Automate your Cloud Operations

Operations

What is Operations?

In the IT world, Operations refers to a team or department within IT which is responsible for the running of a business' IT systems and infrastructure.

So what kind of activities this team perform on day to day basis?

Building, modifying, provisioning, updating systems, software and infrastructure to keep them available, performing and secure which ensures that users can be as productive as possible.

When moving to public cloud platforms the areas of focus for Operations are:

Cost reduction: if we design it properly and apply good practices when managing it (scale down / switch off)

Smarter operation: Use of Automation and APIs

Agility: faster in provisioning infrastructure or environments by Automating the everything

Better Uptime: Plan for failover, and design effective DR solutions more cost effectively.

If Cloud is the new normal then Automation is the new normal.

For this blog post we will focus on automation using AWS CloudFormation. The template I will use for this post for educational purposes only and may not be suitable for production workloads :).

AWS CloudFormation

AWS CloudFormation provides developers and ~~system administrators~~ DevOps an easy way to create and manage a collection of related AWS resources, including provisioning and updating them in an orderly and predictable fashion. AWS provides various CloudFormation templates, snippets and reference implementations.

Let's talk about versioning before diving deeper into CloudFormation. It is extremely important to version your AWS infrastructure in the same way as you version your software. Versioning will help you to track change within your infrastructure by identifying:

What changed?

Who changed it?

When was it changed?

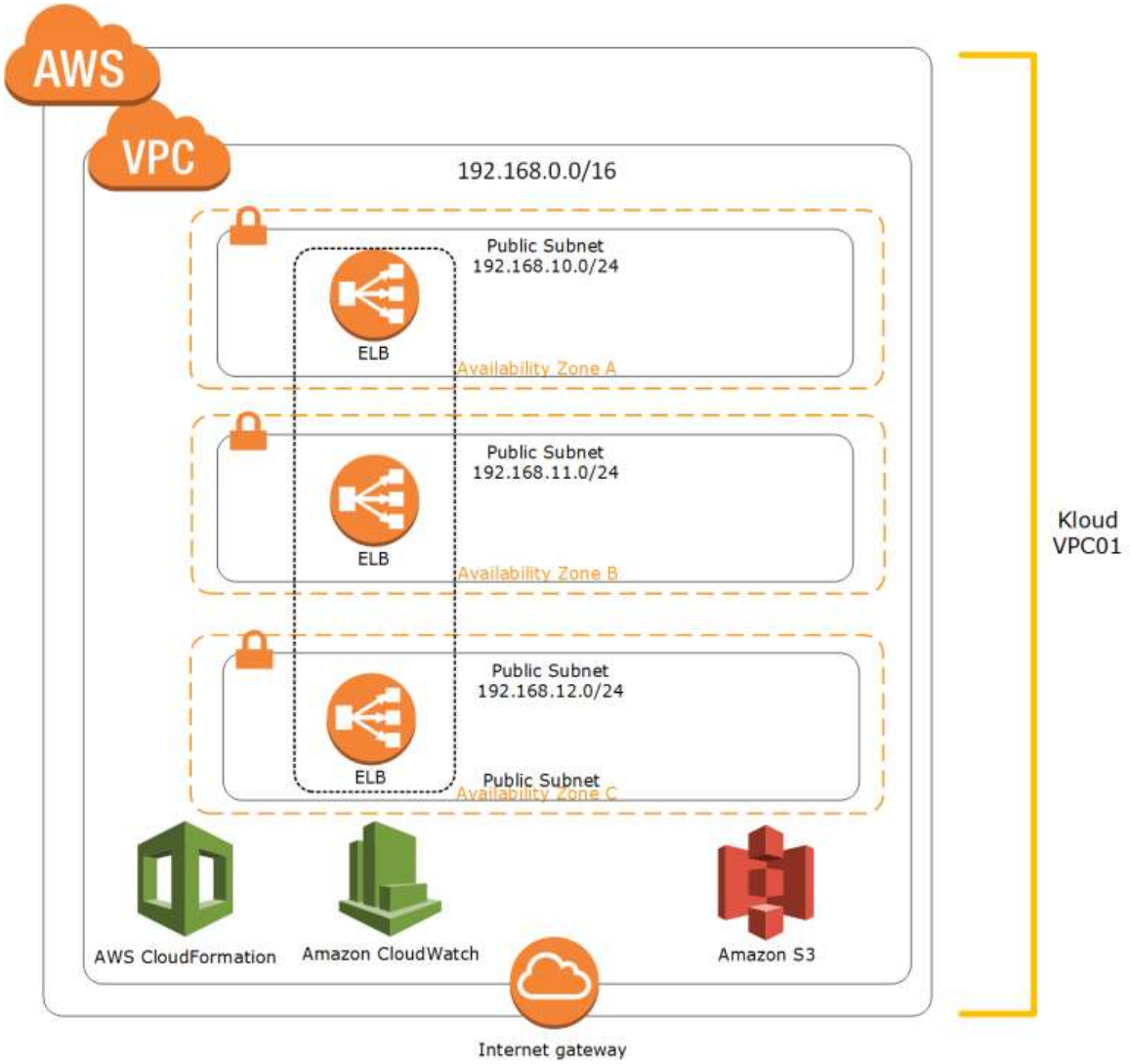
Why was it changed?

You can tie this version to a service management or project delivery tools if you wish.

You should also put your templates into source control. Personally I am using Github to version my infrastructure code, but any system such as Team Foundation Server (TFS) will do.

AWS Infrastructure

The below diagram illustrates the basic AWS infrastructure we will build and automate for



Initial Stack

Firstly we will create the initial stack. Below are the components for the initial stack:

A VPC with CIDR block of 192.168.0.0/16 : 65,543 IPs

Three Public Subnets across 3 Availability Zones : 192.168.10.0/24, 192.168.11.0/24, 192.168.12.0/24

An Internet Gateway attached to the VPC to allow public Internet access. This is a routing construct for VPC and not an EC2 instance

Routes and Route tables for three public subnets so EC2 instances in those public subnets can communicate

Default Network ACLs to allow all communication inside of the VPC.

Below is the CloudFormation template to build the initial stack.

```
{
"AWSTemplateFormatVersion" : "2010-09-09",
"Description" : "Builds a VPC w/ INET Gateway and 3 public subnets. **WARNING** This template creates Amazon EC2 instance(s). You will be billed for the AWS resources used if you create a stack from this template.",
"Resources" : {
"VPC" : {
"Type" : "AWS::EC2::VPC",
"Properties" : {
"CidrBlock" : "192.168.0.0/16",
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Public" }
]
}
},
"PublicSubnetA" : {
"Type" : "AWS::EC2::Subnet",
"Properties" : {
"VpcId" : { "Ref" : "VPC" },
"CidrBlock" : "192.168.10.0/24",
"AvailabilityZone" : { "Fn::Select": [ "0", {"Fn::GetAZs": {"Ref": "AWS::Region"}} ]},
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Public" }
]
}
```

```
},
"PublicSubnetB" : {
"Type" : "AWS::EC2::Subnet",
"Properties" : {
"VpcId" : { "Ref" : "VPC" },
"CidrBlock" : "192.168.11.0/24",
"AvailabilityZone" : { "Fn::Select": [ "1", {"Fn::GetAZs": {"Ref": "AWS::Region"}} ]},
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Public" }
]
}
},
"PublicSubnetC" : {
"Type" : "AWS::EC2::Subnet",
"Properties" : {
"VpcId" : { "Ref" : "VPC" },
"CidrBlock" : "192.168.12.0/24",
"AvailabilityZone" : { "Fn::Select": [ "2", {"Fn::GetAZs": {"Ref": "AWS::Region"}} ]},
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Public" }
]
}
},
"InternetGateway" : {
"Type" : "AWS::EC2::InternetGateway",
```

```
"Properties" : {
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Public" }
]
}
},
"AttachGateway" : {
"Type" : "AWS::EC2::VPCGatewayAttachment",
"Properties" : {
"VpcId" : { "Ref" : "VPC" },
"InternetGatewayId" : { "Ref" : "InternetGateway" }
}
},
"PublicRouteTable" : {
"Type" : "AWS::EC2::RouteTable",
"Properties" : {
"VpcId" : {"Ref" : "VPC"},
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Public" }
]
}
},
"PublicRoute" : {
"Type" : "AWS::EC2::Route",
"Properties" : {
```

```
"RouteTableId" : { "Ref" : "PublicRouteTable" },
"DestinationCidrBlock" : "0.0.0.0/0",
"GatewayId" : { "Ref" : "InternetGateway" }
}
},
"PublicSubnetRouteTableAssociationA" : {
"Type" : "AWS::EC2::SubnetRouteTableAssociation",
"Properties" : {
"SubnetId" : { "Ref" : "PublicSubnetA" },
"RouteTableId" : { "Ref" : "PublicRouteTable" }
}
},
"PublicSubnetRouteTableAssociationB" : {
"Type" : "AWS::EC2::SubnetRouteTableAssociation",
"Properties" : {
"SubnetId" : { "Ref" : "PublicSubnetB" },
"RouteTableId" : { "Ref" : "PublicRouteTable" }
}
},
"PublicSubnetRouteTableAssociationC" : {
"Type" : "AWS::EC2::SubnetRouteTableAssociation",
"Properties" : {
"SubnetId" : { "Ref" : "PublicSubnetC" },
"RouteTableId" : { "Ref" : "PublicRouteTable" }
}
}
},
```

```
"Outputs" : {
"VpcId" : {
"Value" : {"Ref" : "VPC"},
"Description" : "VPC ID of newly created VPC"
},
"PublicSubnetA" : {
"Value" : {"Ref" : "PublicSubnetA"},
"Description" : "Public Subnet in AZ A"
},
"PublicSubnetB" : {
"Value" : {"Ref" : "PublicSubnetB"},
"Description" : "Public Subnet in AZ B"
},
"PublicSubnetC" : {
"Value" : {"Ref" : "PublicSubnetC"},
"Description" : "Public Subnet in AZ C"
}
}
}
```

view rawlab1-vpc_ELB_combined.json hosted with by GitHub

The template can be downloaded here: https://s3-ap-southeast-2.amazonaws.com/andreaswasita/cloudformation_template/demo/lab1-vpc_ELB_combined.template

I put together the following video on how to use the template:

Understanding a CloudFormation template

AWS CloudFormation is pretty neat and FREE. You only need to pay for the AWS resources provisioned by the CloudFormation template.

The next bit is understanding the Structure of the template. Typically CloudFormation template will have 5 sections:

Headers

Parameters

Mappings

Resources

Outputs

Headers: Example:

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

"Description" : "Builds a VPC w/ INET Gateway and 3 public subnets. **WARNING** This template creates Amazon EC2 instance(s). You will be billed for the AWS resources used if you create a stack from this template.",
```

view rawJSON_Headers hosted with by GitHub

Parameters: Provision-time spec command-line options. Example:

```
"Parameters" : {

"KeyName" : {

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

"Type" : "String",

"MinLength": "1",

"MaxLength": "64",

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

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

},

"VpcId" : {

"Type" : "String",

"Description" : "VpcId of your existing Virtual Private Cloud (VPC)"

},

"SubnetId" : {

"Type" : "String",

"Description" : "SubnetId of an existing Public facing subnet in your Virtual Private Cloud (VPC)"
```

```
    }
    },
```

view rawParameters_JSON hosted with by GitHub

Mappings: Conditionals Case Statements. Example:

```
    "Mappings" : {
    "AWSNATAMI": {
    "us-east-1": {"AMI": "ami-6e9e4b06"},
    "us-west-2": {"AMI": "ami-8b6912bb"},
    "us-west-1": {"AMI": "ami-1d2b2958"},
    "eu-west-1": {"AMI": "ami-14913f63"},
    "ap-northeast-1": {"AMI": "ami-27d6e626"}
    }
    },
```

view rawMappings_JSON hosted with by GitHub

Resources: All resources to be provisioned. Example:

```
    "Resources" : {
    "NATIPAddress" : {
    "Type" : "AWS::EC2::EIP",
    "Properties" : {
    "Domain" : "vpc",
    "InstanceId" : { "Ref" : "NATDevice" }
    }
    },
    "NATDevice" : {
    "Type" : "AWS::EC2::Instance",
    "Properties" : {
    "InstanceType" : "m3.medium",
```

```
"KeyName" : { "Ref" : "KeyName" },
"SubnetId" : { "Ref" : "SubnetId" },
"SourceDestCheck" : "false",
"ImageId" : { "Fn::FindInMap" : [ "AWSNATAMI", { "Ref" : "AWS::Region" }, "AMI" ]},
"SecurityGroupIds" : [{ "Ref" : "NATSecurityGroup" }]
}
},
"NATSecurityGroup" : {
"Type" : "AWS::EC2::SecurityGroup",
"Properties" : {
"GroupDescription" : "Enable internal access to the NAT device",
"VpcId" : { "Ref" : "VpcId" },
"SecurityGroupIngress" : [
{ "IpProtocol" : "tcp", "FromPort" : "0", "ToPort" : "65535", "CidrIp" : "192.168.0.0/16" },
{ "IpProtocol" : "udp", "FromPort" : "0", "ToPort" : "65535", "CidrIp" : "192.168.0.0/16" } ,
{ "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1", "CidrIp" : "192.168.0.0/16" }
],
"SecurityGroupEgress" : [
{ "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : "0.0.0.0/0" },
{ "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" : "9418", "ToPort" : "9418", "CidrIp" : "0.0.0.0/0" },
{ "IpProtocol" : "tcp", "FromPort" : "0", "ToPort" : "65535", "CidrIp" : "192.168.0.0/16" },
{ "IpProtocol" : "udp", "FromPort" : "123", "ToPort" : "123", "CidrIp" : "0.0.0.0/0" },
{ "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1", "CidrIp" : "0.0.0.0/0" }
]
```

```
    }
    },
    "PrivateRouteTable" : {
    "Type" : "AWS::EC2::RouteTable",
    "Properties" : {
    "VpcId" : {"Ref" : "VpcId"},
    "Tags" : [
    {"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
    {"Key" : "Network", "Value" : "Private Route" }
    ]
    }
    },
    "PrivateRoute" : {
    "Type" : "AWS::EC2::Route",
    "Properties" : {
    "RouteTableId" : { "Ref" : "PrivateRouteTable" },
    "DestinationCidrBlock" : "0.0.0.0/0",
    "InstanceId" : { "Ref" : "NATDevice" }
    }
    }
    },
```

view rawResources_JSON hosted with by GitHub

Outputs: Example:

```
    "Outputs" : {
    "PrivateRouteTableId" : {
    "Value" : {"Ref" : "PrivateRouteTable"},
    "Description" : "Private Route Table ID"
```

```
    }
  }
}
```

view rawOutputs_JSON hosted with by GitHub

Note: Not all AWS Resources can be provisioned using AWS CloudFormation and it has the following limitations.

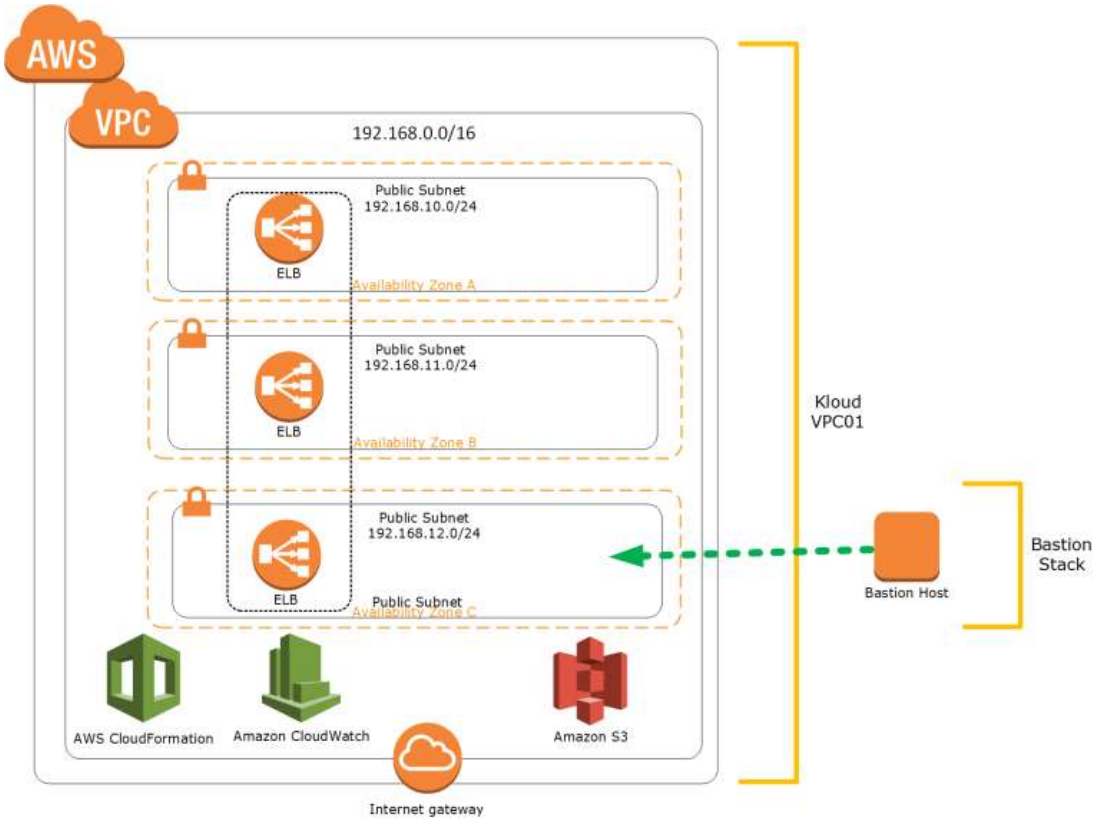
Stacking the AWS CloudFormation

This will help the reader on how to layer the stack on top of the existing AWS CloudFormation stack using AWS CloudFormation instead of modifying the base template. AWS resources can be added into existing VPC using the outputs detailing the resources from the main VPC stack instead of having to modify the main template.

This will allow us to compartmentalize, separate out any components of AWS infrastructure and again **versioning** all different AWS infrastructure code for every components.

Note: The template I will use for this post for educational purposes only and may not be suitable for production workloads :).

Diagram below will help to illustrate the concept:



Bastion Stack

Previously (Part 1), we created the initial stack which provide us the base VPC. Next, we will provision bastion stack which will create a bastion host on top of our base VPC. Below are the components of the bastion stack:

- Create IAM user that can find out information about the stack and has permissions to create KeyPairs and actions related.
- Create bastion host instance with the AWS Security Group enable SSH access via port 22
- Use CloudFormation Init to install packages, create files and run commands on the bastion host instance also take the creds created for the IAM user and setup to be used by the scripts
- Use the EC2 UserData to run the cfn-init command that actually does the above via a bash script
- The condition handle: the completion of the instance is dependent on the scripts running properly, if the scripts fail, the CloudFormation stack will error out and fail

Below is the CloudFormation template to build the bastion stack:

```
{
"AWSTemplateFormatVersion" : "2010-09-09",
"Description" : "Add a bastion host to an existing VPC. VPC must have an internet gateway already. **WARNING** This template creates an Amazon EC2 instance. You will be billed for the AWS resources used if you create a stack from this template.",
"Parameters" : {
"KeyName" : {
"Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instances",
"Type" : "String",
"MinLength": "1",
"MaxLength": "64",
"AllowedPattern" : "[-_ a-zA-Z0-9]*",
"ConstraintDescription" : "can contain only alphanumeric characters, spaces, dashes and underscores."
},
"BastionKeyName" : {
"Description" : "Name of the EC2 KeyPair we will create internally to access instances in our VPC",
"Type" : "String",
"MinLength": "1",
"MaxLength": "64",
"AllowedPattern" : "[-_ a-zA-Z0-9]*",
```

```
"ConstraintDescription" : "can contain only alphanumeric characters, spaces, dashes and underscores."
},
"VpcId" : {
"Type" : "String",
"Description" : "VpcId of your existing Virtual Private Cloud (VPC)"
},
"SubnetId" : {
"Type" : "String",
"Description" : "SubnetId of an existing Public facing subnet in your Virtual Private Cloud (VPC)"
}
},
"Mappings" : {
"AWSInstanceType2Virt": {
"t2.micro": {"Virt": "HVM"},
"t2.small": {"Virt": "HVM"},
"t2.medium": {"Virt": "HVM"},
"m3.medium": {"Virt": "HVM"},
"m3.large": {"Virt": "HVM"},
"m3.xlarge": {"Virt": "HVM"},
"m3.2xlarge": {"Virt": "HVM"}
},
"AWSRegionVirt2AMI": {
"us-east-1": {
"PVM": "ami-50842d38",
"HVM": "ami-08842d60"
},
"us-west-2": {
"PVM": "ami-af86c69f",
"HVM": "ami-8786c6b7"
},
"us-west-1": {
"PVM": "ami-c7a8a182",
"HVM": "ami-cfa8a18a"
},
"eu-west-1": {
"PVM": "ami-9c7ad8eb",
```

```
"HVM": "ami-6e7bd919"
},
"ap-northeast-1": {
"HVM": "ami-35072834"
}
}
},
"Resources" : {
"CfnUser": {
"Type": "AWS::IAM::User",
"Properties": {
"Path": "/",
"Policies": [
{
"PolicyName": "root",
"PolicyDocument": {
"Statement": [
{
"Effect" : "Allow",
"Action": [
"ec2:CreateKeyPair",
"ec2:DescribeKeyPairs",
"ec2:DescribeRegions",
"ec2:ImportKeyPair"
],
"Resource" : "*"
},
{
"Effect": "Allow",
"Action": "cloudformation:DescribeStackResource",
"Resource": "*"
}]
}
}
]
}
},
"CfnKeys": {
```

```
"Type": "AWS::IAM::AccessKey",
"Properties": {
"UserName": {
"Ref": "CfnUser"
}
}
},
"IPAddress" : {
"Type" : "AWS::EC2::EIP",
"Properties" : {
"Domain" : "vpc",
"InstanceId" : { "Ref" : "BastionHost" }
}
},
"BastionSecurityGroup" : {
"Type" : "AWS::EC2::SecurityGroup",
"Properties" : {
"VpcId" : { "Ref" : "VpcId" },
"GroupDescription" : "Enable SSH access via port 22",
"SecurityGroupIngress" : [ { "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" :
"22", "CidrIp" : "0.0.0.0/0" } ]
}
},
"BastionHost" : {
"Type" : "AWS::EC2::Instance",
"Metadata": {
"AWS::CloudFormation::Init": {
"config": {
"packages": {
"yum": {
"python-boto": []
}
},
"files": {
"/home/ec2-user/create-keypair" : {
"content" : {
"Fn::Join" : ["", ["#!/usr/bin/python\n",
"import string\n",
```

```
"import random\n",
"import boto.ec2\n",
"kp_name = '",{ "Ref" : "BastionKeyName" },"'\n",
"ec2 = boto.ec2.connect_to_region('", {"Ref" : "AWS::Region" }, "')\n",
"keypair = ec2.create_key_pair(kp_name)\n",
"keypair.save('/home/ec2-user/.ssh/')\n",
"print 'Created keypair: %s' % kp_name\n"]]
},
"mode" : "000750",
"owner" : "ec2-user",
"group" : "ec2-user"
},
"/home/ec2-user/.boto": {
"content": {
"Fn::Join": ["", [ "[Credentials]\n",
"aws_access_key_id = ", { "Ref": "CfnKeys" }, "\n",
"aws_secret_access_key = ", { "Fn::GetAtt": ["CfnKeys", "SecretAccessKey"] }, "\n",
"[Boto]\n",
"ec2_region_name = ", { "Ref" : "AWS::Region" }, "\n",
"ec2_region_endpoint = ec2.", { "Ref" : "AWS::Region" }, ".amazonaws.com\n"]]
},
"mode": "000600",
"owner": "ec2-user",
"group": "ec2-user"
}
},
"commands" : {
"00create-keypair" : {
"command" : ["su", "ec2-user", "-c", "python create-keypair"],
"cwd" : "/home/ec2-user"
}
}
}
}
},
"Properties" : {
"InstanceType" : "t2.medium",
```

```
"ImageId" : { "Fn::FindInMap" : [ "AWSRegionVirt2AMI", { "Ref" : "AWS::Region" },
{"Fn::FindInMap": ["AWSInstanceType2Virt", "t2.medium", "Virt"]} ]},
"SecurityGroupIds" : [{ "Ref" : "BastionSecurityGroup" }],
"SubnetId" : { "Ref" : "SubnetId" },
"KeyName" : { "Ref" : "KeyName" },
"UserData": {
"Fn::Base64" : { "Fn::Join" : ["", [
"#!/bin/bash -v\n",
"yum update -y aws-cfn-bootstrap\n",
"# Helper function\n",
"function error_exit\n",
"{\n",
" /opt/aws/bin/cfn-signal -e 1 -r \"$1\" '", { "Ref" : "BastionHostHandle" }, "'\n",
" exit 1\n",
"}\n",
"/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" }, " -r BastionHost ",
" --access-key ", { "Ref" : "CfnKeys" },
" --secret-key ", {"Fn::GetAtt": ["CfnKeys", "SecretAccessKey"]},
" --region ", { "Ref" : "AWS::Region" }, " || error_exit 'Failed to run cfn-
init'\n",
"# All is well so signal success\n",
"/opt/aws/bin/cfn-signal -e 0 -r \"Server setup complete\" '", { "Ref" :
"BastionHostHandle" }, "'\n"
]
]
}
}
}
},
"BastionHostHandle" : {
"Type" : "AWS::CloudFormation::WaitConditionHandle"
},
"ControllerCondition" : {
"Type" : "AWS::CloudFormation::WaitCondition",
"DependsOn" : "BastionHost",
"Properties" : {
"Handle" : { "Ref" : "BastionHostHandle" },
"Timeout" : "900"
```

```
}
}
},
"Outputs" : {
"InstanceID" : {
"Value" : {"Ref": "BastionHost"},
"Description" : "Bastion Instance ID"
},
"IPAddress" : {
"Value" : { "Ref" : "IPAddress" },
"Description" : "Public IP address of instance"
},
"BastionKeyName" : {
"Value" : { "Ref" : "BastionKeyName" },
"Description" : "The internal bastion KeyPair name"
}
}
}
```

view rawlab1-bastion_stack.json hosted with by **GitHub**

Following are the high level steps to layer the bastion stack on top of the initial stack:

- Get the subnet details which we created previously on initial stack
- Download the bastion stack template: https://s3-ap-southeast-2.amazonaws.com/andreaswasita/cloudformation_template/demo/lab1-bastion_stack.template
- Create new stack on AWS CloudFormation and specify the initial stack VPC ID, SubnetId, KeyName and BastionKey Name in the related parameters
- Tags the instance
- Next we need to connect and validate the bastion host instance also "cat" out the bastion keypair. You can use PuTTY or Git Bash for Windows users or OSX Terminal for OSX users.

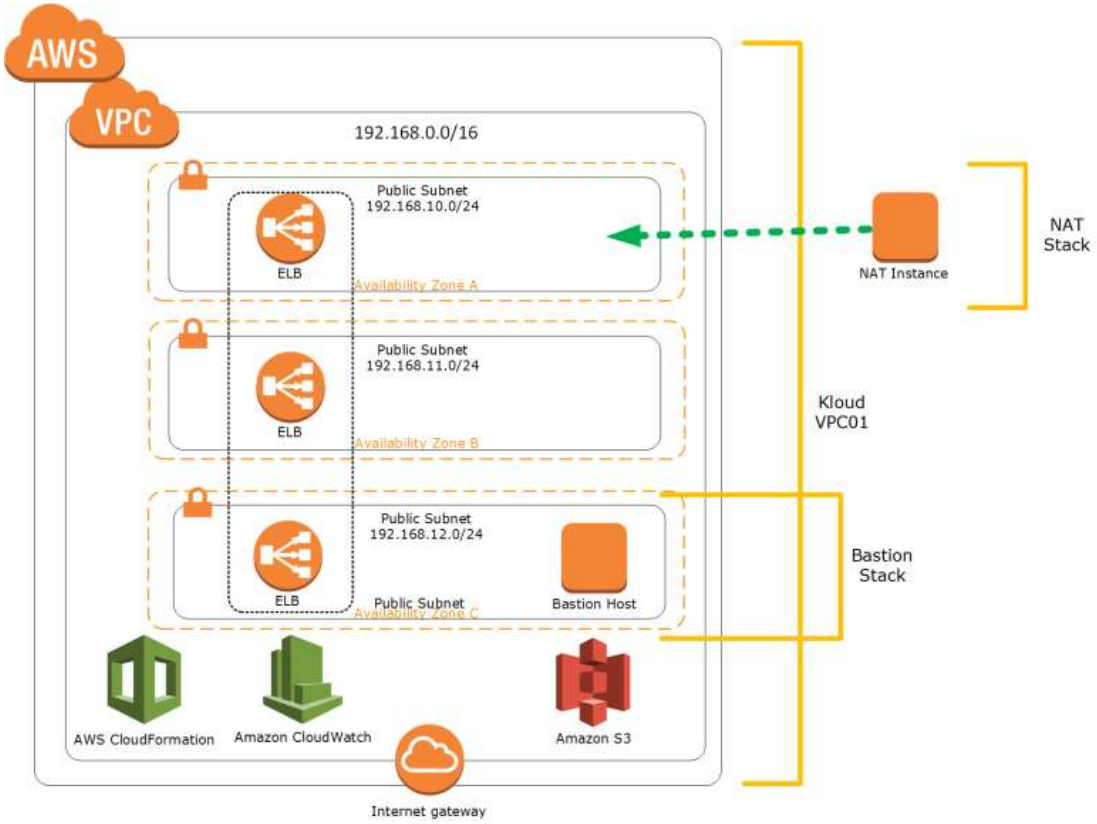
I put together the following video on how to use the template:

NAT Stack

It is important to design the VPC with security in mind. I recommend to design your Security Zone and network segregation, I have written a blog post regarding how to Secure Azure Network. This approach also can be impelemented on AWS environment using VPC, subnet and security groups. At the very minimum we will segregate the Private subnet and Public subnet on our VPC.

NAT instance will be added to our Initial VPC “public” subnets so that the future private instances can use the NAT instance for communication outside the Initial VPC. We will use exact same method like what we did on Bastion stack.

Diagram below will help to illustrate the concept:



The components of the NAT stack:

- An Elastic IP address (EIP) for the NAT instance
- A Security Group for the NAT instance: Allowing ingress traffic tcp, udp from port 0-65535 from internal subnet ; Allowing egress traffic tcp port 22, 80, 443, 9418 to any and egress traffic udp port 123 to Internet and egress traffic port 0-65535 to internal subnet
- The NAT instance
- A private route table
- A private route using the NAT instance as the default route for all traffic

Following is the CloudFormation template to build the stack:

```
{
"AWSTemplateFormatVersion" : "2010-09-09",
"Description" : "Builds a NAT host. **WARNING** This template creates Amazon EC2 instance(s). You will be billed for the AWS resources used if you create a stack from this template.",
"Parameters" : {
```

```
"KeyName" : {
"Description" : "Name of an existing EC2 KeyPair to enable SSH access to the
instances",
"Type" : "String",
"MinLength": "1",
"MaxLength": "64",
"AllowedPattern" : "[-_ a-zA-Z0-9]*",
"ConstraintDescription" : "can contain only alphanumeric characters, spaces, dashes
and underscores."
},
"VpcId" : {
"Type" : "String",
"Description" : "VpcId of your existing Virtual Private Cloud (VPC)"
},
"SubnetId" : {
"Type" : "String",
"Description" : "SubnetId of an existing Public facing subnet in your Virtual
Private Cloud (VPC)"
}
},
"Mappings" : {
"AWSNATAMI": {
"us-east-1": {"AMI": "ami-6e9e4b06"},
"us-west-2": {"AMI": "ami-8b6912bb"},
"us-west-1": {"AMI": "ami-1d2b2958"},
"eu-west-1": {"AMI": "ami-14913f63"},
"ap-northeast-1": {"AMI": "ami-27d6e626"}
}
},
"Resources" : {
"NATIPAddress" : {
"Type" : "AWS::EC2::EIP",
"Properties" : {
"Domain" : "vpc",
"InstanceId" : { "Ref" : "NATDevice" }
}
},
"NATDevice" : {
```

```
"Type" : "AWS::EC2::Instance",
"Properties" : {
"InstanceType" : "m3.medium",
"KeyName" : { "Ref" : "KeyName" },
"SubnetId" : { "Ref" : "SubnetId" },
"SourceDestCheck" : "false",
"ImageId" : { "Fn::FindInMap" : [ "AWSNATAMI", { "Ref" : "AWS::Region" }, "AMI" ]},
"SecurityGroupIds" : [{ "Ref" : "NATSecurityGroup" }]
}
},
"NATSecurityGroup" : {
"Type" : "AWS::EC2::SecurityGroup",
"Properties" : {
"GroupDescription" : "Enable internal access to the NAT device",
"VpcId" : { "Ref" : "VpcId" },
"SecurityGroupIngress" : [
{ "IpProtocol" : "tcp", "FromPort" : "0", "ToPort" : "65535", "CidrIp" :
"192.168.0.0/16" },
{ "IpProtocol" : "udp", "FromPort" : "0", "ToPort" : "65535", "CidrIp" :
"192.168.0.0/16" } ,
{ "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1", "CidrIp" :
"192.168.0.0/16" }
],
"SecurityGroupEgress" : [
{ "IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" : "0.0.0.0/0"
},
{ "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" : "9418", "ToPort" : "9418", "CidrIp" :
"0.0.0.0/0" },
{ "IpProtocol" : "tcp", "FromPort" : "0", "ToPort" : "65535", "CidrIp" :
"192.168.0.0/16" },
{ "IpProtocol" : "udp", "FromPort" : "123", "ToPort" : "123", "CidrIp" : "0.0.0.0/0"
},
{ "IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1", "CidrIp" : "0.0.0.0/0"
}
]
}
```

```
},
"PrivateRouteTable" : {
"Type" : "AWS::EC2::RouteTable",
"Properties" : {
"VpcId" : {"Ref" : "VpcId"},
"Tags" : [
{"Key" : "Application", "Value" : { "Ref" : "AWS::StackName"} },
{"Key" : "Network", "Value" : "Private Route" }
]
}
},
"PrivateRoute" : {
"Type" : "AWS::EC2::Route",
"Properties" : {
"RouteTableId" : { "Ref" : "PrivateRouteTable" },
"DestinationCidrBlock" : "0.0.0.0/0",
"InstanceId" : { "Ref" : "NATDevice" }
}
}
},
"Outputs" : {
"PrivateRouteTableId" : {
"Value" : {"Ref" : "PrivateRouteTable"},
"Description" : "Private Route Table ID"
}
}
}
```

view rawlab1-nat_stack.json hosted with by **GitHub**

Similar like the previous steps on how to layer the stack:

- Get the details on public subnet to put the NAT instance in
- Download the NAT stack template: https://s3-ap-southeast-2.amazonaws.com/andreaswasita/cloudformation_template/demo/lab1-nat_stack.template
- Create new stack on AWS CloudFormation and specify the initial stack VPC ID, SubnetId, KeyName and BastionKey Name in the related parameters
- Tags as usual