# Orchestrating BIG-IP in the Public Cloud

Demonstrating various deployment models in AWS EC2 Using Ansible

Last updated: September 19th, 2015

Tai Tran, Director, Product Management, Virtual Edition and Cloud, t.tran@f5.com
Alex Applebaum, PME, F5 Networks, a.applebaum@f5.com
Chris Mutzel, PME, F5 Networks, c.mutzel@f5.com

# 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, and referenced sources are listed at the end of this document.

The relevant audience for this work includes those in networking, cloud, or system architecture, engineering, and administration roles who wish to learn more about how to deploy BIG-IP in public cloud environments, and how to manage the lifecycle of BIG-IP in a automated manner. There are two primaries goals for this project.

1. Provide the opportunity to easily test deployment models 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 architectural conversation, focus should be placed on Vagrant/VirtualBox or Docker dependencies used to setup the test environment. These are the only technologies which need to be explicitly configured 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 templates as a useful way to deploy virtual resources in many cloud environments.

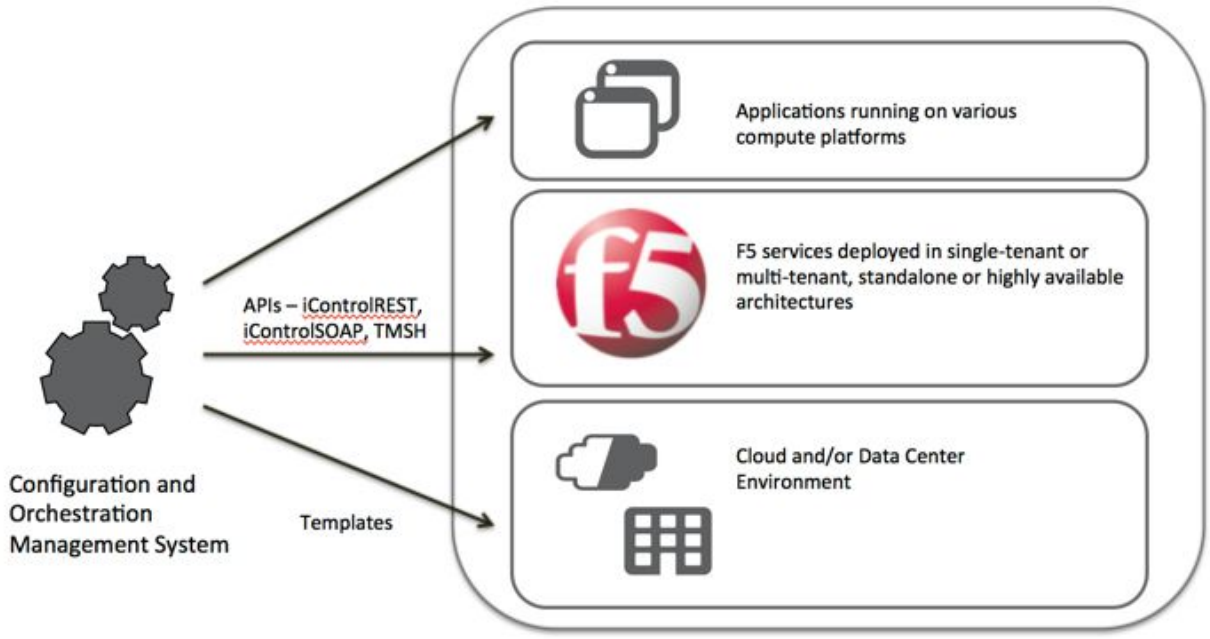



Figure 1 - High-level architecture of this project, representing the generic deployment of applications and network services on top of virtualized or programmable infrastructure.

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.

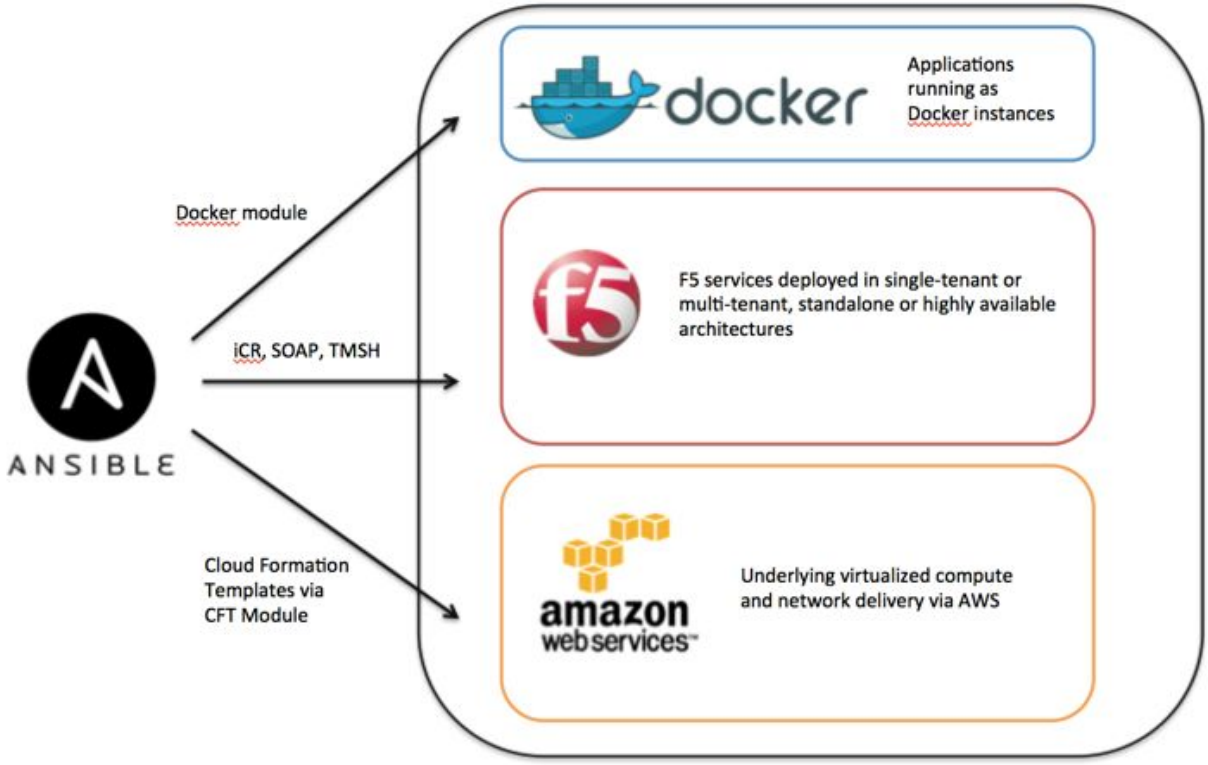



Figure 2 - Technologies chosen to implement a proof of concept, showing how the BIG-IP VE footprint and service can be fully automated.

The following descriptions discuss how these and other technologies are used. The end product of this work is a python script which provides an API for fully 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.

## Amazon Web Services

In 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

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 prototype 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 me 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 host.

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, application hosts. 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*, which 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 across one or more Docker containers. We use an Elastic Compute Service (ECS) optimized host in EC2 for running the Docker daemon and container processes.

The other use case for Docker in this project is as a way to make this project portable for execution across user environments. Along with the Vagrant/VirtualBox packaging described later, you may use Docker as a way to configure a test environment for our example code.

## JMeter

In some deployment models with the aws-deployments tool, traffic may be generated to demonstrate reporting and analytics technology. 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. iCR does not include functionality for device licensing, or clustering. We will 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.

## Operating Systems, Python and associated Packages

When using Ansible for configuration management, a central host is required from which ssh probes will be launched to configure remote hosts. We have used a Ubuntu base image that is launched within a vagrant environment (see below) to fill this role. 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 Docker or Vagrant, 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 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 provide ease-of-use for this toolchain, 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 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.

# Environment Setup

In this section we document the necessary steps for installing the dependencies for this running the `f5aws` script in ./bin. There are three approaches for configuring your environment, and we have attempted to make it as easy as possible quickly get started. 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.

## Downloading the Code

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

### 1.a) Clone the code

In your terminal, run:

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

### 1.b) Download as compressed zip

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

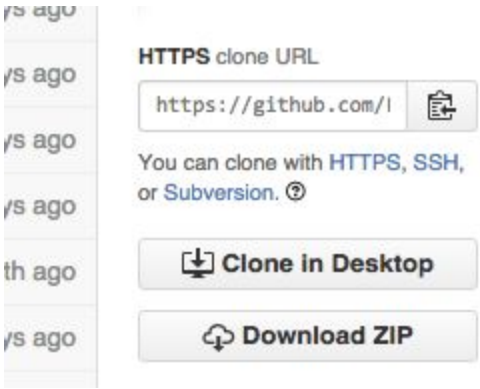



Figure 3 - Showing the multiple options for downloading the code on Github

### 2) Change into the directory

*bash$> cd ./aws-deployments*

## Setting up the test environment

The following three mutually exclusive options are available for setting up an environment which includes the project dependences. In all three cases, the end result is a test environment controller host. Vagrant and VirtualBox provide the easiest approach, but may limit users to certain operating systems. We also provide a more generic setup option using Docker for better operating system portability. Lastly the steps for complete manual environment configuration are provided for completeness though we hope these will not be necessary.

### 3.a) Setup using Vagrant and VirtualBox

Change into the vagrant directory:

*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*

### OR 3.b) Setup using Docker

Change into the docker directory:

*bash$> cd ./docker*

Build a Docker container image from the DockerFile (sudo command not necessary when using boot2docker):

*bash$> sudo docker build -t f5demo .*

Launch the Docker container (sudo command not necessary when using boot2docker). This will start the container in interactive mode and bring up a prompt within the container process:

*bash$> sudo docker run -v <local path to aws-deployments>:/aws-deployments -i -t f5demo bash --rcfile /venv/bin/activate*

*ex.*
From an ECS-Optimized AMI in AWS:
*[ec2-user@ip-172-16-2-146 docker]$ sudo docker run -v /home/ec2-user/aws-deployments:/aws-deployments -i -t f5demo bash --rcfile /venv/bin/activate*
*(venv)root@76ded817f346:/aws-deployments#*

*NOTE: You may feel like you're trapped in the movie "Inception" but we even installed docker in the vagrant "f5demo" image itself so if you would like to test using a docker environment as well, you can run the above from inside there.*

From within the "f5demo" vagrant Image:
*(venv)vagrant@f5demo:/aws-deployments/docker$ sudo docker run -v /aws-deployments:/aws-deployments -i -t f5demo bash --rcfile /venv/bin/activate*
*(venv)root@7dbc2465cb3b:/aws-deployments#*

### OR 3.c) Manual environment setup

1) Install python 2.7.9 (see relevant blog post for this step in the appendix)
2) Install python virtualenv and activate venv as necessary (optional)
3) Install the python modules listed in ./requirements.txt

## Configuring environment settings

### 4) Setting the AWS Credentials for boto and AWS CLI

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.

***AWS Console - IAM - Users -> Create Access Keys***

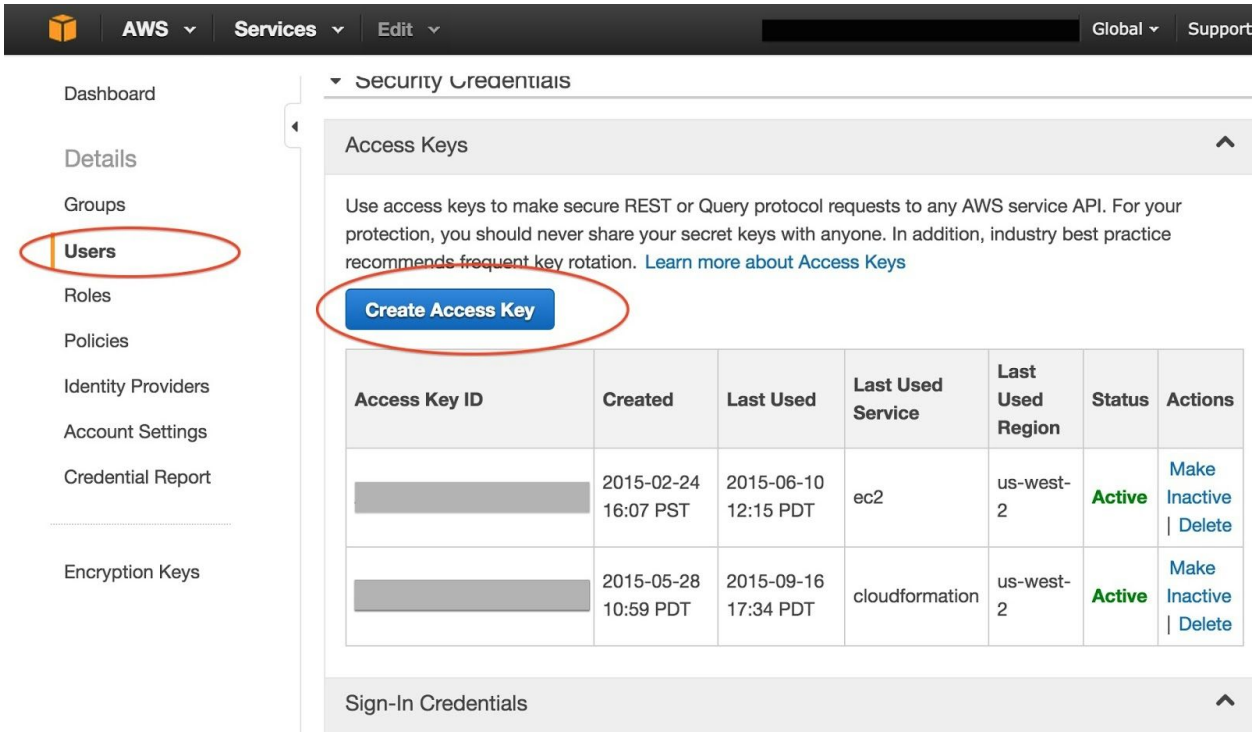



Figure 4 - Showing the obtaining the AWS Access eys from AWS Console

Once the Access Keys are obtained, edit ~/.aws/credentials with your favorite editor (nano,pico,vi) to include values for the aws_access_key and aws_secret_key parameters.

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

## 5) Adding SSH Private Key

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.

***AWS Console - EC2 -> Network & Security -> Key Pairs (Create or Import)***

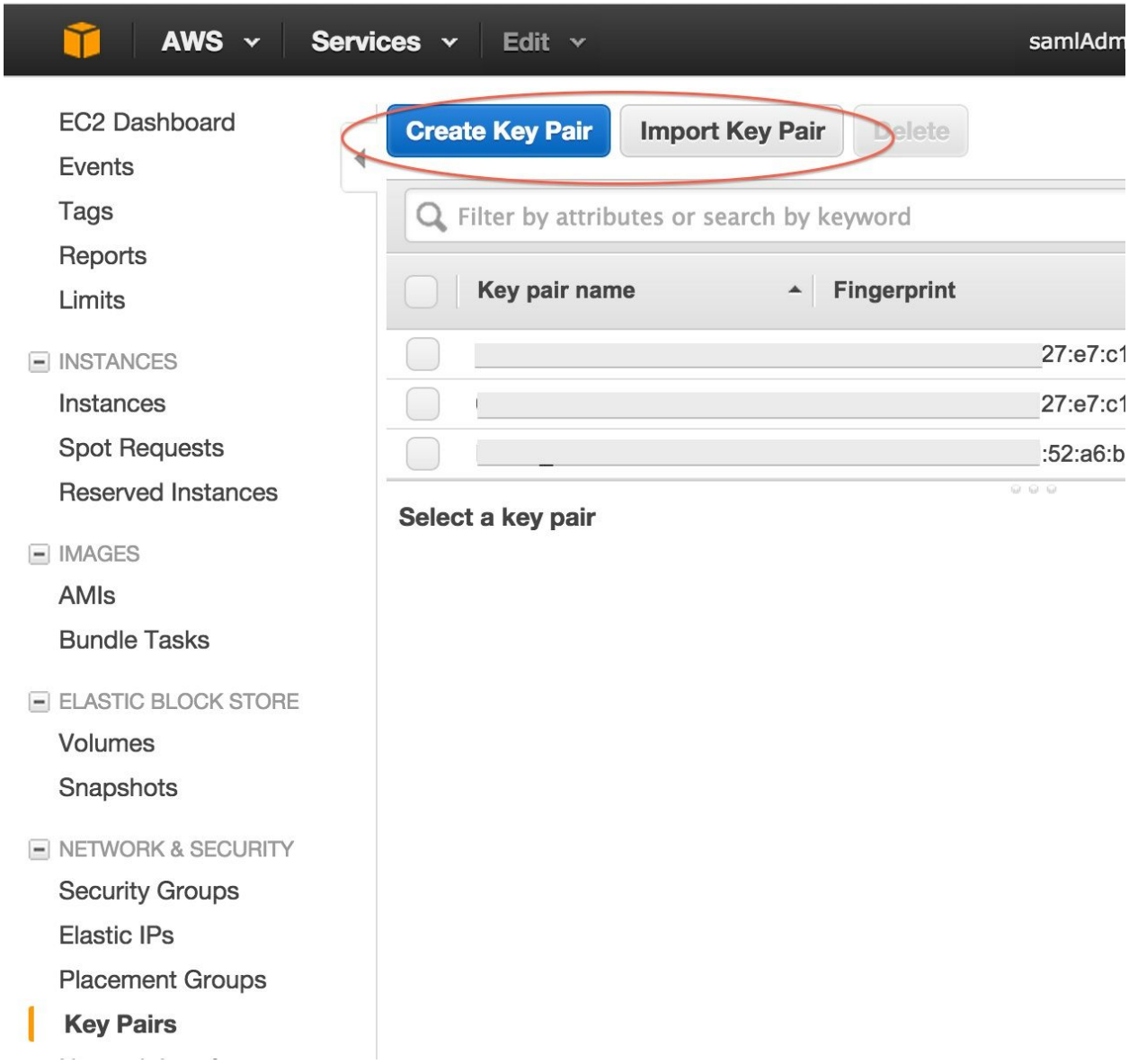



Figure 5 - Showing the obtaining the AWS SSH keys from AWS Console

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. from within a container

```
(venv)root@db881107b6f4:/aws-deployments# mkdir ~/.ssh/
(venv)root@db881107b6f4:/aws-deployments# vi ~/.ssh/My-AWS-SSH-Private-Key.pem
(venv)root@db881107b6f4:/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

## 6) Setting the AWS Credentials for boto and AWS CLI

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 seperate 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 6 - Contents of the ~/.f5aws file which must be edited.

## EULA Acceptance

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
***Pool Member/Docker Host:*** Amazon ECS-Optimized Amazon Linux AMI:
https://aws.amazon.com/marketplace/pp/B00U6QTYI2
***Client:*** Ubuntu 14.04 LTS
https://aws.amazon.com/marketplace/pp/B00JV9JBDS

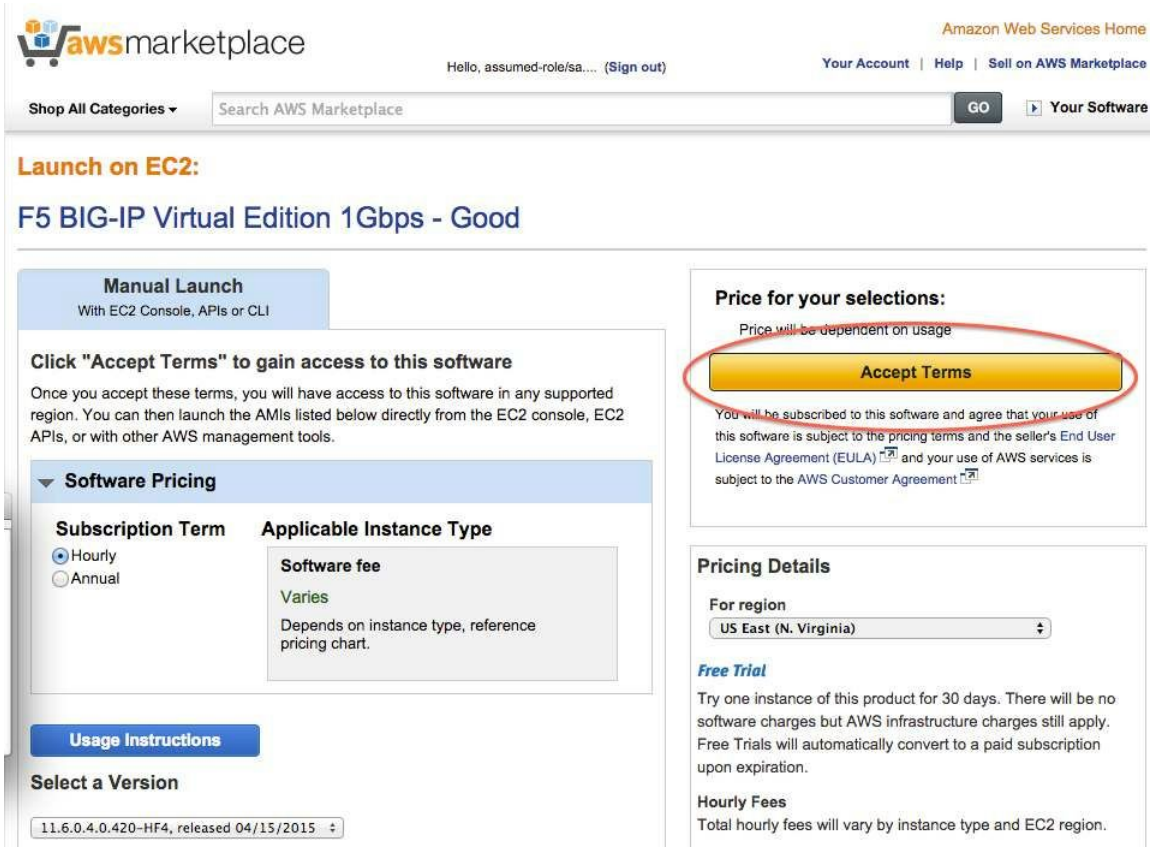



Figure 7 - EULA Acceptance for the 1Gbps Good AMI.

TECH-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…}'
Initialize the ansible inventory for a new deployment. 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"]}'`

*NOTE: Because we leverage Amazon's ECS-Optimized AMIs for the Docker hosts, only certain regions may have them available. We've tested with us-east-1, us-west-2, & eu-west-1. See roles/infra/files/apphost.json for which ami-ids are currently populated in the CFT & AWS's [https://aws.amazon.com/marketplace/pp/B00U6QTYI2](https://aws.amazon.com/marketplace/pp/B00U6QTYI2) for more information.*

### 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 commands 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 the CloudFormation stacks that have or will be deployed as part of this deployment, along with associated output variables. The 'init' subcommand prints login information for deployed hosts.

### 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 complete 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.

# 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)

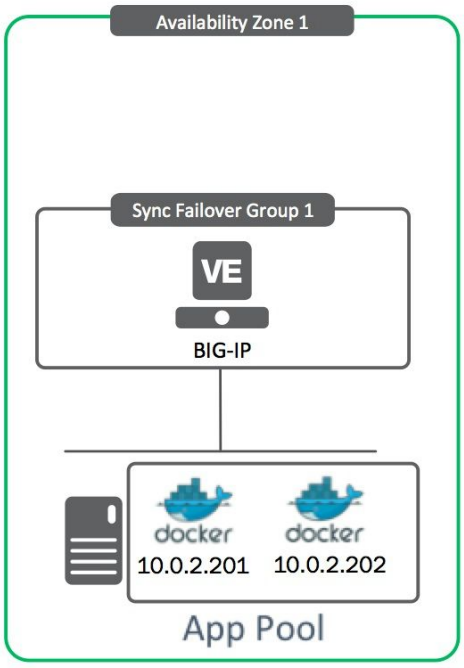



Figure 8 - 'single-standalone' deployment model

## 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)

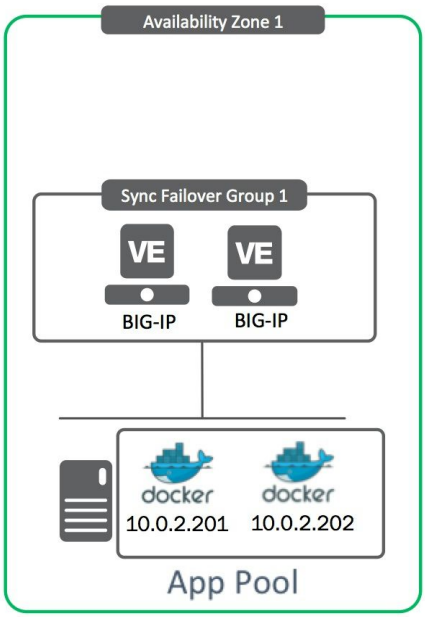



Figure 9 - 'single-cluster' deployment model

### 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)

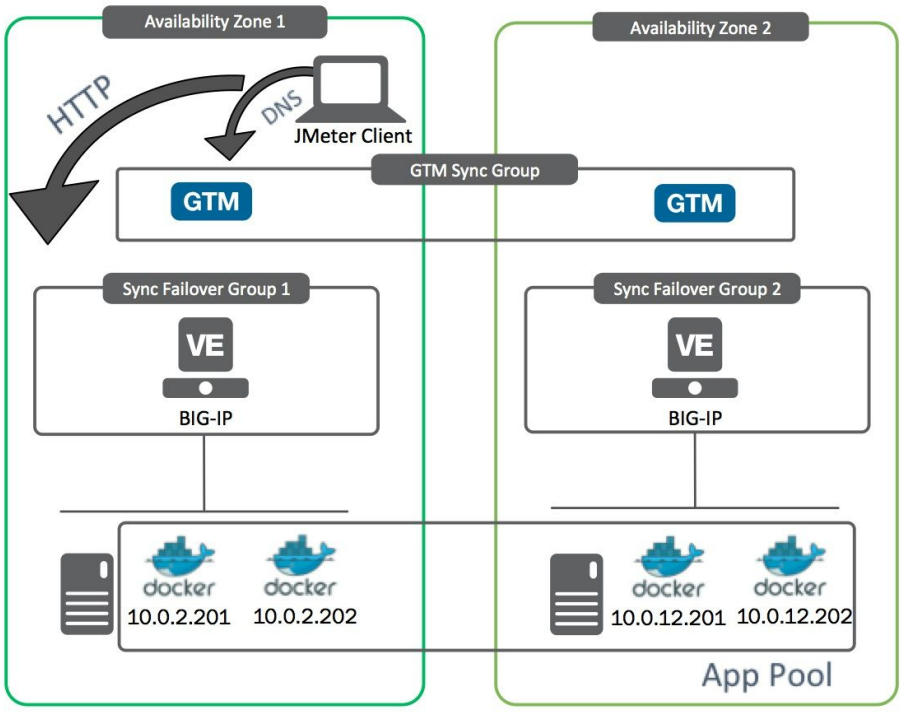



Figure 10 - 'standalone-per-zone' deployment model

### 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 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)

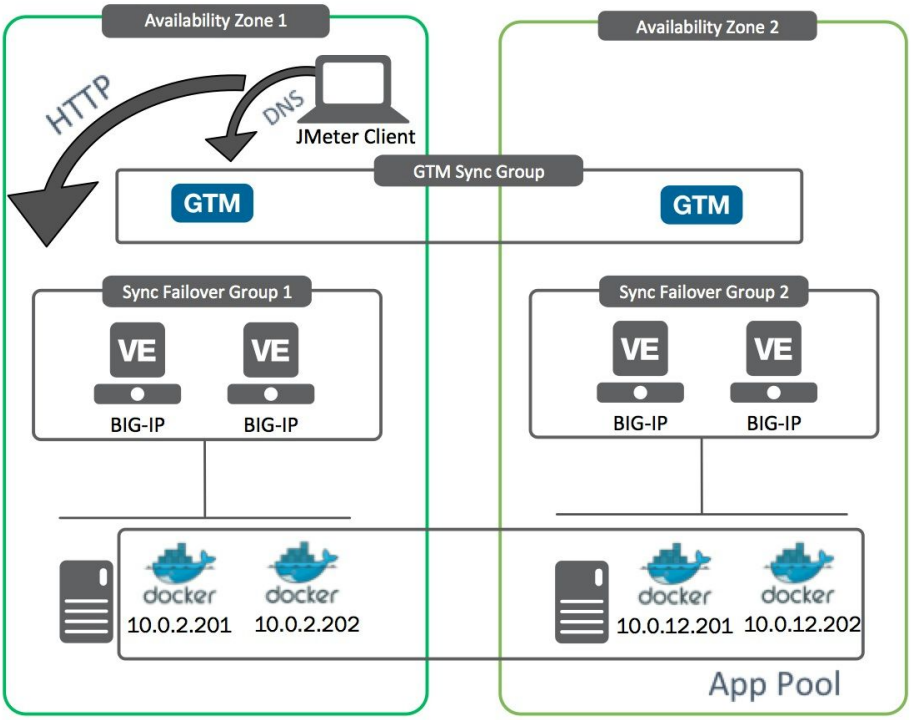



Figure 11 - 'cluster-per-zone' deployment model

## 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.

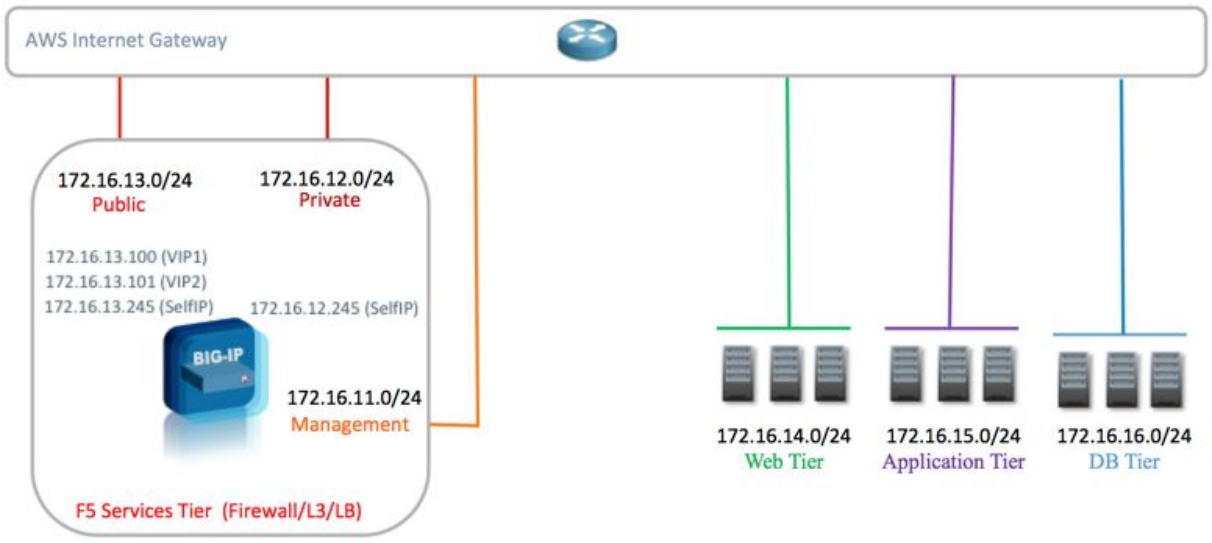



Figure 12 - Multi-tier routed architecture

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 customer 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 13 - 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.yaml 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.

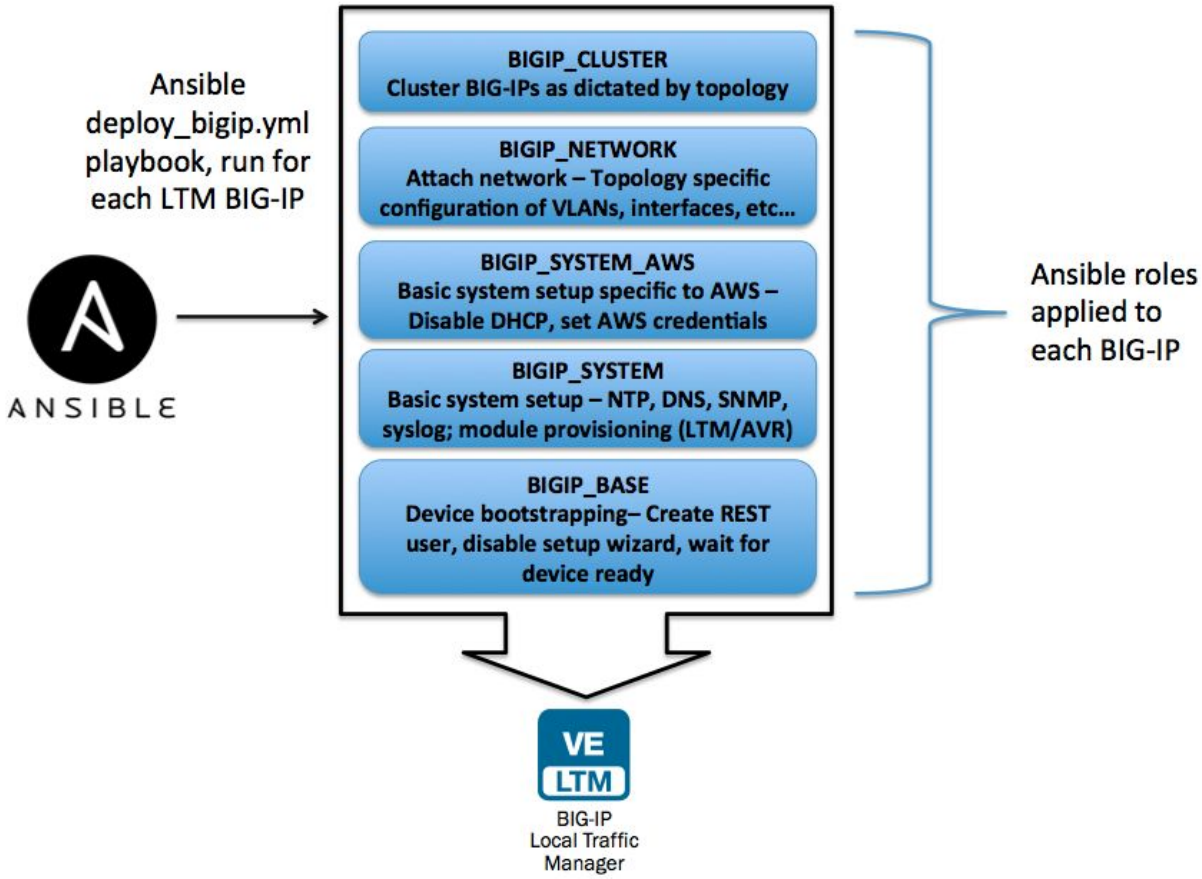



Figure 14 - Showing how the LTM configuration on BIG-IP is deployed using Ansible roles

## 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.

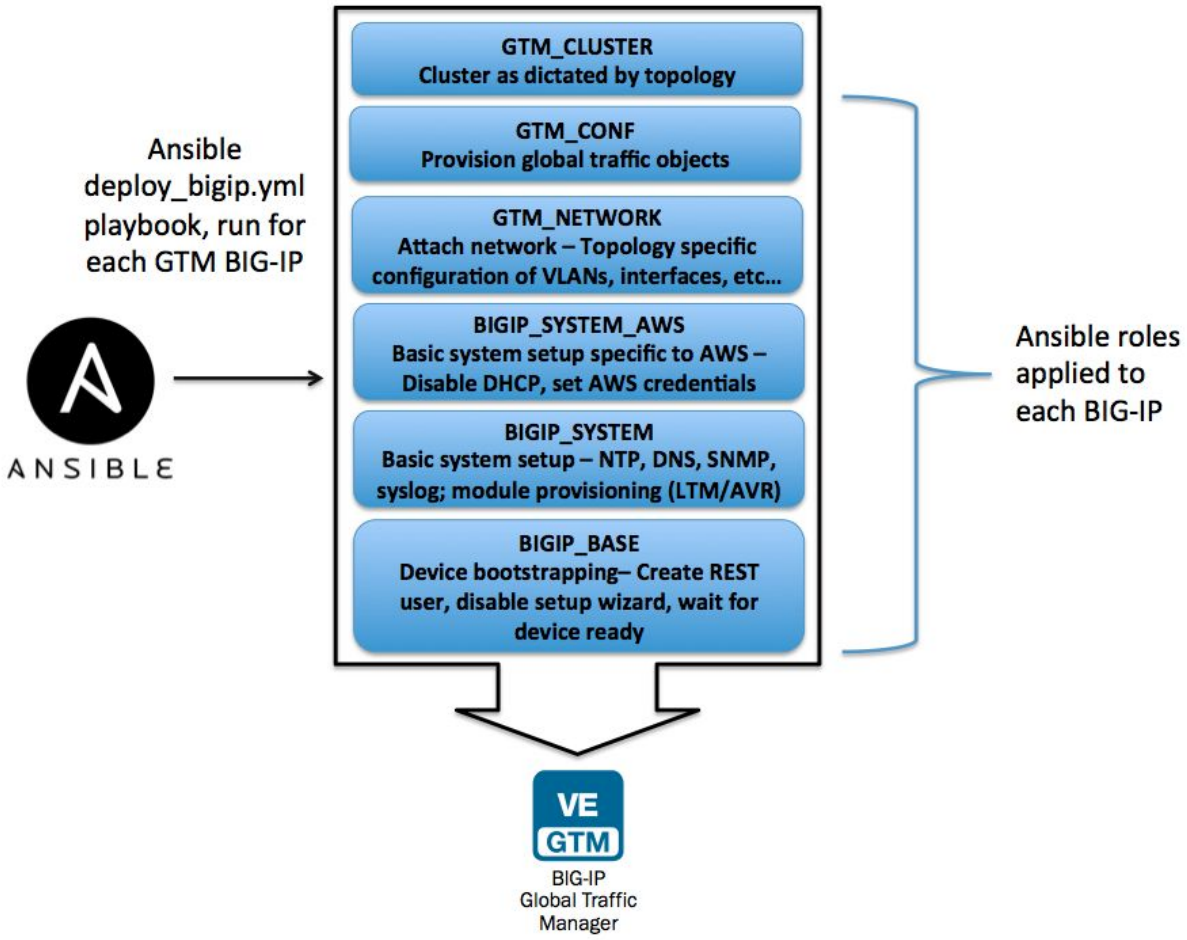



Figure 15 - Showing how the GTM configuration o3n BIG-IP is deployed using Ansible roles

## Virtual Servers

As part of all deployment topologies, we create two virtual servers:

### Virtual 1

We leverage iApps as a way to consistently deploy the TMOS traffic configuration for our first virtual server. 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.

- This built-in f5.http iApp deployed in our example has references several iRules
- A template of the iApp we deploy is available in: /roles/bigip_app/templates/demo_iApp.cfg.j2.

The playbook 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_app/files/
2) Background and sorry page images to an internal data-group.
These images can be found in ./roles/bigip_app/files/

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 VIP.

Virtual 2

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 still 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.)
```

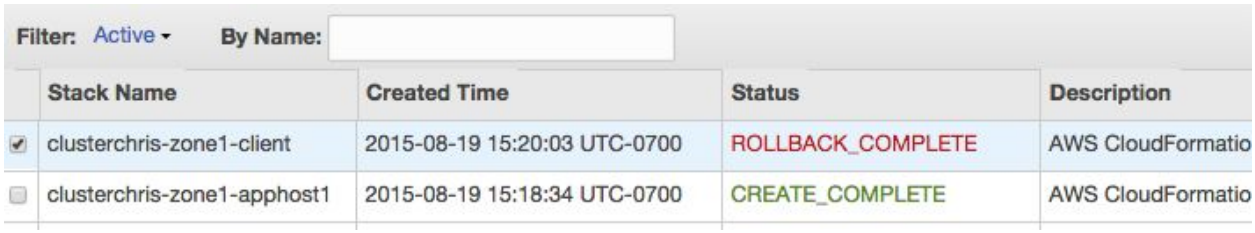
Fix:
Reset the system clock on your VM. On Ubuntu the command to do this is "sudo ntpdate ntp.ubuntu.com"

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
```

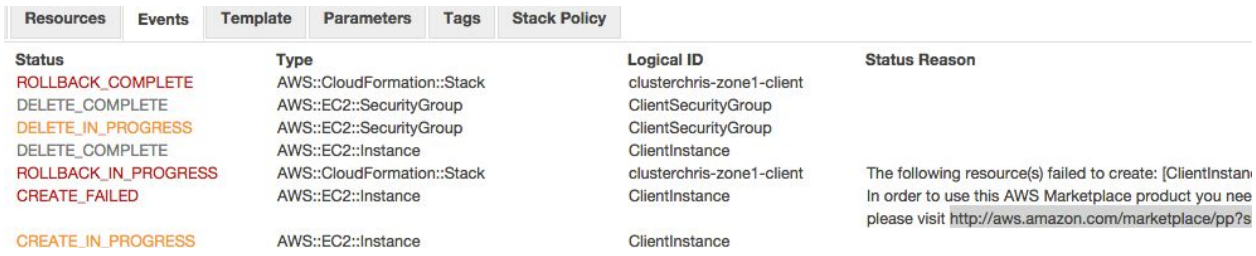
```
ClientSecurityGroup CREATE_IN_PROGRESS", "StackEvent
AWS::CloudFormation::Stack clusterchris-zone1-client
CREATE_IN_PROGRESS"], "failed": true, "output": "Problem with
CREATE. Rollback complete", "stack_outputs": {}}
```




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.



# 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 the 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 (CMDBs). Ansible itself provides several ansible specific solutions like fact_caching (implemented using redis) as well as integrations to other CMDBs 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