-1"      : { "PV64" : "ami-7c807d14", "HVM64" : "ami-76817c1e", "HVMG2" : "ami-9c13ecf4",  "HVMGPU" : "ami-c6867bae"  },
      "us-west-2"      : { "PV64" : "ami-1b3b462b", "HVM64" : "ami-d13845e1", "HVMG2" : "ami-6d8cf15d",  "HVMGPU" : "NOT_SUPPORTED" },
      "us-west-1"      : { "PV64" : "ami-a8d3d4ed", "HVM64" : "ami-f0d3d4b5", "HVMG2" : "ami-84494fc1",  "HVMGPU" : "NOT_SUPPORTED" },
      "eu-west-1"      : { "PV64" : "ami-672ce210", "HVM64" : "ami-892fe1fe", "HVMG2" : "ami-1138f166",  "HVMGPU" : "ami-972fe1e0"  },
      "ap-southeast-1" : { "PV64" : "ami-56b7eb04", "HVM64" : "ami-a6b6eaf4", "HVMG2" : "ami-7a1a4528",  "HVMGPU" : "NOT_SUPPORTED" },
      "ap-northeast-1" : { "PV64" : "ami-25dd9324", "HVM64" : "ami-29dc9228", "HVMG2" : "ami-4d3b734c",  "HVMGPU" : "NOT_SUPPORTED" },
      "ap-southeast-2" : { "PV64" : "ami-6bf99c51", "HVM64" : "ami-d9fe9be3", "HVMG2" : "ami-010c683b",  "HVMGPU" : "NOT_SUPPORTED" },
      "sa-east-1"      : { "PV64" : "ami-c7e649da", "HVM64" : "ami-c9e649d4", "HVMG2" : "NOT_SUPPORTED", "HVMGPU" : "NOT_SUPPORTED" },
      "us-gov-west-1"  : { "PV64" : "ami-ab4a2d88", "HVM64" : "ami-a54a2d86", "HVMG2" : "NOT_SUPPORTED", "HVMGPU" : "NOT_SUPPORTED" },
      "cn-north-1"     : { "PV64" : "ami-cab82af3", "HVM64" : "ami-ccb82af5", "HVMG2" : "NOT_SUPPORTED", "HVMGPU" : "NOT_SUPPORTED" }
    }
```

Later in the template, we use this mapping to specify the AMI ID for Amazon EC2 instances in the launch configuration. Given the region and instance architecture, we use the `FindInMap` intrinsic function to look up the correct AMI ID.

Load Balancer (Elastic Load Balancing)

If you have multiple instances in an Auto Scaling group, the load balancer evenly distributes incoming traffic among all instances. The load balancer routinely checks the health of registered instances by sending requests to the specified target. If the Elastic Load Balancing service determines that an instance is unhealthy, Auto Scaling can terminate the instance and launch a new one to replace it. You can also use the load balancer to maintain a single endpoint, as instances in the Auto Scaling group come and go. The following snippet describes the load balancer's properties:

```
    "ElasticLoadBalancer" : {
      "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
      "Metadata" : {
        "Comment1" : "Configure the Load Balancer with a simple health check and cookie-based stickiness",
        "Comment2" : "Use install path for healthcheck to avoid redirects - ELB healthcheck does not handle 302 return codes"
      },
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : "" },
        "LBCookieStickinessPolicy" : [ {
          "PolicyName" : "CookieBasedPolicy",
          "CookieExpirationPeriod" : "30"
        } ],
        "Listeners" : [ {
          "LoadBalancerPort" : "80",
          "InstancePort" : "80",
          "Protocol" : "HTTP",
          "PolicyNames" : [ "CookieBasedPolicy" ]
        } ],
        "HealthCheck" : {
          "Target" : "HTTP:80/wp-admin/install.php",
          "HealthyThreshold" : "2",
          "UnhealthyThreshold" : "5",
          "Interval" : "10",
          "Timeout" : "5"
        }
      }
    },
```

- `LBCookieStickinessPolicy`

  Enables the load balancer to bind a user's session to a specific instance. When the load balancer receives a request, it first checks to see if a load-balancer-generated cookie is present in the request. If so, the request is sent to the instance that is specified in the cookie. For this sample template, the sticky session duration is 30 seconds.

- `Listeners`

  Describes the type of traffic that the load balancer listens for. For this sample template, the load balancer listens for HTTP traffic on port 80.

- `HealthCheck`

  Describes the settings for the health check, such as the interval between health checks, the target to ping, and the number of successes and failures required before an instance is determined to be healthy or unhealthy. The health check is a request sent by the load balancer to an instance. If the target responds with an HTTP status code 200, the health check passes. In this sample, health checks are done in ten second intervals. An instance is declared healthy if it passes two consecutive health checks and unhealthy if it fails five consecutive health checks. Also, if the target doesn't respond after five seconds, the health check fails.

Web Server (Auto Scaling Group and Launch Configuration)

An Auto Scaling group manages the minimum and maximum number of Amazon EC2 instances in a group. For example, if CPU utilization increases past a certain threshold, the Auto Scaling group could launch more instances to handle the additional load. In this scenario, the sample template specifies a desired group size of one instance. You could declare a single Amazon EC2 instance without an Auto Scaling group. However, Auto Scaling can automatically stop unhealthy instances and launch new instances, ensuring that the WordPress

blog isn't down for long periods of time. The following snippet describes the properties of the Auto Scaling group:

```
    "WebServerGroup" : {
      "Type" : "AWS::AutoScaling::AutoScalingGroup",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : "" },
        "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },
        "MinSize" : "1",
        "MaxSize" : "5",
        "DesiredCapacity" : { "Ref" : "WebServerCapacity" },
        "HealthCheckType" : "ELB",
        "LoadBalancerNames" : [ { "Ref" : "ElasticLoadBalancer" } ]
      }
    }
```

- `LaunchConfigurationName` and `LoadBalancerNames`

  Specifies the launch configuration and load balancer to associate with this Auto Scaling group. Because the launch configuration and load balancer are defined in other parts of the template, the sample template uses the `Ref` function to point to those resources.

- `HealthCheckType`

  Specifies whether to use just the Auto Scaling health check (the Amazon EC2 instance status checks) or both the Auto Scaling and load balancer health check. Because the sample template includes a load balancer, we include the load balancer in the health check.

- `MinSize` and `MaxSize`

  Specifies the minimum and maximum number of instances in the Auto Scaling group. These properties are more relevant when you add Auto Scaling policies. For example, you can have an Auto Scaling policy that increases or decreases the number of instances, depending on their CPU load.

- `DesiredCapacity`

  Specifies the number of instances that you want Auto Scaling to have in service. The sample template defaults to a value of 1, which provides a fault-tolerant web server. If the instance becomes unhealthy Auto Scaling can terminate it and launch a new one. You can increase the number of instances, depending on the web traffic that you expect.

- `AvailabilityZones`

  A list of Availability Zones in which instances can be launched.

A launch configuration describes the Amazon EC2 instances that are to be launched in the Auto Scaling group. We can specify properties such as the instance type, the AMI ID, and bootstrapping information. The instance properties are shown in the following snippet:

```
      "Properties": {
        "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" : "AWS::Region" },
                          { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" : "InstanceType" }, "Arch" ] } ] },
        "InstanceType"   : { "Ref" : "InstanceType" },
        "KeyName"        : { "Ref" : "KeyName" },
        "SecurityGroups" : [ {"Ref" : "WebServerSecurityGroup"} ],
        "UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
                       "#!/bin/bash –xe\n",
                       "yum update aws-cfn-bootstrap\n",

                       "/opt/aws/bin/cfn-init ",
                       "         --stack ", { "Ref" : "AWS::StackName" },
                       "         --resource LaunchConfig ",
                       "         --configsets wordpress_install ",
                       "         --region ", { "Ref" : "AWS::Region" }, "\n",

                       "/opt/aws/bin/cfn-signal –e $? '", { "Ref" : "WebServerWaitHandle" }, "'\n"
        ]]}}
      }
    },
```

- `ImageId`

  Specifies the AMI ID, which is dependent on the AWS region that the stack is launched in and the instance type. The AMI IDs are mapped in the Mapping section of the template.

- `InstanceType`

  Specifies the type of instance to use, such as t2.micro or m1.small. The value is whatever was specified for the `InstanceType` input parameter.

- `KeyName`

  Specifies the Amazon EC2 key pair name to use. The value is whatever was specified for the `KeyName` input parameter.

- `SecurityGroups`

  Specifies the security group (the firewall rules) to use. The security group is defined by the WebServerSecurityGroup resource, which is defined elsewhere in the template.

- `UserData`

  Data sent to the instance. In this scenario, we're sending a shell script to the instance. The script updates the AWS CloudFormation bootstrapping tools and then calls cfn-init. The cfn-init helper script reads the template metadata in the `AWS::CloudFormation::Init` resource and performs tasks that are based on that metadata. You must specify the stack name and the logical ID of the resource that contains the metadata to use. In this sample, we also specify the configuration set to use. After the cfn-init helper script finishes successfully, a signal is sent to the wait handle, and stack creation proceeds.

In addition to the instance properties, the launch configuration includes a metadata section with the `AWS::CloudFormation::Init` resource. The `AWS::CloudFormation::Init` resource defines a set of tasks to perform when used in combination with the `cfn-init` helper script, such as installing packages or writing files to disk. In the sample template, we use both to download and install Chef, download a WordPress cookbook, and install WordPress.

In the `AWS::CloudFormation::Init` resource, a group of tasks are contained within a *configuration*. You can have multiple configurations for cases where you might have tasks that have prerequisite tasks. To determine which configurations to use and in what order to run them, you list configurations in a *configuration set*. The following snippet shows the configuration sets for the WordPress sample template, which includes five different configurations:

```
        "configSets" : {
          "wordpress_install" : ["install_cfn", "install_chefdk", "install_chef", "install_wordpress", "run_chef"]
        },
```

A single configuration can consists of one or more sections that are run in the following order: packages, groups, users, sources, files, commands, and services. Each section performs a specific task. For example, in the packages section, you can install apt, msi, python, rpm, rubygems, and yum packages. For more information about each section, see `AWS::CloudFormation::Init` in the *AWS CloudFormation User Guide*.[2]

Consider a sample configuration: The first configuration is the `install_cfn` configuration, which starts a daemon (`cfn-hup`) that monitors any changes in the launch configuration metadata. The configuration also creates two configuration files: `/etc/cfn/cfn-hup.conf` and `/etc/cfn/hooks.d/cfn-auto-reloader.conf`. If the metadata in the stack that is specified in the `cfn-hup.conf` file changes (like after a stack update), the daemon detects that change and then performs the specified user action in the `cfn-auto-reloader.conf` file (rerun the `cfn-init` helper script).

```
        "install_cfn" : {
          "files": {
            "/etc/cfn/cfn-hup.conf": {
              "content": { "Fn::Join": [ "", [
                "[main]\n",
                "stack=", { "Ref": "AWS::StackId" }, "\n",
                "region=", { "Ref": "AWS::Region" }, "\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            },
            "/etc/cfn/hooks.d/cfn-auto-reloader.conf": {
              "content": { "Fn::Join": [ "", [
                "[cfn-auto-reloader-hook]\n",
                "triggers=post.update\n",
                "path=Resources.LaunchConfig.Metadata.AWS::CloudFormation::Init\n",
                "action=/opt/aws/bin/cfn-init ",
                        "         --stack ", { "Ref" : "AWS::StackName" },
                        "         --resource LaunchConfig ",
                        "         --configsets wordpress_install ",
                        "         --region ", { "Ref" : "AWS::Region" }, "\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            }
          },
          "services" : {
            "sysvinit" : {
              "cfn-hup" : {
                "enabled" : "true",
                "ensureRunning" : "true",
                "files" : ["/etc/cfn/cfn-hup.conf", "/etc/cfn/hooks.d/cfn-auto-reloader.conf"]
              }
            }
          }
        }
```

[2] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-init.html

Using the `packages` section, the `install_chefdk` configuration downloads and installs the Chef Development Kit rpm package. To specify the exact rpm package, we point to a specific URL that was obtained from the Chef website (http://downloads.getchef.com/chef-dk/redhat/#/). The Chef Development Kit includes additional tools that can help you work with Chef, such as Berkshelf.

```
        "install_chefdk" : {
          "packages" : {
            "rpm" : {
              "chefdk" : "https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chefdk-0.2.0-2.el6.x86_64.rpm"
            }
          }
        },
```

In the `install_chef` configuration, the `sources` section sets up a local Chef repository on the instance. The `files` section writes files on the instance: a Chef installation file (`install.sh`), a Knife configuration file (`knife.rb`), and a Chef client configuration file (`client.rb`). The source of the installation file is a URL that was obtained from the Chef website. The configuration files specify the default Chef cookbook and node paths.

The `commands` section runs the specified commands on the instance. We make the `/var/chef` directory readable, run the Chef installation, and then start Chef local mode by using the `client.rb` file that was created in the `files` section. The commands are run in alphanumeric order according to their command name, so we prefix the command names with the number in which we want to run the commands.

```
        "install_chef" : {
          "sources" : {
            "/var/chef/chef-repo" : "http://github.com/opscode/chef-repo/tarball/master"
          },
          "files" : {
            "/tmp/install.sh" : {
              "source" : "https://www.opscode.com/chef/install.sh",
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            },
            "/var/chef/chef-repo/.chef/knife.rb" : {
              "content" : { "Fn::Join": [ "", [
                "cookbook_path [ '/var/chef/chef-repo/cookbooks' ]\n",
                "node_path [ '/var/chef/chef-repo/nodes' ]\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            },
            "/var/chef/chef-repo/.chef/client.rb" : {
              "content" : { "Fn::Join": [ "", [
                "cookbook_path [ '/var/chef/chef-repo/cookbooks' ]\n",
                "node_path [ '/var/chef/chef-repo/nodes' ]\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            }
          },
          "commands" : {
            "01_make_chef_readable" : {
              "command" : "chmod +rx /var/chef"
            },
            "02_install_chef" : {
              "command" : "bash /tmp/install.sh",
              "cwd"  : "/var/chef"
            },
            "03_create_node_list" : {
              "command" : "chef-client -z -c /var/chef/chef-repo/.chef/client.rb",
              "cwd" : "/var/chef/chef-repo",
              "env" : { "HOME" : "/var/chef" }
            }
          }
        },
```

The `install_wordpress` configuration installs WordPress by using a WordPress cookbook. In the `files` section, the `knife.rb` and `client.rb` files are overwritten to point to the cookbooks that are required to install WordPress. The `aws_rds_config.rb` attribute specifies the Amazon RDS database instance as the WordPress database.

In the `commands` section, we first download a WordPress community cookbook that is hosted on Chef Supermarket. Next, we unpackage the WordPress cookbook in the `/var/chef/chef-repo/cooks/wordpress` directory. Then we initialize Berkshelf for the WordPress cookbook so that we can use Berkshelf to manage all of the dependent cookbooks. To install dependent cookbooks to disk, we use the `vendor` command. The source and dependent cookbooks are described in the WordPress cookbook's Berksfile. Finally, we add the WordPress cookbook to the node list that was created by the `install_chef` configuration.

```
        "install_wordpress" : {
          "files" : {
            "/var/chef/chef-repo/.chef/knife.rb" : {
              "content" : { "Fn::Join": [ "", [
                "cookbook_path [ '/var/chef/chef-repo/cookbooks/wordpress/berks-cookbooks' ]\n",
                "node_path [ '/var/chef/chef-repo/nodes' ]\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            },
            "/var/chef/chef-repo/.chef/client.rb" : {
              "content" : { "Fn::Join": [ "", [
                "cookbook_path [ '/var/chef/chef-repo/cookbooks/wordpress/berks-cookbooks' ]\n",
                "node_path [ '/var/chef/chef-repo/nodes' ]\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            },
            "/var/chef/chef-repo/cookbooks/wordpress/attributes/aws_rds_config.rb" : {
              "content": { "Fn::Join": [ "", [
                "normal['wordpress']['db']['pass'] = '", {"Ref" : "DBPassword"}, "'\n",
                "normal['wordpress']['db']['user'] = '", {"Ref" : "DBUsername"}, "'\n",
                "normal['wordpress']['db']['host'] = '", {"Fn::GetAtt" : ["DBInstance", "Endpoint.Address"]}, "'\n",
                "normal['wordpress']['db']['name'] = '", {"Ref" : "DBName"}, "'\n"
              ]]},
              "mode"  : "000400",
              "owner" : "root",
              "group" : "root"
            }
          },
          "commands" : {
            "01_get_cookbook" : {
              "command" : "knife cookbook site download wordpress",
              "cwd" : "/var/chef/chef-repo",
              "env" : { "HOME" : "/var/chef" }
            },
            "02_unpack_cookbook" : {
              "command" : "tar xvfz /var/chef/chef-repo/wordpress*",
              "cwd" : "/var/chef/chef-repo/cookbooks"
            },
            "03_init_berkshelf": {
              "command" : "berks init /var/chef/chef-repo/cookbooks/wordpress --skip-vagrant --skip-git",
              "cwd" : "/var/chef/chef-repo/cookbooks/wordpress",
              "env" : { "HOME" : "/var/chef" }
            },
            "04_vendorize_berkshelf" : {
              "command" : "berks vendor",
              "cwd" : "/var/chef/chef-repo/cookbooks/wordpress",
              "env" : { "HOME" : "/var/chef" }
            },
            "05_configure_node_run_list" : {
              "command" : "knife node run_list add -z `knife node list -z` recipe[wordpress]",
              "cwd" : "/var/chef/chef-repo",
              "env" : { "HOME" : "/var/chef" }
            }
          }
        },
```

The `run_chef` configuration runs the Chef client in local mode to install and configure WordPress. We use the `client.rb` file to specify which recipes to run. This file is the same file we created in the `install_wordpress` configuration.

```
        "run_chef" : {
          "commands" : {
            "01_run_chef_client" : {
              "command" : "chef-client -z -c /var/chef/chef-repo/.chef/client.rb",
              "cwd" : "/var/chef/chef-repo",
              "env" : { "HOME" : "/var/chef" }
            }
          }
        }
```

Back-end Database (Amazon RDS)

WordPress requires a database to store and retrieve data, such as posts, pages, comments, and other site options. For this scenario, the template declares an Amazon RDS MySQL database instance with 5 GB of storage. Many of the database properties are specified by inputs, which are declared in the parameters section of the sample template.

```
    "DBInstance" : {
      "Type": "AWS::RDS::DBInstance",
      "Properties": {
        "DBName"            : { "Ref" : "DBName" },
        "Engine"            : "MySQL",
        "MultiAZ"           : { "Ref": "MultiAZDatabase" },
        "MasterUsername"    : { "Ref" : "DBUsername" },
        "DBInstanceClass"   : { "Ref" : "DBClass" },
        "DBSecurityGroups"  : [{ "Ref" : "DBSecurityGroup" }],
        "AllocatedStorage"  : { "Ref" : "DBAllocatedStorage" },
        "MasterUserPassword": { "Ref" : "DBPassword" }
      }
    }
```

Security Groups

The sample template applies rules that limit incoming traffic to the web server and the database. The web server is locked down to receive traffic only from the load balancer or from an SSH connection so that it cannot be directly accessed from the Internet. The database can receive traffic only from the web server. The load balancer can receive traffic from the Internet; it does not require any rules.

For the web server and database instance, the template uses security groups to define inbound and outbound rules. However, in the sample template, we only use inbound rules to limit traffic. For example, only inbound HTTP traffic from the load balancer on port 80 and traffic from the specified CIDR range on port 22 can reach the web server, as shown in the following snippet:

```
    "WebServerSecurityGroup" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Enable HTTP access via port 80 locked down to the load balancer + SSH access",
        "SecurityGroupIngress" : [
          {
            "IpProtocol" : "tcp",
            "FromPort" : "80",
            "ToPort" : "80",
            "SourceSecurityGroupOwnerId" : {"Fn::GetAtt" : ["ElasticLoadBalancer", "SourceSecurityGroup.OwnerAlias"]},
            "SourceSecurityGroupName" : {"Fn::GetAtt" : ["ElasticLoadBalancer", "SourceSecurityGroup.GroupName"]}
          },
          {
            "IpProtocol" : "tcp",
            "FromPort" : "22",
            "ToPort" : "22",
            "CidrIp" : { "Ref" : "SSHLocation"}
          }
        ]
      }
    }
```

The Elastic Load Balancing service will create a security group for the load balancer if no security group is specified when the load balancer is created.

Summary

Not only can AWS CloudFormation set up and configure your AWS resources, you can also integrate it with other tools, such as Chef, to help bootstrap your instances. This integration can help you reliably and consistently deploy any web application on your Amazon EC2 instances. The Chef-WordPress sample template is just one example, which you can use it as a starting point to run your own recipes.

You can learn more about AWS CloudFormation by viewing the AWS CloudFormation User Guide.[3]

[3] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html