

Orchestrating BIG-IP in the Public Cloud

Demonstrating various deployment models in AWS EC2 Using Ansible

Last updated: November 23th, 2015

Alex Applebaum, PME, F5 Networks, a.applebaum@f5.com

Chris Mutzel, PME, F5 Networks, c.mutzel@f5.com

[Overview, Intended Audience, and Goals](#)

[Background Information and Technologies](#)

[Project Code on Github and Code Layout](#)

[Technologies](#)

[Amazon Web Services](#)

[Ansible](#)

[Docker](#)

[Hackazon](#)

[JMeter](#)

[BIG-IP](#)

[iApps](#)

[Splunk](#)

[Operating Systems, Python and associated Packages](#)

[Connectivity Requirements](#)

[Vagrant and Virtualbox](#)

[Putting it all together](#)

[Installing the F5aws tool](#)

[AWS Pre-requisites](#)

[Acquiring your AWS Access Keys](#)

[Creating an SSH Key Pair](#)

[Download the code](#)

[Clone the code](#)

[OR Download as compressed zip](#)

[Setting up the automation host environment](#)

[Quick start using CloudFormation templates](#)

[OR Setup using Vagrant and VirtualBox](#)

[Configuring settings on your automation host](#)

[Installing your SSH Private Key](#)

[Adding your AWS Credentials for use boto and the AWS CLI](#)

[Specifying AWS credentials and private key location for the f5aws code](#)

[Accepting EULA Agreements](#)

[Requesting Additional IP Address From Amazon \(optional\)](#)

[Tool Usage](#)

[Command: init](#)

[Deployment model, region, and availability zones](#)

[Deployment Type](#)

[Deploy Analytics](#)

[Command: deploy](#)

[Command: list](#)

[Command: info](#)

[Command: start_traffic](#)

[Command: stop_traffic](#)

[Command: teardown](#)

[Command: remove](#)

[Related notes on usage](#)

[Example End-to-End Usage](#)

[Architecture](#)

[Deployment models](#)

[single-standalone](#)

[single-cluster](#)

[standalone-per-zone](#)

[cluster-per-zone](#)

[Deployment Type](#)

[Deploying Splunk](#)

[AWS Configuration Overview](#)

[BIGIP-Configuration Overview](#)

[Use of Ansible with iControlREST](#)

[Configuring BIG-IP Running LTM](#)

[Configuring BIG-IP Running GTM](#)

[Virtual Servers](#)

[Virtual 1 \(bigip_app1 role\)](#)

[Virtual 2 \(bigip_app2 role\)](#)

[Handling Common Errors](#)

[Description: Various task timeouts:](#)

[Description: Incorrect system time on vagrant VM causes fatal error when using the AWS CLI or boto libraries to interact with AWS EC2 endpoints.](#)

[Description: CloudFormation Deployment Errors](#)

[Improving this Project](#)

[Areas for Improvement](#)

[Reduce EIPs](#)

[Reduce CFTs](#)

[Persistence Models](#)

[Clustering & Licensing](#)

[Contributing](#)

[Additional Resources](#)

[F5 Documentation, Articles, and Videos](#)

[Inspiration](#)

[Technical](#)

[Acknowledgements](#)

Overview, Intended Audience, and Goals

This document summarizes the aws-deployments project available under the F5 Networks account on GitHub, <https://github.com/f5networks/aws-deployments>

We provide background information, setup and usage instructions, a description of the implemented architecture, and a vision for future improvements. Additional resources, content for further reading and inspiration.

The relevant audience for this work includes those in networking, cloud, or system architecture, engineering, and administration roles. It is particularly applicable to those who wish to learn more about how to deploy application services with BIG-IP in public cloud environments, and how to manage the lifecycle of BIG-IP in an automated manner.

There are two primary goals for this project:

1. Provide the opportunity to easily test deployment models and use cases of BIG-IP in AWS EC2. While AWS is used to provide a virtual compute and networking infrastructure, best practices shown here may be applicable to other public and private 'cloud' environments.
2. To show how the lifecycle of BIG-IP services can be automated using open-source configuration management and orchestration tools, in conjunction with the APIs provided by the BIG-IP platform.

Finally, while this document and the associated sample code attempts to provide best practices for the deployment and integration of F5 services in AWS and other public cloud environments, we strongly urge the reader to visit the official documentation for F5 products on support.f5.com.

Background Information and Technologies

In this section, we discuss technologies used within this project. Some of these technologies are used as development and test tools; they are not critical to the function of the aws-deployments tool, but are used in order to make it easier to use. We also define dependency and version requirements by the code where relevant. For readers who are specifically interested in using the aws-deployments tool rather than engaging in an architectural conversation, focus should be placed on steps for setup of the test environment. These are the only steps which need to be explicitly followed for tool usage.

At a high-level, this project started with the architecture shown in Figure 1. We wish to deploy applications, protected and made highly available with F5 services, on top of a programmable infrastructure. In Figure 1, we have called out the use of templated infrastructure deployment technologies, like those provided by cloud providers, as a useful way to deploy virtual resources in many cloud environments.

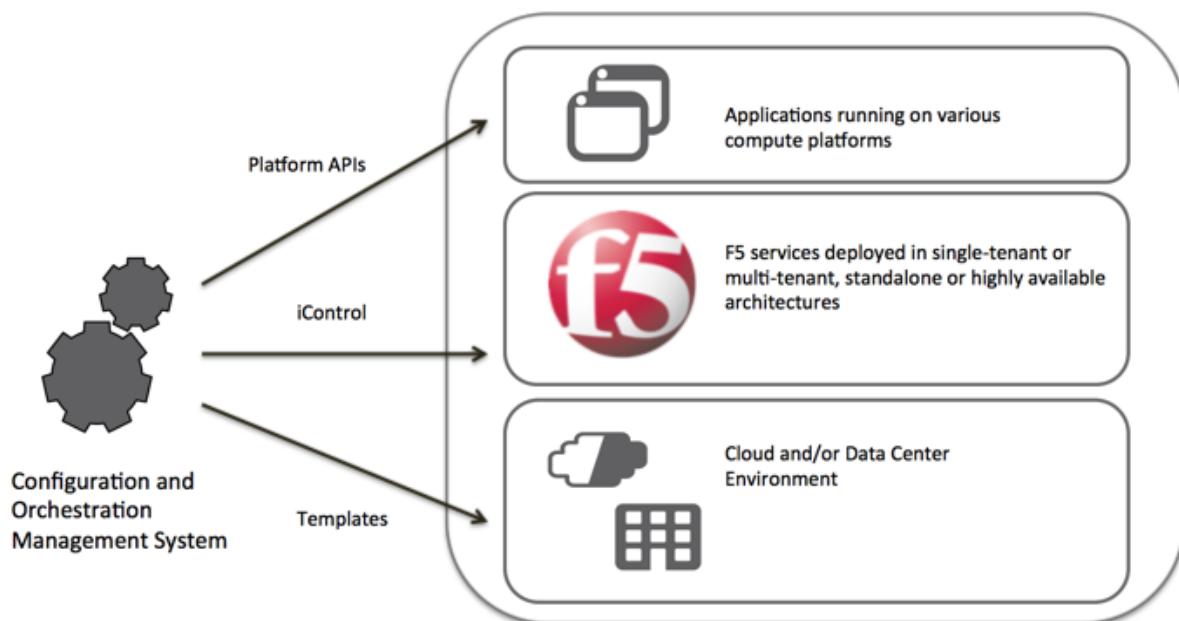


Figure 1 - High-level architecture of this project, representing the generic deployment of applications and network services on top of virtualized or programmable infrastructure.

Using this architecture, we wish to show how the lifecycle of BIG-IP, running as a virtual edition, can be fully automated. To implement a prototype showing use of the BIG-IP APIs to manage the service deployment and lifecycle, we have chosen technologies shown in Figure 2.

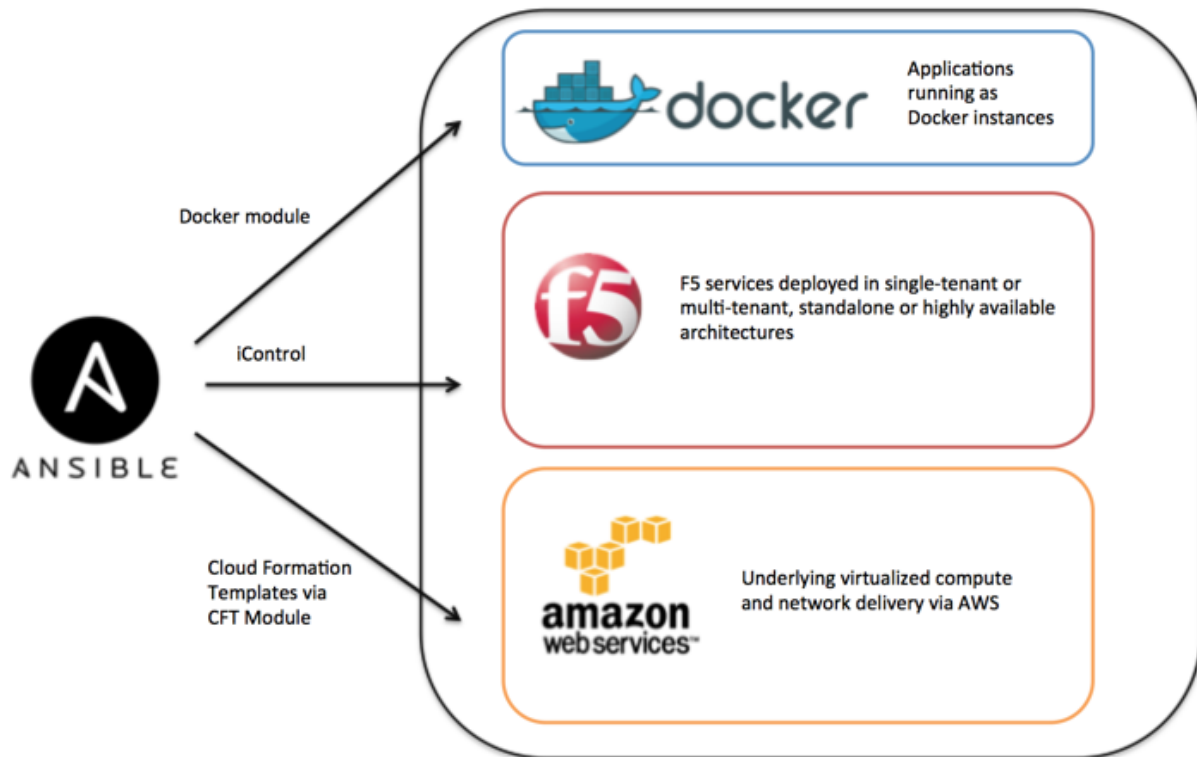


Figure 2 - Technologies chosen to implement a proof of concept, showing how the BIG-IP VE footprint and service can be fully automated.

In this next 'Technologies' section, we discuss how these and other various technologies have been used. The end product of this work is a command line program which provides an API for deploying F5 in numerous footprints in AWS for demonstration purposes. We refer to this script as the 'f5aws' tool. It is located in the `./bin` directory of the code associated with this project.

Project Code on Github and Code Layout

As mentioned, this document refers to code on Github. Throughout the rest of this document, we will assume that you have either downloaded the code for review, or are browsing the project repository online. We will make many references to the following items:

- `./bin/f5aws`: This script provides a command line interface for deploying F5. We refer to it as the `f5aws aws-deployments` tool
- `./src`: Includes some custom python modules to implement the `f5aws` CLI.
- `./roles`: This directory includes roles we have written to manage the various hosts in the deployment models describe later. These roles are used by our Ansible playbooks
- `./playbooks`: This directory includes Ansible playbooks (e.g. workflows) which leverage the roles above.
- `./library`: This directory includes libraries we have included to supplement the number of built-in modules that ship with Ansible.

Technologies

Amazon Web Services

On the F5 community site, DevCentral, we have commented on the best practices driving architecture and deployment strategy of BIG-IP in AWS. Please consult these articles for background information on the basics of EC2/VPC networking, running BIG-IP as a compute node in AWS, and highly-availability deployment topologies for your applications. These topics can be found in the following posts:

[F5 in AWS Part 1 - AWS Networking Basics](#)

[F5 in AWS Part 2 - Running BIG-IP in an EC2 Virtual Private Cloud](#)

[F5 in AWS Part 3 - Advanced topologies and more on highly-available services](#)

At a high-level, it is important to understand that virtual networking in EC2 prevents access to layer 2 protocols like GARP and 802.1Q tagging. Additionally, Amazon provides basic network services like load balancing (ELB) and DNS (Route53). These services are limited to basic functionality (for example, no UDP). Granular control is provided over the networking infrastructure in the form of DHCP option sets, route tables, NAT instances, DNS, and internet and VPN gateways. Again, please see the above content to better understand best practices for using BIG-IP along with these constructs.

Amazon provides several ways of orchestrating resources in EC2 and other Amazon Web Services. We do so in this project using the AWS CLI and the Python library, Boto. To authenticate requests when using these APIs, the `f5aws` tool requires knowledge of the AWS Access Key and AWS Secret key associated with an account which has rights to provision resources within EC2. We expect that the reader already has these, or that these keys may be procured through the relevant request process at your company. We do not share, distribute, or record these keys in anyway, and they will remain private to your environment.

Associated with your AWS account are limits on EC2 resources including the number of VPCs, CloudFormation stacks, and EIPs which can be used concurrently. In the deployment models

section of this document, we list the requirements on these resources for each topology. For certain topologies, you may be required to request an increase in account limits for certain resource types.

Ansible

For this project, we have chosen to use Ansible to fill the need of configuration management and orchestration of BIG-IP, AWS services, and our demo application. Ansible is an open-source tool provided by Ansible, Inc. It is agentless, SSH-based and written in Python.

It is important that we identify a few pieces of vocabulary that are specific to Ansible. An Ansible *playbook* is a defined workflow that may deploy, orchestrate, configure, or otherwise manage an IT system. These playbooks are made up of *tasks*, which are granular steps to be executed in a procedural order. Ansible provides the concept of an inventory, which is used to define sets of *hosts*, or systems under management. These hosts may be grouped by host type (i.e. “database servers”, or “bigips”) against which playbooks are executed. Like Puppet, Chef, and other IT-workflow or configuration management tools, Ansible provides a diverse set of standard libraries to interact with common infrastructure elements known as *modules*. Examples of modules included with the out-of-the-box distribution are ‘npm’ for managing Node.js packages, ‘cloudformation’ for deploying CloudFormation stacks in AWS, and ‘template’ for dynamically creating files on a file-system from pre-defined Jinja2 templates. In this project, we have leveraged the above concepts to build workflows which can be used to manage the lifecycle of BIG-IP in AWS.

Docker

We use Docker in this project for the same reasons that many others have quickly adopted it. Docker provides an application execution platform and associated development tools which make it lightweight and portable. In this project, a simple HTTP web application is launched within one or more Docker containers. We use a custom built Ubuntu AMI to run Docker for hosting our demo application.

Hackazon

To provide a demonstration of the capabilities of BIG-IP Application Security Manager (ASM), we chose to use Hackazon as the demonstration web app in our deployment. Hackazon is a vulnerable web app, built using modern web technologies. You can find more information about Hackazon here:

<https://github.com/rapid7/hackazon>

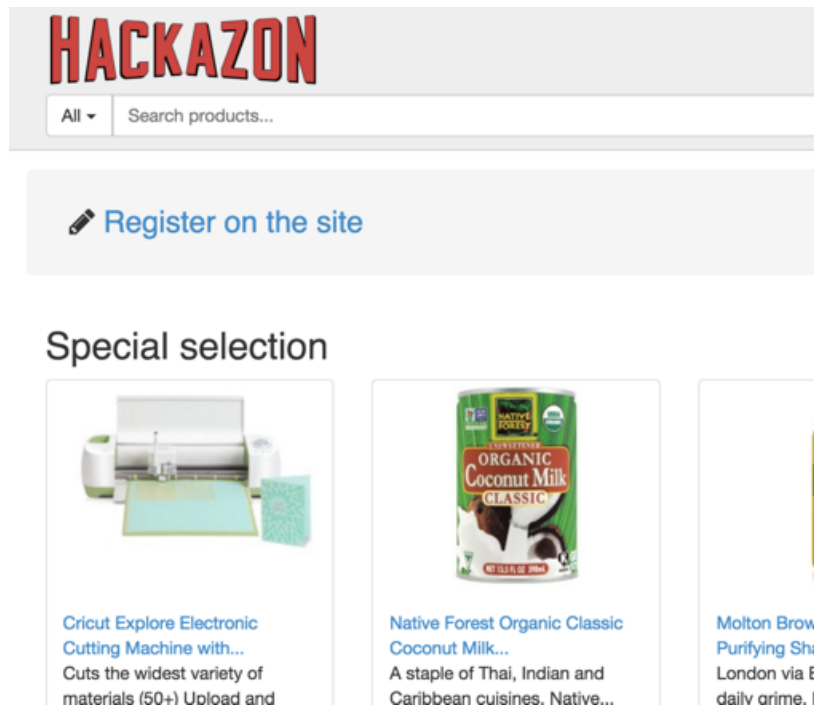


Figure 3 - Showing the login page for the Hackazon application deployed

JMeter

In some deployment models with the aws-deployments tool, traffic may be generated to demonstrate reporting and analytics features. Traffic is generated using Apache JMeter, an open-source, functional load-testing application written in Java. JMeter loads are defined via an XML-styled text document.

BIG-IP

All development and testing of this project has been performed using BIG-IP 11.6. In 11.6, iControlREST (iCR) has been released as a GA feature of the BIG-IP platform. Also in this version, iCR does not include coverage for device licensing or clustering. We fall back to the use of iControlSOAP (which is leveraged within the BigSuds or pycontrol python libraries) for missing features as necessary. The version of BIG-IP launched when using the aws-deployments tool is defined in ./roles/inventory_manager/defaults and does not need to be altered.

Within the deployment models described later in this document, the f5aws API allows you deploy a solution providing only load balancing, or one that includes a Web application firewall.

Traffic management and web security features are provided using features from the Local Traffic Manager (LTM) and Application Security Manager (ASM) modules, respectively.

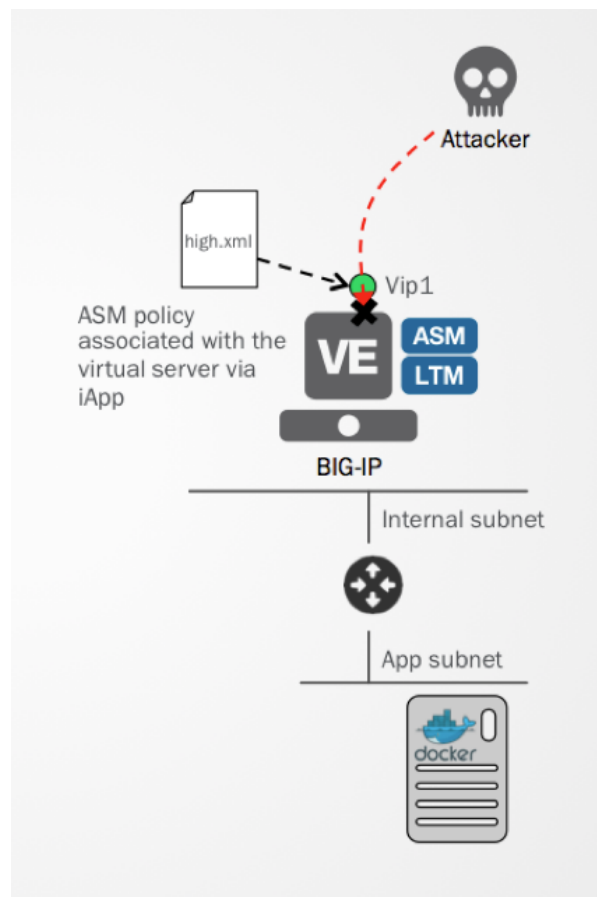


Figure 4 - Topology if ASM is deployed in addition to LTM. When installed, ASM can be used to deploy a web application file wall to block L4-L7 vulnerabilities.

In deployment models where Global Traffic Manager (GTM) is deployed for DNS resolution, a separate BIG-IP is launched to manage this function. In this situation, GTM monitors the virtual servers deployed through the big3d protocol, in order to determine the health of the application. If the virtual server is down, GTM can be configured to resolve DNS accordingly. This is shown in Figure 5.

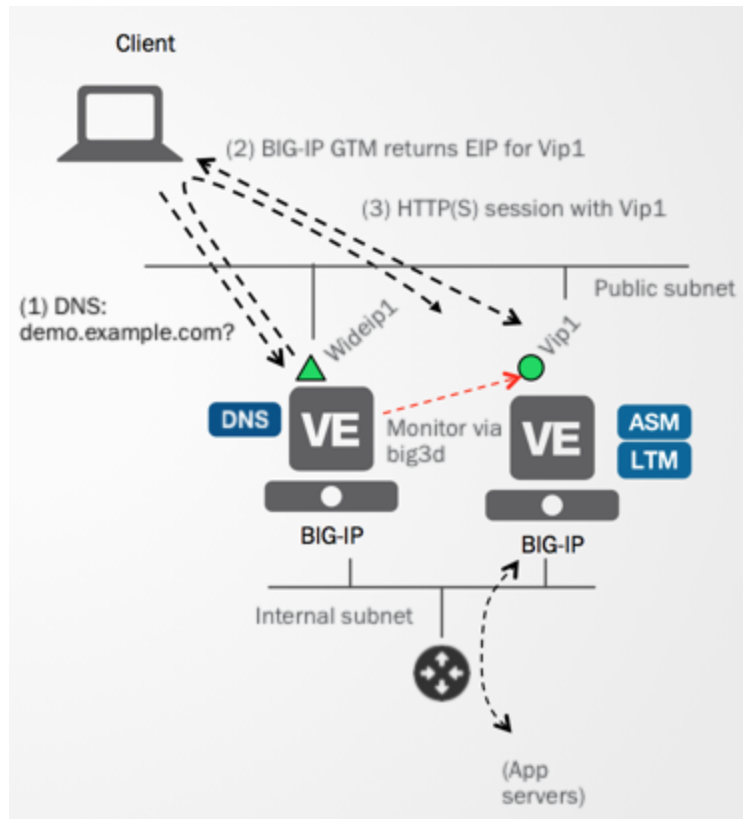


Figure 5 - Showing how GTM (referred to here in this diagram as DNS, its new name), monitors the virtual server deployed and handles DNS resolution intelligently.

iApps

We use iApps from F5 to easily instantiate and manage application services for our deployment. iApps are F5's powerful re-entrant templates for creating virtual services. They allow you to see and manage all the elements for the virtual service while providing custom menus using language that all users in your organization can understand.

We use a version of the F5 HTTP iApp available on DevCentral, linked below. This iApp allows you to easily associate ASM policy to a virtual server. In general, iApps provide a declarative, question and answer interface to application owners who administrators who manage and application deployment.

<https://devcentral.f5.com/codeshare/http-iapp-v110>

Splunk

To demonstrate the ability to integrate BIG-IP within various analytics platforms, we provide to ability to deploy Splunk. Using the High-Speed Logging feature on BIG-IP, log events are pushed to the Splunk host, which is running recently developed F5 plugins. The Splunk instance

deployed can thus be used to visualize metrics for LTM metrics, and ASM metrics if ASM has been deployed.

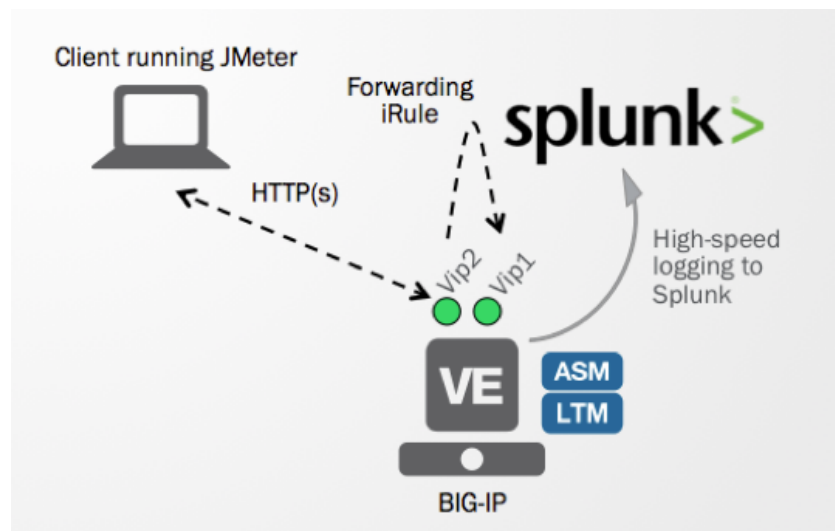


Figure 6 - Splunk can be deployed using the f5aws tool to examine the recent F5 plugins developed for Splunk Enterprise

Operating Systems, Python and associated Packages

When using Ansible for configuration management, a central host is required from which Ansible will run. Steps later in the document detail the setup of this host, for which have provided a base image. In this base image, Python 2.7.9 has been installed for use within a Python virtualenv. Python 2.7.9 includes a completely backported ssl module from Python 3.4.

In addition to installing a specific Python version, a number of Python modules are also installed within the Python virtualenv. For the complete list of installed modules, see the top-level file, requirements.txt. A few of the more important modules are:

- boto: Python API for interacting with AWS over HTTPS

- bigsuds: Module which leverages iControlSoap for TMOS device provisioning

The above operating system and python requirements are handled for you when using the setup processes using Vagrant or the quick start CFTs (CloudFormation templates), and are provided for information purposes only.

Connectivity Requirements

Use of the f5aws tool within the aws-deployments project requires internet connectivity. Specifically, HTTP(S) connections are made to DockerHub, pypi.org, AWS API endpoints, the AWS public IP address range, and apt-get public repositories.

Vagrant and Virtualbox

Vagrant is a software development tool which provides programmatic setup, configuration, and teardown of virtualized environments. In an effort to simplify use of the f5aws tool, we have configured a Vagrant environment that includes the above operating system and python dependencies. This should allow the reader to download the project code from Github, execute the “vagrant up” command, and get started without dealing with complex dependency installation or resolution issues. VirtualBox is used as the virtual machine provider though it might be possible to substitute other hypervisor technologies like VMWare Fusion. This has not been tested. All development has been completed and testing using Vagrant 1.7.2 and VirtualBox 4.3.30

We configure the vagrant machine to use a bridged networking mode by using the `config.vm.network = “public_network”` option. We have also provided a base image “f5networks/demo” which includes Python 2.7.9. These options can be seen in `./vagrant/VagrantFile`. By pre-baking this image, we have reduced the launch time of the vagrant environment.

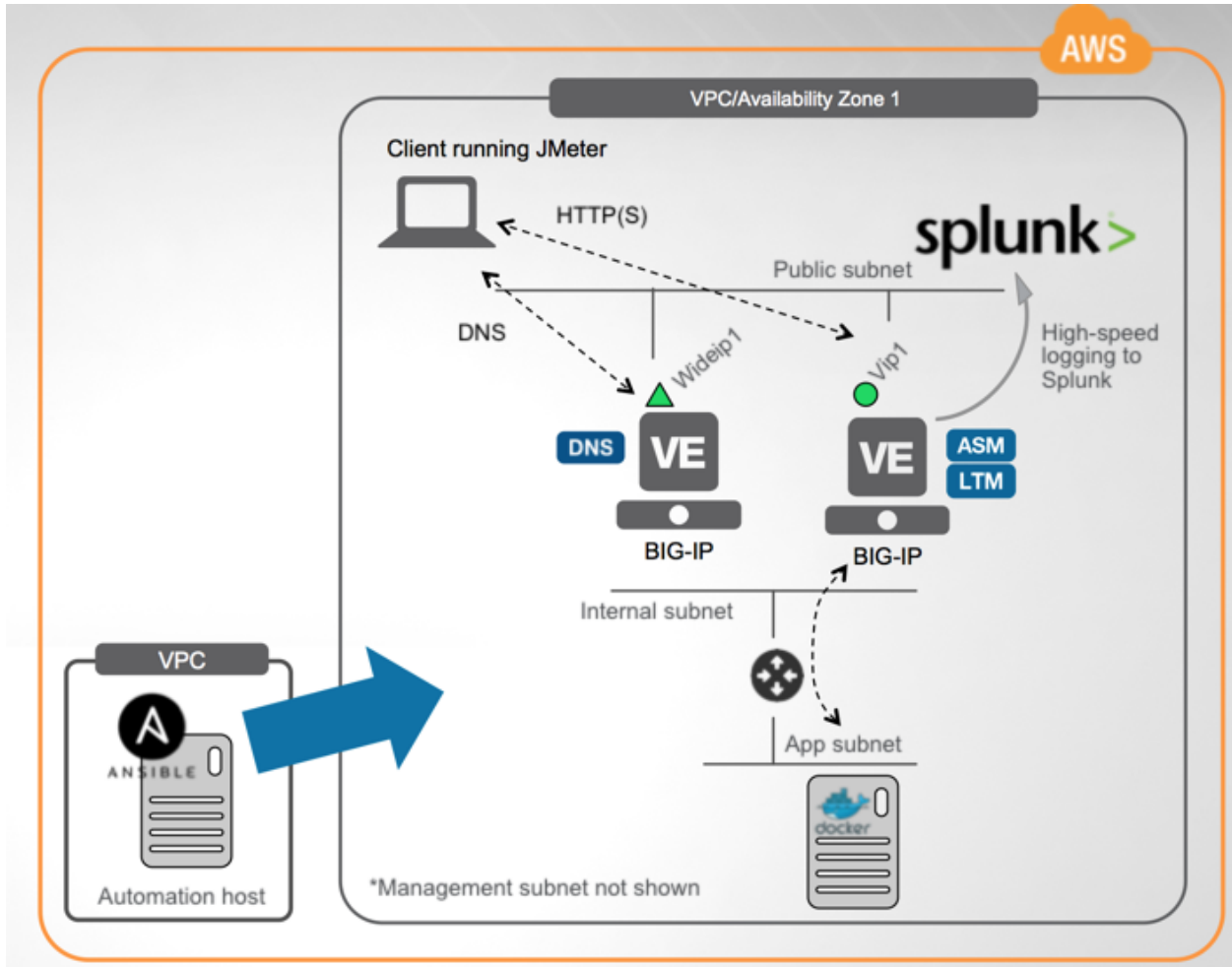


Figure 7 - Showing the standalone-per-zone topology described later in this document that can be deployed with the f5aws tool.

Putting it all together

Figure 7 shows all the technologies leveraged for a deployment using the f5aws tool. Notice that Ansible runs on a separate host, outside of the VPC for the application deployment. This host could also be realized using the steps provided for running Vagrant + VirtualBox.

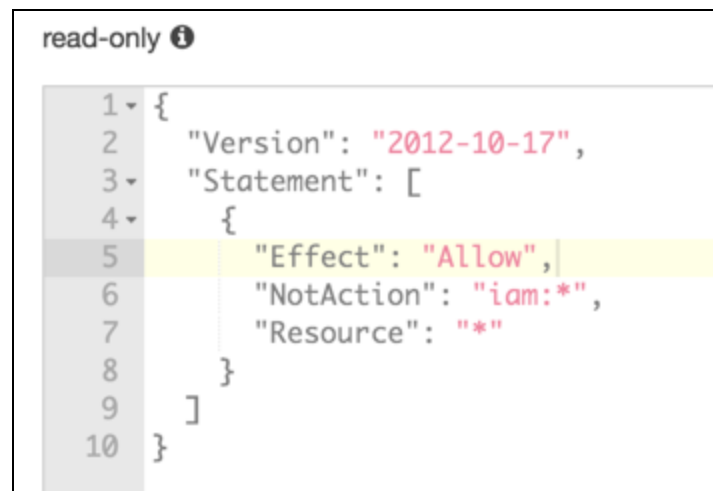
Installing the F5aws tool

In this section we document the necessary steps for fulfilling dependencies for, and setting up the f5aws tool. Once you have configured the environment from which you will run the f5aws tool, the final steps in the setup process require that you create an SSH key pair in AWS, and that you accept the End User License Agreement ("EULA") for all software you will use in the AWS Marketplace.

AWS Pre-requisites

Acquiring your AWS Access Keys

This tool requires AWS Access Keys for IAM users with sufficient privileges to create VPCs, launch CFTs, launch AMIs, etc. To obtain your `aws_access_key` and `aws_secret_key`, talk to your administrator. For those with sufficient permissions, the following steps guide you through the process of creating a set of AWS Access Keys. We suggest creating a role with permissions equivalent to those shown in Figure 8. More granular permissions are likely possible, but the authors have not performed this investigation.



```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "NotAction": "iam:*",
7       "Resource": "*"
8     }
9   ]
10 }
```

Figure 8 - Showing user the policy document configured within AWS IAM required for this tool.

Figure 9 below shows where you can configure this role within the IAM service in the AWS web portal. To create access keys for a specific user, visit AWS Console -> IAM - Users -> Create Access Keys.

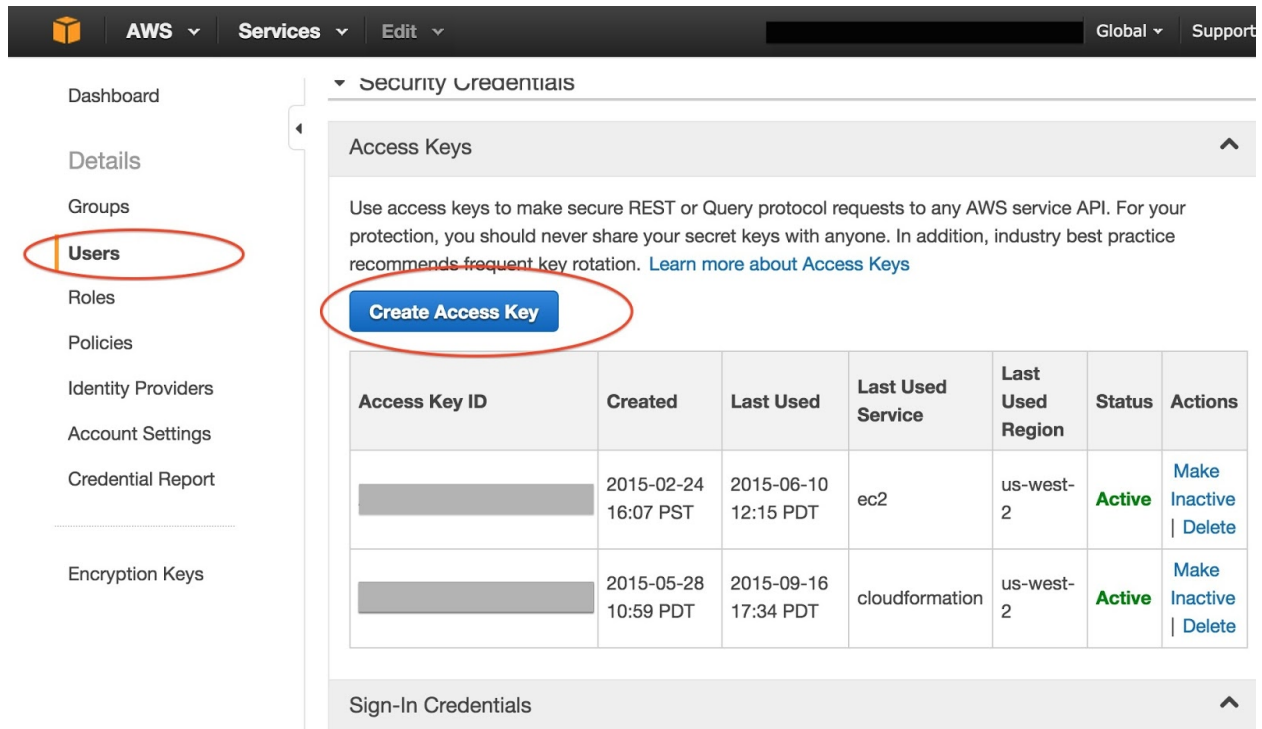


Figure 9 - Showing how to obtain your AWS Access keys from AWS Console

These credentials will be used by the Python Boto library which Ansible leverages to make API calls to AWS.

Creating an SSH Key Pair

Amazon requires that initial access to EC2 instances is performed via SSH public/private key pairs. Ansible will use this ssh key to access the AMI instances which are provisioned as part of the deployment script.

To create a new key pair, visit the AWS Console. At the top right, select the EC2 region you will be working within. Key pairs you create are specific to each region.

Next go to EC2 -> Network & Security -> Key Pairs (Create or Import).

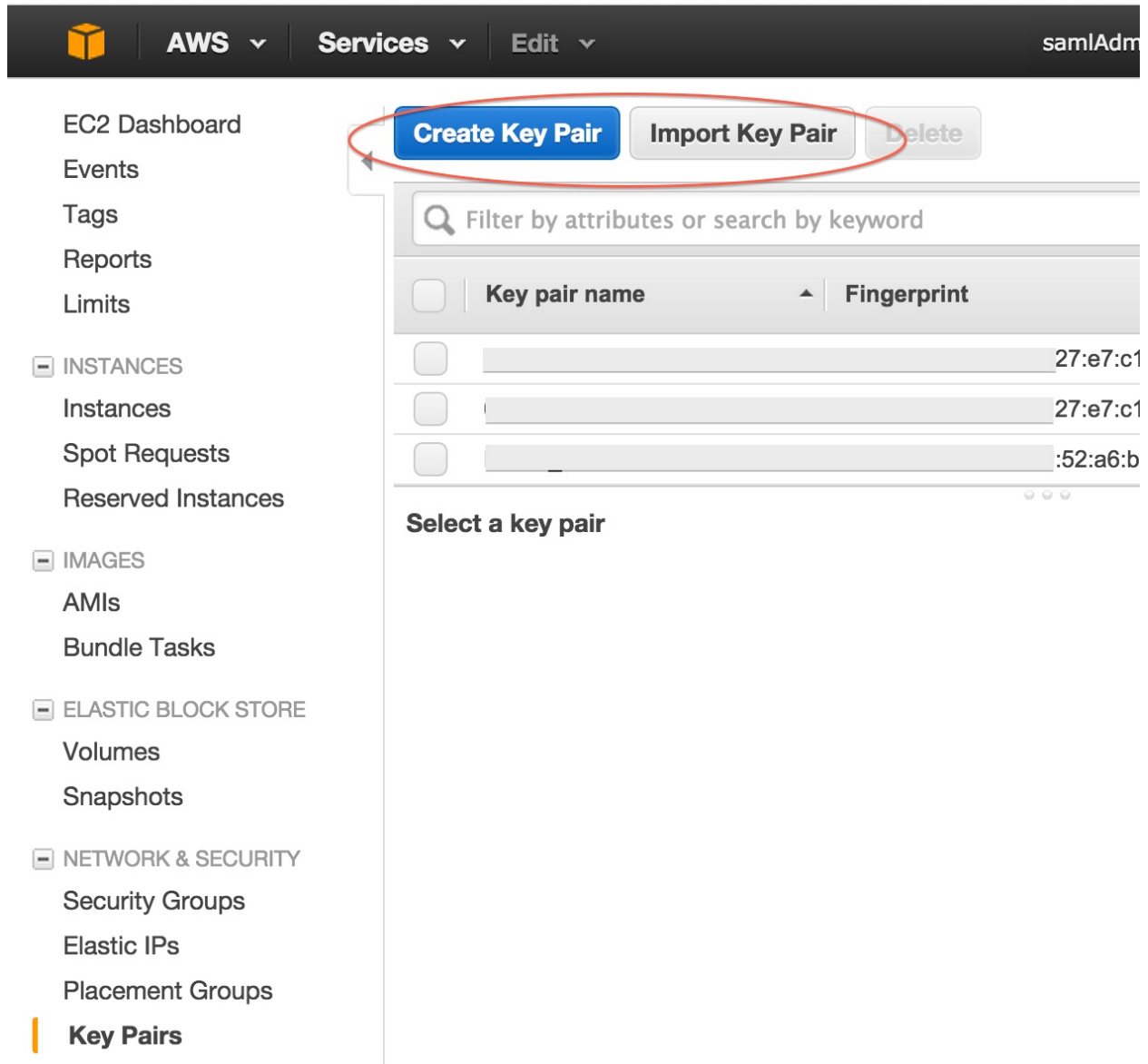


Figure 10 - Showing the obtaining the AWS SSH keys from AWS Console

Download the code

There are two options for downloading the code for use from Github.com:

Clone the code

In your terminal, run:

```
bash$> git clone https://github.com/F5Networks/aws-deployments.git
```

OR Download as compressed zip

Using the “Download zip” link found on github.com/F5Networks/aws-deployments, download and unzip as necessary.

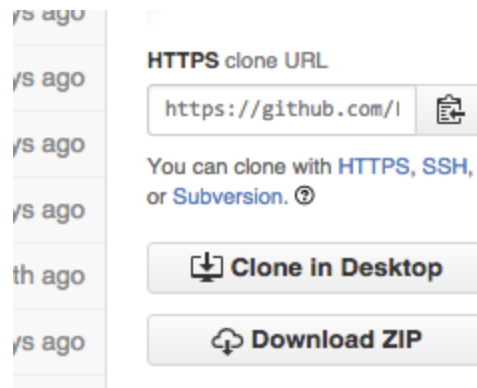


Figure 11 - Showing the multiple options for downloading the code on Github

Setting up the automation host environment

The following two mutually exclusive options are available for setting up host on which you will run Ansible. In both cases, the end result what we will refer to as the automation host. Using CFTs we have provided to get started quickly is the easiest approach. Vagrant and VirtualBox provide an approach which will allow you to run Ansible locally, but may limit users to certain operating systems.

Quick start using CloudFormation templates

In the top-level directory of the aws-deployments project you have cloned from Github, you will find two CloudFormation templates we have provided to get you quickly started in testing this project. These are automation_host_cft-w-existing-vpc.json and automation_host_cft-w-new-vpc.

These CFTs can be used to launch an automation host in AWS which will run the Ansible code.

Navigate to the AWS Portal, then to the CloudFormation page.

Choose “Create Stack”.

Upload one of the following CFTs, depending on your requirements:

- automation_host_cft-w-existing-vpc.json - Use this CFT if you already have a VPC and subnet in which you would like to install the ansible host.
- automation_host_cft-w-new-vpc.json - Use this CFT if you do not have a VPC and subnet available for use. These will be created within the CloudFormation stack deployed from this template, and the Ansible host will be provisioned inside of them.

Define the parameters for the new stack

- Stack name: "<first initial><lastname>-automation-host"
- InstanceType: m3.medium
- KeyName: <Select your SSH Key Pair>
- Subnet: <Subnet from Step 2, relevant only for first CFT above>
- VPC: <VPC from Step 2, relevant only for first CFT above>

Create tags for your CloudFormation stack as necessary and deploy the stack.

Monitor the status of the deployment. You can view whether the stack deployment has completed.

Once the control host created via the CloudFormation template is up and running, you need to login to setup the host.

Under the "OUTPUTS" tab for the stack you created in step 2.3.1, find the row labeled "PublicIp"

Log into the instance via the key pair you provided:

```
ssh -i <location of your private key> ubuntu@<value of PublicIp>
```

OR Setup using Vagrant and VirtualBox

Change into the vagrant directory with the aws-deployments project:

```
bash$> cd ./vagrant
```

Run the vagrant up command. This will launch a VirtualBox machine and execute a set of scripts within the VM on startup.

```
bash$> vagrant up
```

During the machine startup process, you will be prompted to provide an interface to use for the virtual machine. Choose one with internet access.

Once the vagrant up process is complete, SSH into the machine:

```
bash$> vagrant ssh
```

Configuring settings on your automation host

At this point, you should be logged into the automation host that you created using either Vagrant + VirtualBox or CloudFormation stack as described above.

Installing your SSH Private Key

Create a file `~/.ssh/<your private key>.pem` and include the contents of your private key. Remember to change permissions to RO (`chmod 400 <key file>`).

Ex. working in Vagrant host

```
(venv)vagrant@f5demo:/aws-deployments$ mkdir ~/.ssh/  
(venv)vagrant@f5demo:/aws-deployments$ vi ~/.ssh/My-AWS-SSH-Private-Key.pem  
(venv)vagrant@f5demo:/aws-deployments$ chmod 400 ~/.ssh/My-AWS-SSH-Private-Key.pem
```

NOTE: We extract the AWS label for the key from the filename provided so the name of this key must match label as seen in the AWS Console.

ex.If

“My-AWS-SSH-Private-Key” = seen in AWS Console

My-AWS-SSH-Private-Key.pem = name of key in .f5aws config

Adding your AWS Credentials for use boto and the AWS CLI

Edit `~/.aws/credentials` to include your AWS access and secret keys. Additional instructions on this can be found here:

<http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html#cli-config-files>

Specifying AWS credentials and private key location for the f5aws code

Edit `~/.f5aws` to provide the following information:

ssh_key: the full path to the ssh key above, i.e. `~/.ssh/<your private key>.pem`

bigip_rest_user: user for BIG-IP account that will be created by Ansible, and used for all iControlRest calls

bigip_rest_password: password for BIG-IP account that will be created by Ansible, and used for all iControlRest calls. For simplicity it will change the default “admin” user’s password to this as well.

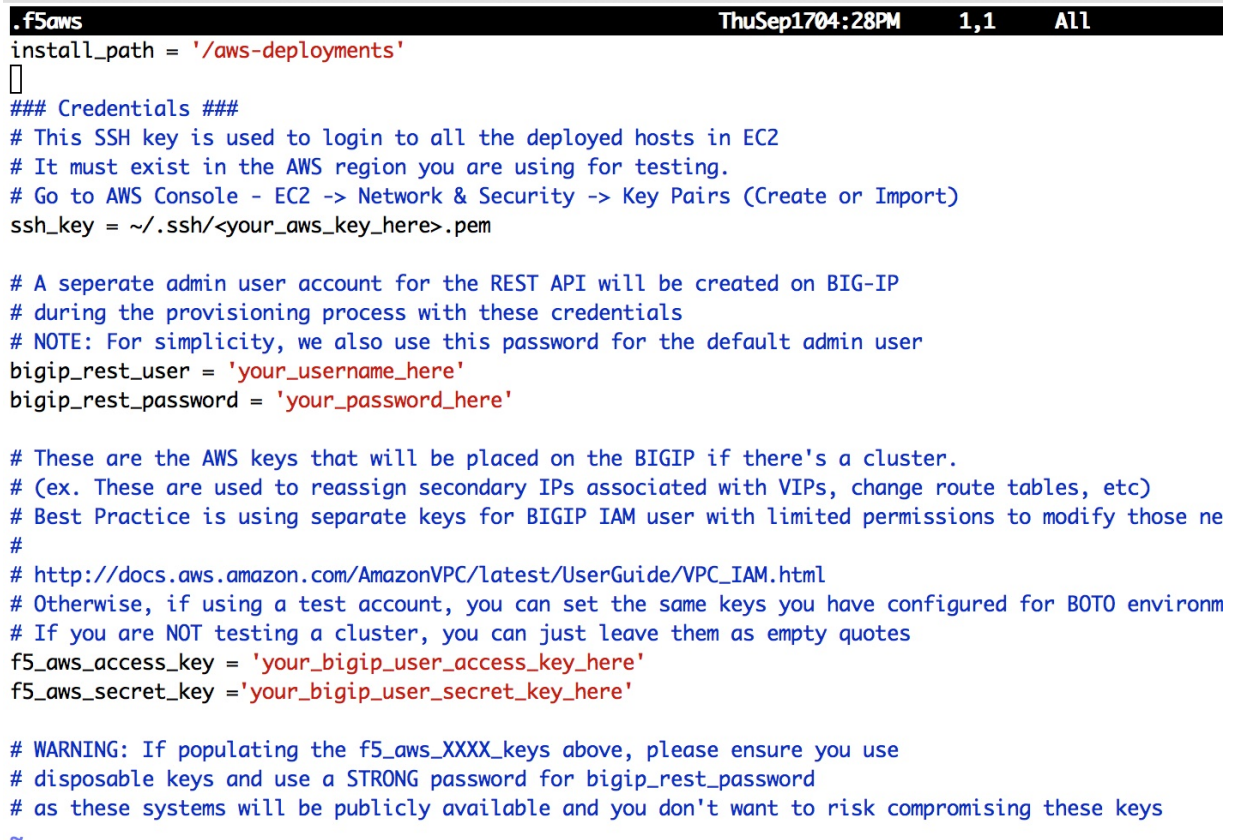
f5_aws_access_key: AWS access key that BIG-IPs will use when making API calls to AWS

f5_aws_secret_key: AWS secret key that BIG-IPs will use when making API calls to AWS

Note that these last two items may be the same or different than those you provided in ~/.aws/credentials above, depending on your security and user account posture.

WARNING: If AWS Access keys are provided for BIG-IP, remember that these devices will be publicly available, so please make sure to use strong passwords so these keys (even if just for testing) will not be easily compromised.

Additional notes/instructions are provided in the file itself.



```
.f5aws ThuSep1704:28PM 1,1 All
install_path = '/aws-deployments'
[]
### Credentials ###
# This SSH key is used to login to all the deployed hosts in EC2
# It must exist in the AWS region you are using for testing.
# Go to AWS Console - EC2 -> Network & Security -> Key Pairs (Create or Import)
ssh_key = ~/.ssh/<your_aws_key_here>.pem

# A separate admin user account for the REST API will be created on BIG-IP
# during the provisioning process with these credentials
# NOTE: For simplicity, we also use this password for the default admin user
bigip_rest_user = 'your_username_here'
bigip_rest_password = 'your_password_here'

# These are the AWS keys that will be placed on the BIGIP if there's a cluster.
# (ex. These are used to reassign secondary IPs associated with VIPs, change route tables, etc)
# Best Practice is using separate keys for BIGIP IAM user with limited permissions to modify those ne
#
# http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_IAM.html
# Otherwise, if using a test account, you can set the same keys you have configured for BOTO environm
# If you are NOT testing a cluster, you can just leave them as empty quotes
f5_aws_access_key = 'your_bigip_user_access_key_here'
f5_aws_secret_key = 'your_bigip_user_secret_key_here'

# WARNING: If populating the f5_aws_XXXX_keys above, please ensure you use
# disposable keys and use a STRONG password for bigip_rest_password
# as these systems will be publicly available and you don't want to risk compromising these keys
~
```

Figure 12 - Contents of the ~/.f5aws file which must be edited.

Accepting EULA Agreements

Before launching some of the AMIs used in this demo using any kind of AWS API (CLI, Boto, etc), the EULA must be accepted in the AWS marketplace web portal. Below, the photo shows how to accept the license terms for a BIG-IP 1Gbps Good image. For the BIG-IPs, this process is required for each AMI type (i.e. good 25mpbs, better 25mbps, best 25bmps, good 50 mbps,...).

Note: The AMIs leveraged will inevitably change over time. To accept the terms for the AMIs currently in use, please visit links below:

BIG-IP: F5 BIG-IP Virtual Edition 25 Mbps - Better:
<https://aws.amazon.com/marketplace/pp/B00JL3Q2VI>

The screenshot displays the AWS Marketplace interface for the F5 BIG-IP Virtual Edition 1Gbps - Good AMI. The page is titled "Launch on EC2: F5 BIG-IP Virtual Edition 1Gbps - Good". The "Manual Launch" section is active, showing instructions to click "Accept Terms" to gain access to the software. A red circle highlights the "Accept Terms" button in the "Price for your selections" section. Below this, the "Software Pricing" section shows the "Subscription Term" set to "Hourly" and the "Applicable Instance Type" set to "Software fee" with a "Varies" status. The "Pricing Details" section shows the region set to "US East (N. Virginia)" and includes a "Free Trial" section stating that users can try one instance for 30 days with no software charges, though AWS infrastructure charges still apply. The "Hourly Fees" section notes that total hourly fees will vary by instance type and EC2 region.

Figure 13 - EULA Acceptance for the 1Gbps Good AMI.

NOTE: If need to change, the AMI-IDs can also be found in various Cloud Formation Templates located in:

aws-deployments/roles/infra/files/*

Requesting Additional IP Address From Amazon (optional)

Some of the deployment topologies implemented in this demonstration tool may require a larger amount resources your account has by default. To deploy and explore these topologies, you may need to request a limit increase for EIPs + Cloudformation templates. See below topologies for the various resource requirements.

Tool Usage

Once you have completed the environment setup steps above, you are ready to run the f5aws script in ./bin to deploy BIG-IP in a number of different configurations.

There are a number of commands provided by the f5aws CLI. Here we walk through those in detail:

Command: init

Usage: f5aws init <env name> --extra-vars '{...deployment options...}'

This command initializes the ansible inventory for a new deployment. There are a number of parameters that may be used to specify details for your deployment.

Deployment model, region, and availability zones

Define the deployment model, region, and availability zones you would like to deploy and provide a name for your environment as the first position argument. Default variables are provided in ./roles/inventory_manager/defaults/main.yml.

Example of 'init' for a 'single-standalone topology' called demo-standalone-bigip:

```
./bin/f5aws init demo-standalone-bigip --extra-vars '{"deployment_model": "single-standalone",  
"region": "us-east-1", "zone": "us-east-1b"}'
```

Example of 'init' for a 'single-cluster topology' called demo-standalone-cluster:

```
./bin/f5aws init demo-standalone-cluster --extra-vars '{"deployment_model": "single-cluster",  
"region": "us-east-1", "zone": "us-east-1b"}'
```

Example of 'init' for a 'standalone-per-zone' topology called demo-standalone-per-zone:

```
./bin/f5aws init demo-standalone-per-zone --extra-vars '{"deployment_model":  
"standalone-per-zone", "region": "us-east-1", "zones": ["us-east-1b", "us-east-1c"]}'
```

Example of 'init' for a 'cluster-per-zone' topology called demo-cluster-per-zone:

```
./bin/f5aws init demo-cluster-per-zone --extra-vars '{"deployment_model": "cluster-per-zone",  
"region": "us-east-1", "zones": ["us-east-1b", "us-east-1c"]}'
```

Deployment Type

For any of the above deployments models, you may choose one of the following deployment types:

1. "lb_only" - Deploys only BIG-IP LTM for traffic management and load balancing at layers 4-7.
2. "lb_and_waf" - In addition to features installed and configured for #1 above, if this option is used, a web application firewall is deployed using BIG-IP ASM. While we recognize that WAF policies should be tailored to meet the specific needs of application protected, we use a generic linux WAF policy as an opportunity to demonstrate some very simple, out-of-the-box vulnerability blocking capabilities.

The default deployment type is "lb_only", and you may skip this flag if you wish to deploy this default. To set deployment type to "lb_and_waf", add this argument. As an example:

```
./bin/f5aws init demo-standalone-per-zone --extra-vars '{"deployment_model":
"standalone-per-zone", "deployment_type": "lb_and_waf", "region": "us-east-1", "zones":
["us-east-1b", "us-east-1c"]}'
```

Deploy Analytics

To deploy Splunk as a visualization and analytics tool for your deployment, pass the "deploy_analytics": "true" flag. This is also possible for any of the deployment models you choose.

```
./bin/f5aws init demo-standalone-per-zone --extra-vars '{"deployment_model":
"standalone-per-zone", "deploy_analytics": "true", "region": "us-east-1", "zones":
["us-east-1b", "us-east-1c"]}'
```

Command: deploy

Usage: f5aws deploy <env name>

Deploy, or redeploy an environment which has been initialized. This command takes only the name of the environment as defined when it was initialized. This command runs a set of playbooks, whose output will be printed to standard out as it is executed. When the command execution completes, the playbooks that have been executed will be listed, along with the execution times. This command is designed to be idempotent and reentrant.

Command: list

Usage: f5aws list

Show all deployments, basic variables, the current state.

Command: info

f5aws info {inventory|resources|login} <env name>

Command to provide additional information about an environment. The 'inventory' subcommand prints the ansible inventory and variables. The resources subcommand prints the CloudFormation stacks that have or will be deployed as part of this deployment, along with associated output variables. The login subcommand prints login information for deployed hosts.

Command: start_traffic

`f5aws start_traffic {inventory|resources|login} <env name>`

If the client hosts was provisioned as part of the deployment mode, a JMeter client is started. This JMeter client will send HTTP request to `demo.example.com`, for which there is a GTM listener + WideIP configured.

Command: stop_traffic

`f5aws stop_traffic {inventory|resources|login} <env name>`

Stops the JMeter client that was started by the command above. Only relevant for deployment models where a client host was deployed.

Command: teardown

Usage: `f5aws teardown <env name>`

Teardown ALL resources associated with an environment. This will bring an environment back to the initialized state but will not delete it.

Command: remove

Usage: `f5aws remove <env name>`

Remove the ansible inventory associated with a deployment. After removal, the environment will no longer appear in the output of the ``list`` command.

Related notes on usage

As stated above, some deployment topologies require more EIPs than the basic account limits may provide. We suggest deploying the 'single-standalone' topology to explore the basics of running and automating BIG-IP in AWS if you have not increased your account limit.

After you have completed your testing efforts with the `f5aws` tool, be sure to delete any Elastic Block Store volumes. You can do this by visiting the EC2 web portal then clicking volumes in the side menu. Available volumes may be deleted unless they remain from other deployments.

Example End-to-End Usage

We recommend the following path for those who wish to try out this tool for the first time. This deployment model will require 6 EIPs.

- 1) Setup the AWS prerequisites and automation host as discussed above.
- 2) Launch the standalone-per-zone deployment model with ASM and Splunk:
 - `./bin/f5aws init demo-standalone-per-zone --extra-vars '{"deployment_model": "standalone-per-zone", "deployment_type": "lb_and_waf", "deploy_analytics": "true", "region": "us-east-1", "zones": ["us-east-1b", "us-east-1c"]}'`
- 3) Deploy the environment
 - `./bin/f5aws deploy demo-standalone-per-zone`
- 4) Start traffic via JMeter
 - `./bin/f5aws start_traffic demo-standalone-per-zone`
- 5) Execute the `login info` command to get the login information for all the resources that have been deployed
 - `./bin/f5aws info login demo-standalone-per-zone`
- 6) Log into Splunk to view the traffic management and security plugins
 - In output from the `info login` command, find the value for `"analyticshost"` > `"<your availability zone>/zone1-analyticshost1"` > `"http"`.
 - Visit this URL in your browser.
 - Login via username `"admin"`, and password you entered for `bigip_password` in the `~/f5aws` file. Skip the password reset page.
 - Navigate to the Top Security Events page
 - i) Click Splunk for F5 Security (side nav bar)
 - ii) Click Application Security Manager (top nav bar) > Top Security Event Stats > Top Security Events
 - Navigate to the Top Security Events page
 - i) Click Splunk for F5 Networks (side nav bar)
 - ii) Click Web Access Statistics (top nav bar) > Traffic by Request
- 7) Login into a BIG-IP running LTM + ASM
 - In output from the `info login` command, find the value for `"bigip"` > `"<your availability zone>/zone1-bigip1"` > `"https"`.
 - Visit this URL in your browser.
 - Login via username `"admin"`, and password you entered for `bigip_password` in the `~/f5aws` file.
 - View the virtual servers
 - i) Navigate to Local Traffic > Virtual Servers
 - View the iApps
 - i) Navigate to iApps > Application Services > Vip1_iApp
- 8) Login into a BIG-IP running GTM

- In output from the `info login` command, find the value for “gtm” > “<your availability zone>/zone1-gtm1” > “https”.
- Visit this URL in your browser.
- Login via username “admin, and password you entered for bigip_password in the ~/.f5aws file.
- View the configured GSLB datacenters
 - i) Navigate to DNS > Datacenters > Servers
- View the configured WideIPs
 - i) Navigate to DNS > GSLB > WideIPs > demo.example.com
- 9) Visit the application directly (HTTP, no BIG-IP in-line)
 - In output from the `info login` command, find the value for “apphost” > “<your availability zone>/zone1-gtm1” > “http”.
 - Try a simple vulnerability, e.g. curl -sk -X GET -H 'Content-type: application/json; ls /usr/bin' <the HTTP address
- 10) Visit the application through BIG-IP virtual server
 - In output from the `info login` command, find the value for “apphost” > “<your availability zone>/zone1-bigip1” > “elastic_ips” > “eipAddress”
 - Try a simple vulnerability, e.g. curl -sk -X GET -H 'Content-type: application/json; ls /usr/bin' <the eipAddress>
- 11) Teardown the environment when you are complete with your exploration
 - ./bin/f5aws deploy demo-standalone-per-zone
- 12) Teardown Vagrant or the automation host CFT you deployed as necessary.

Architecture

There are several excellent configuration management tools that are popular in the devops community (ex. Chef, Puppet, Salt, Ansible, etc.) but the goals of this project were not to promote any one management tool vs. another. As stated earlier, the goals were to provide an opportunity to easily test various deployment models in AWS as well as illustrate how BIG-IP services can be deployed and automated using some of these tools.

There are also many other factors that are important to selecting a deployment model, including BIG-IP licensing, version, and images. For this particular project, we chose BIG-IP 11.6.0 Better 25 Mbps Hourly/Subscription images because:

- 1) At the time of writing, 11.6.0 was the latest release available
- 2) It was the most economical image to demonstrate advanced functionality
- 3) Hourly / Subscription had low barrier to entry, allowing perfect self service evaluation.

Although we have shared some of the decisions we make for this particular project, we strongly recommend that you speak with your local Field Engineer to discuss what may be best for your particular environment or situation.

Deployment models

The initial deployments this tool creates are what we have called:

1) single-standalone

- This is the simplest deployment and has the smallest footprint.
- Will provision a BIG-IP and allow users to login and inspect a working BIG-IP environment.
- It creates a bare-minimum BIG-IP deployment w/
 - 1 VPC
 - 4 subnets in a single AZ
 - 1 BIG-IP w/
 - 3 EIPs (for BIG-IP mgmt IP, unique Self-IP, VIP1)
 - 1 Docker Host (w/ 2 containers)

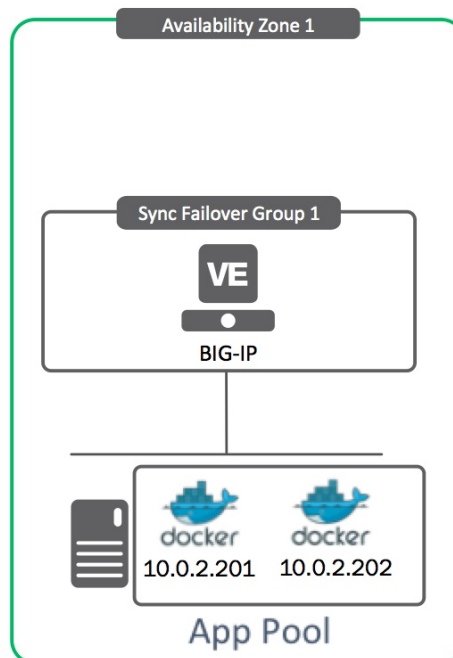


Figure 14 - 'single-standalone' deployment model

2) single-cluster

- This is the next simplest deployment
- Will provision a BIG-IP cluster and allow users to login and inspect a working BIG-IP clustered environment.
- It creates a bare-minimum BIG-IP deployment w/
 - 1 VPC
 - 4 subnets in a single AZ
 - 2 BIG-IPs w/
 - 4 EIPs
 - 1 for each BIG-IP mgmt IP
 - 1 for each unique public Self-IP
 - 1 for a shared VIP1
 - 1 Docker Host (w/ 2 containers)

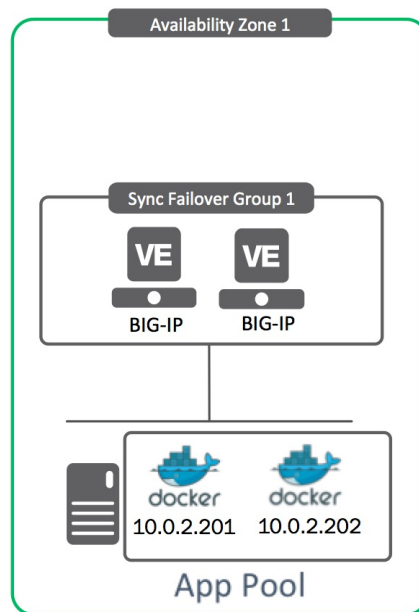


Figure 15 - 'single-cluster' deployment model

3) standalone-per-zone

- This is a slightly more complex deployment, adding a client (ubuntu w/ jmeter) and gtm to the above. In addition, it provides an option to scale out that same deployment to multiple AZs.
- The tool will provision BIGIP(s) and GTM(s), allowing users to login and inspect a working BIG-IP + GTM deployment that can spans across AZs. It also provides ability to easily generate traffic.
- It creates an example BIG-IP deployment w/
 - 1 VPC
 - 1 Client Host (w/ Jmeter)

PER Availability Zone:

- 4 subnets
- 1 BIG-IP
 - 3 EIPs (for BIG-IP mgmt IP, unique Self-IP and VIP1) (* number of zones provided)
- 1 GTM (up to 2 total)
 - 3 EIPs (for BIG-IP mgmt IP, unique Self-IP, Listener)
- 1 Docker Host (w/ 2 containers)

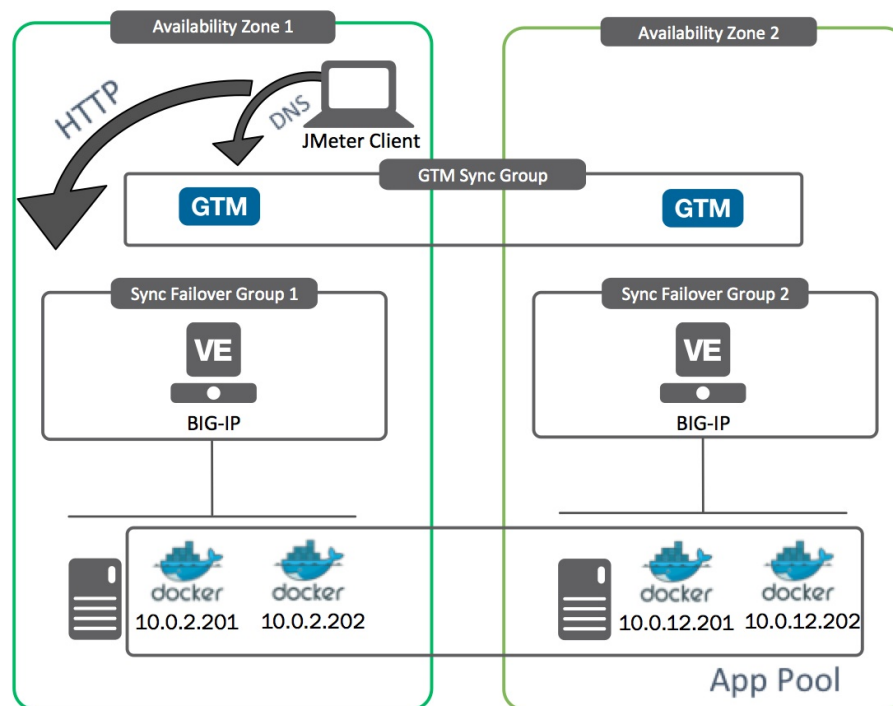


Figure 16 - 'standalone-per-zone' deployment model

4) cluster-per-zone

- Similar to the "standalone-per-zone" model except deploying a cluster in each AZ.
 - The tool will provision BIGIP(s) and GTM(s), allowing users to login and inspect a working BIG-IP + GTM deployment that can span across AZs. It also provides ability to easily generate traffic.
 - It creates an example BIG-IP deployment w/
 - 1 VPC
 - 1 Client Host (w/ Jmeter)
- PER Availability Zone:
- 4 subnets
 - 1 BIG-IP
 - 3 EIPs (for BIG-IP mgmt IP, unique Self-IP and VIP1) (* number of zones provided)
 - 1 GTM (up to 2 total)
 - 3 EIPs (for BIG-IP mgmt IP, unique Self-IP, Listener)
 - 1 Docker Host (w/ 2 containers)

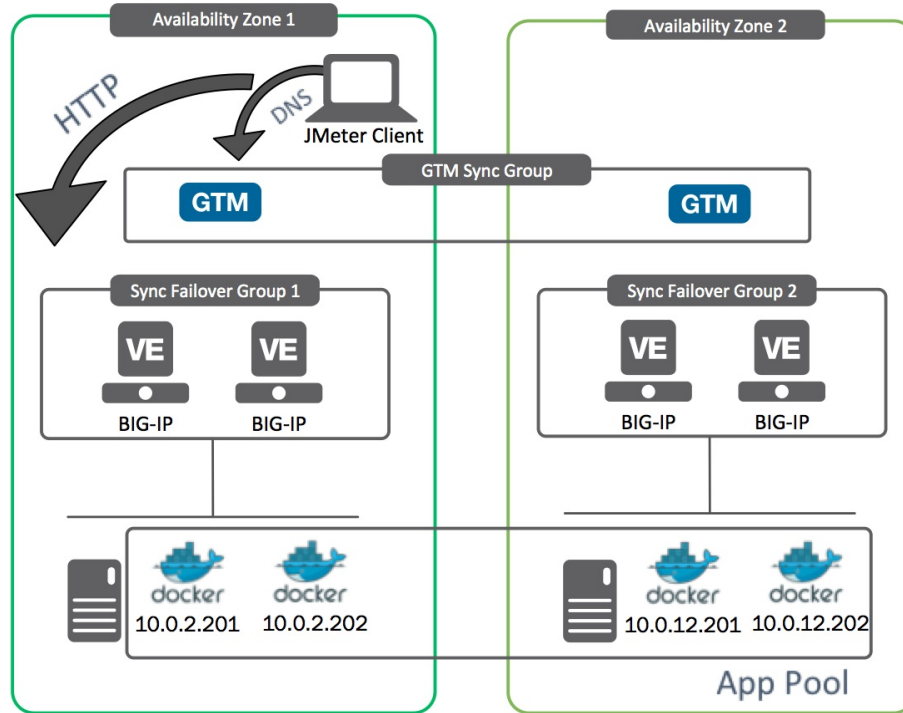


Figure 17 - 'cluster-per-zone' deployment model

Deployment Type

For all of the deployment models listed above, there are two deployment types you may choose:

Deploying Splunk

AWS Configuration Overview

NOTE: AWS has several best practice enterprise network architectures (ex. hub and spoke - where shared network services like BIG-IP/Firewall live in the hub) but that is out of scope for this particular conversation. Please engage your friendly F5 and AWS field engineers.

Typically in cloud, routed architectures are preferred. This particular deployment uses 4 subnets (management, public, private, and application). The BIG-IPs are directly connected to the management, public and private subnets.

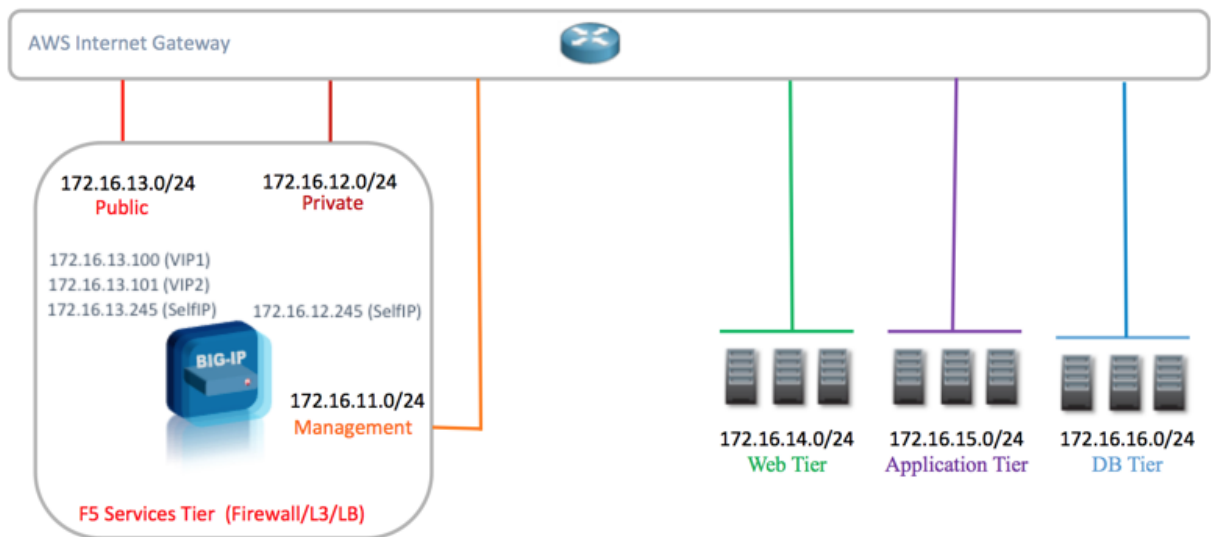


Figure 18 - Multi-tier routed architecture

Other sensible options just have management and public subnets (which we call “one-armed”). In this deployment, we use the private subnet to provide the traditional inline “airgap” design, providing a clear separation of external and internal traffic.

BIGIP-Configuration Overview

In addition to the infrastructure, the tool also demonstrates how the lifecycle of BIG-IP can be automated.

Use of Ansible with iControlREST

The following section discusses how we have logically organized BIG-IP provisioning steps in various playbooks and tasks. Before discussing those items, it is important to understand how we are using Ansible at a low-level to configure BIG-IP using iControlREST, and why we have made the given design decisions.

In our Ansible inventory, which can be found in `~/vars/f5aws/env/<name of your env>/inventory`, you will see that we make a distinction between BIG-IPs which are provisioned to run LTM, and BIG-IPs which are provisioned to run GTM. We refer to the former ansible host group as ‘bigips’,

and the latter host group as 'gtms'. Before we examine what is done for these different host groups, let's dive down deep to see how we are using iControlREST.

iCR is leveraged within most tasks in which we configure BIG-IP. One example is `./roles/bigip_network/tasks/main.yml`. In this file, you will see that each task makes use of a custom ansible module, "bigip_config". This module is a wrapper around iControlREST calls, and the code behind this module can be viewed in `./library/bigip_config.py`. The module takes as input user/password credentials, the DNS resolvable hostname of a BIG-IP (management port), a URL, and JSON payload. This rudimentary design approach is used to provide full inspection of the resources we are configuring within TMOS. A better approach might be to create custom ansible modules for different logical or procedural aspects of BIG-IP lifecycle management.

```
5
6 - name: Adding/updating internal vlan
7   delegate_to: localhost
8   bigip_config:
9     state=present
10    host={{ ansible_ssh_host }}
11    user={{ bigip_rest_user }}
12    password={{ bigip_rest_password }}
13    payload='{"name":"private", "interfaces":"1.2"}'
14    collection_path='mgmt/tm/net/vlan'
15    resource_key="name"
16
```

Figure 19 - Showing the use of our custom ansible module, "bigip_config", to create a VLAN.

Configuring BIG-IP Running LTM

For the configuration of deployed BIG-IPs which are not running GTM, the following four logical provisioning steps are performed. These groupings are mapped to ansible roles, which are applied to all BIG-IPs. These roles are applied in the `deploy_bigip.yml` playbook.

- 1) Device configuration
 - a) See `./roles/bigip_base`
 - b) Adds users via `tmsh`
 - c) This is the only step where we use SSH instead of iControlREST
- 2) System configuration
 - a) See `./roles/bigip_system`
 - b) Provisions system globals like NTP, DNS, SNMP, syslog, DB keys

- 3) AWS specific system configuration (found in ./roles/bigip_system_aws)
 - a) See ./roles/bigip_system_aws
 - b) Sets AWS keys and disables DHCP
- 4) Network configuration
 - a) See ./roles/bigip_network
 - b) Sets VLANs, self-IPs, routes, which tend to differ on a device-by-device basis.

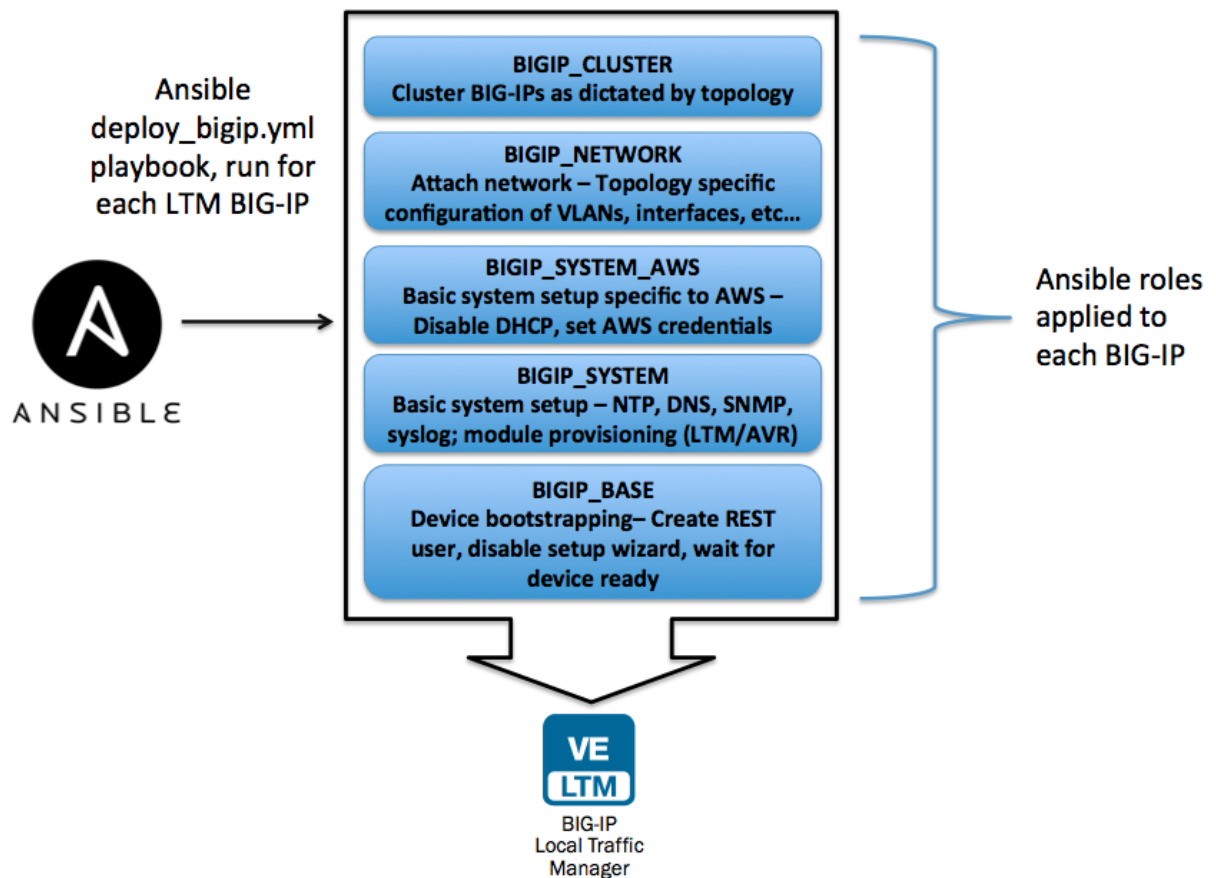


Figure 20 - Showing how the LTM configuration on BIG-IP is deployed using Ansible roles

Configuring BIG-IP Running GTM

When deploying BIG-IP running GTM, we run apply some of the same configuration steps with a few differences:

- 1) Device configuration
 - a) See `./roles/bigip_base`
 - b) Adds users via `tmsh`
 - c) This is the only step where we use SSH instead of `iControlREST`
- 2) System configuration
 - a) See `./roles/bigip_system`
 - b) Provisions system globals like NTP, DNS, SNMP, syslog, DB keys
- 3) AWS specific system configuration (found in `./roles/bigip_system_aws`)
 - a) See `./roles/bigip_system_aws`
 - b) Sets AWS keys and disables DHCP
- 4) Network configuration
 - a) See `./roles/gtm_network`
 - b) Sets VLANs, self-IPs, routes, which tend to differ on a device-by-device basis.
- 5) GTM system
 - a) See `./roles/gtm_system`
 - b) Provisions GTM
- 6) GTM Configuration
 - a) See `./roles/gtm_conf`
 - b) Setup gtm configuration for this network topolog
- 7) GTM clustering
 - a) See `./roles/gtm_cluster`
 - b) Setup up these BIG-IPs in cluster to share route information/status

The additional steps completed for GTM configuration are due to the need to setup the TMOS GTM config globally, whereas for the BIG-IP running LTM, we will had further configuration for LTM objects on a per-virtual server basis.

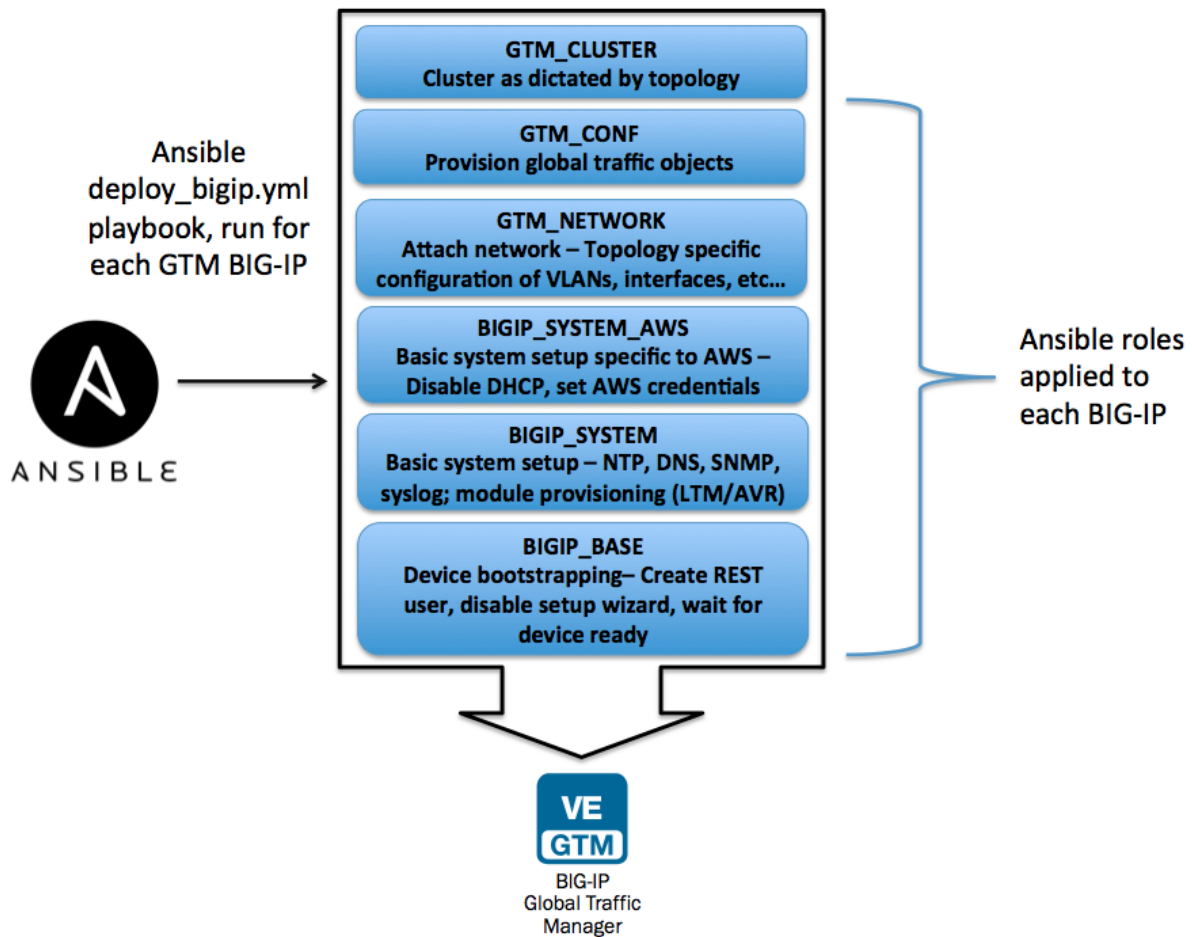


Figure 21 - Showing how the GTM configuration on BIG-IP is deployed using Ansible roles

Virtual Servers

As part of all deployment topologies, we create two virtual servers. These two virtuals are deployed within the `deploy_apps_bigip.yml` playbook, which makes use of the `bigip_app1` and `bigip_app2` roles.

Virtual 1 (bigip_app1 role)

For the first virtual server, iApps are leveraged to deploy the TMOS traffic configuration. Because the F5 HTTP iApp available on box does not include the ability to attach ASM policies, we use a newer version of the iApp available on DevCentral. An instance of the above iApp template is deployed for our application. The iApp service deployed in our example references several iRules, an existing pool, an ASM policy, and an ASM logging profile. A template of the iApp we deploy is available in: `/roles/bigip_app1/templates/demo_iApp.cfg.j2`, and the JSON

representation of the HTTP iApp is at
./roles/bigip_app1/files/iapp_f5_http_backport_template.json.

The role also deploys the supporting content for the iApp:

- 1) iRules including a few demonstrating analytics integration and displaying sorry page.
These iRules can be seen in ./roles/bigip_app1/files/irules_*
- 2) Background and sorry page images to an internal data-group.
These images can be found in ./roles/bigip_app/files/image_* (these are base64 encoded).

After deploying the iApp which contains the virtual server, we attach an EIP to the secondary IP address in Amazon. This secondary EIP matches the listener address of the virtual server.

Virtual 2 (bigip_app2 role)

For this second virtual, we do not use iApps. Instead, we directly use the bigip_config ansible module we have written. The module is used to deploy an iRule, virtual servers for 80 and 443, and an web server pool.

The iRule we deploy will randomly SNAT and redirect traffic to the first virtual server mentioned above. This helps to provide some interesting graphs for viewing traffic in the Analytics module or elsewhere.

Finally, for this virtual, we do not attach an EIP as we did previously. The virtual server will not be reachable from the public internet, but only within the VPC. This step is skipped in order to save on Elastic IP addresses, which are in short supply.

Handling Common Errors

Description: Various task timeouts:

Symptom:

When dealing with complex orchestration, you inevitably run into occasional timing issues. We tried to put sane default retries and timeouts but occasionally there may be a longer than expected blip or timeout. The playbooks are designed to be run over and over again so If you do see a random timeout, you can usually give it another try or two.

ex.

```
TASK: [bigip_system | Disabling Setup Utility in GUI]
*****
changed: [zone2-bigip1 -> localhost]
```

```
failed: [zone1-bigip1 -> localhost] => {"attempts": 20,
"failed": true, "name": "mgmt/tm/sys/db/setup.run", "rc": 1}
msg: Task failed as maximum retries was encountered
```

Description: Incorrect system time on vagrant VM causes fatal error when using the AWS CLI or boto libraries to interact with AWS EC2 endpoints.

Symptom:

Ansible tasks/playbooks which make calls into the Python Boto library fail. This includes use of the cloudformation ansible module for deployment/teardown of EC2 resources like the VPC.

```
TASK: [Creating Vpc]
*****
failed: [vpc-manager] => {"failed": true}
msg: Signature expired: 20150730T193401Z is now earlier than
20150730T194311Z (20150730T194811Z - 5 min.)
```

Fix:

Reset the system clock on your VM. On Ubuntu the command to do this is “sudo ntpdate ntp.ubuntu.com”

Description: CloudFormation Deployment Errors

Symptom:

There are several errors which may occur when deploying CloudFormation templates in AWS. Rather than outline all the possible errors here, the following general troubleshooting steps are suggested. When you see an ansible task fail like the one below, where the failure suggests that a CloudFormation stack failed, login the EC2 portal in AWS. Once logged in, visit the CloudFormation page/tab, and choose the relevant stack showing a failure. The stack will typically be in “Rollback Complete” stage.

```
failed: [zone1-client] => {"changed": true, "events":
["StackEvent AWS::CloudFormation::Stack clusterchris-zone1-client
ROLLBACK_COMPLETE", "StackEvent AWS::EC2::SecurityGroup
ClientSecurityGroup DELETE_COMPLETE", "StackEvent
AWS::EC2::SecurityGroup ClientSecurityGroup DELETE_IN_PROGRESS",
"StackEvent AWS::EC2::Instance ClientInstance DELETE_COMPLETE",
"StackEvent AWS::CloudFormation::Stack clusterchris-zone1-client
ROLLBACK_IN_PROGRESS", "StackEvent AWS::EC2::Instance
ClientInstance CREATE_FAILED", "StackEvent AWS::EC2::Instance
ClientInstance CREATE_IN_PROGRESS", "StackEvent
```



```

AWS::EC2::SecurityGroup      ClientSecurityGroup      CREATE_COMPLETE",
"StackEvent      AWS::EC2::SecurityGroup      ClientSecurityGroup
CREATE_IN_PROGRESS",      "StackEvent      AWS::EC2::SecurityGroup
ClientSecurityGroup      CREATE_IN_PROGRESS",      "StackEvent
AWS::CloudFormation::Stack      clusterchris-zone1-client
CREATE_IN_PROGRESS"], "failed": true, "output": "Problem with
CREATE. Rollback complete", "stack_outputs": {}

```

Filter: Active ▾		By Name: <input type="text"/>		
	Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/>	clusterchris-zone1-client	2015-08-19 15:20:03 UTC-0700	ROLLBACK_COMPLETE	AWS CloudFormatio
<input type="checkbox"/>	clusterchris-zone1-apphost1	2015-08-19 15:18:34 UTC-0700	CREATE_COMPLETE	AWS CloudFormatio

Showing a failed CloudFormation stack deployment where the status is "Rollback Complete"

Fix:

In order to determine the proper fix, we must determine why the CloudFormation stack failed to complete. You can see error messages by selecting the stack in question and viewing the Events tab as shown below. Once you have determined and made the appropriate fix, rerun the 'deploy' command.

Resources	Events	Template	Parameters	Tags	Stack Policy
Status ROLLBACK_COMPLETE DELETE_COMPLETE DELETE_IN_PROGRESS DELETE_COMPLETE ROLLBACK_IN_PROGRESS CREATE_FAILED CREATE_IN_PROGRESS		Type AWS::CloudFormation::Stack AWS::EC2::SecurityGroup AWS::EC2::SecurityGroup AWS::EC2::Instance AWS::CloudFormation::Stack AWS::EC2::Instance AWS::EC2::Instance		Logical ID clusterchris-zone1-client ClientSecurityGroup ClientSecurityGroup ClientInstance clusterchris-zone1-client ClientInstance ClientInstance	
				Status Reason The following resource(s) failed to create: [ClientInstan In order to use this AWS Marketplace product you nee please visit http://aws.amazon.com/marketplace/pp?s	

Improving this Project

As always, time and resources were limited, and there are many places in the design and implementation of this demonstration tool where we have taken shortcuts.

Areas for Improvement

Reduce EIPs

In this demonstration tool, the orchestration host (ansible) lives outside of the Amazon VPC in which we provision. We also are not using any kind of VPN gateway to create a privately routable connection to the hosts provisioned with the VPC. While this approach made initial setup of the test environment easy and more accessible, it requires that publicly routable interfaces are available for SSH for every host under management. For the traffic and client hosts, where the EC2 instances require only 1 interface, a “Public IP” can be attached to the interface. On EC2 instances with multiple interfaces like BIG-IP, CloudFormation templates require us to use Elastic IP addresses instead. For this reason, the management interface of each BIG-IP required an EIP which it may have not have had we initially launched the BIG-IP with one interface and added the additional interfaces afterwards. EIPs were also provisioned on external interfaces to be able to produce publicly accessible deployments. As an alternative, it would be nice to create internal-only deployments. However, that would require the user to provide an existing VPC and host from which to launch it (ex. an existing AMI in that environment). Instead of a an orchestration host with an environment, a VPN gateway could also be used. BIG-IP will also have a single interface version available soon which we’ll demonstrate as well.

Reduce CFTs

For each deployment model, individual CloudFormation templates are launched for various logical parts of the environment stack. For example, in the single-standalone model, separate CloudFormation templates are created for the VPC, for the networking elements in each availability zone, for BIG-IP, the application hosts, and the EIP associated with each VIP. In order to limit the number of CFTs required, we should ideally dynamically generate one monolithic CFT for each type of deployment model. That would require better variable naming.

Persistence Models

This tool uses a basic and limit persistent model which includes saving configuration data to JSON and YAML files local to the ansible control node. Production deployments would leverage

actual databases (CMDDBs). Ansible itself provides several ansible specific solutions like `fact_caching` (implemented using redis) as well as integrations to other [CMDDBs](#) which would make for cleaner, more powerful solution. Our focus on orchestrating BIG-IP and network services limited us from spending further time on this worthy improvement.

Clustering & Licensing

This initial release is focused on orchestration and basic functionality of standalones and clusters in a common scale out model. However, other deployments are planned that address licensing and clustering across AZs.

Contributing

Although not a supported project, we are excited about it and look forward to hearing what you think. Please feel free to kick the tires and submit feedback via the 'issues' feature on Github or to Alex Applebaum or Chris Mutzel via a.applebaum@f5.com or c.mutzel@f5.com.

Additional Resources

F5 Documentation, Articles, and Videos

Official Deployment Guide:

[BIG-IP Virtual Edition Setup Guide for Amazon EC2](#)

Youtube Video:

[Deploying a BIG-IP Virtual Edition HA Pair into AWS](#)

Inspiration

How cloud + virtualization is changing operation models

<http://www.ansible.com/blog/immutable-systems>

Why are we still running virtualenv in a container :-)

<https://hynek.me/articles/virtualenv-lives/>

<https://glyph.twistedmatrix.com/2015/03/docker-deploy-double-dutch.html>

From One to Many - Evolving VPC Design

<https://www.youtube.com/watch?v=GKSsJJqzyyI>

Technical

Install Python 2.7.9 on Ubuntu:

<https://renoirboulanger.com/blog/2015/04/upgrade-python-2-7-9-ubuntu-14-04-lts-making-deb-package/>

Installing Python 2.7.9 on Centos

<https://gist.github.com/jenmontes/a81e3b4ae0125a681e1f>

Install VirtualEnv:

<http://docs.python-guide.org/en/latest/dev/virtualenvs/>

Acknowledgements

Tim Rupp, our resident Ansible expert