Table of Contents

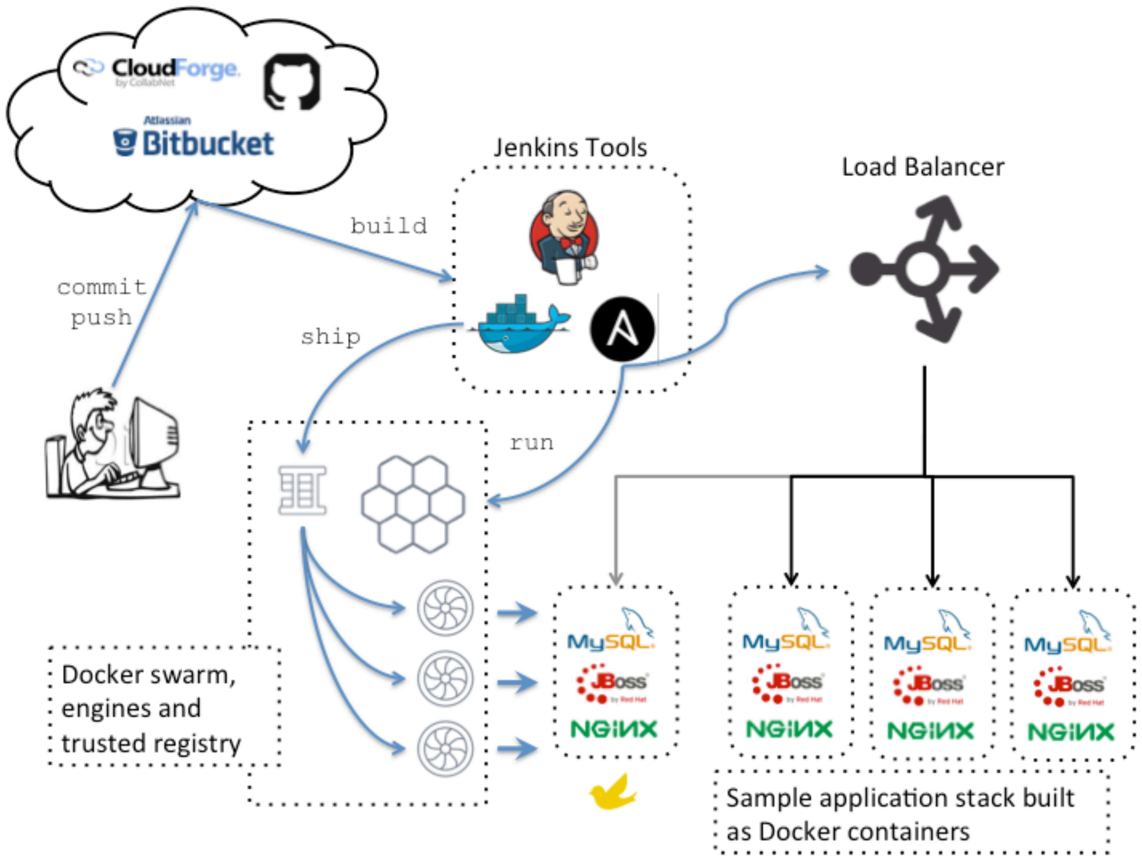
Canary Release Reference Architecture

1. Introduction

The following describes a reference architecture and implementation details to configure a canary release process using the open source tools, Docker, Ansible and Jenkins. This architecture is flexible enough to be deployed on a laptop, a private cloud or public cloud infrastructure.

The configuration should be portable across hypervisors and cloud providers, and the main infrastructure requirement is DNS where TLS certificates are being used.

The high level architecture and process flow are illustrated below.



Jenkins Tools
Load Balancer
build
commit push
ship
run
Docker swarm, engines and trusted registry
Sample application stack built as Docker containers

The Jenkins job pulls down a copy of the code based on the last commit to a VCS. The source code repository has a Dockerfile included to build the target container for the project runtime. For complex container runtimes, a Docker Compose file can be used instead. The Jenkins job builds a Docker image that includes the artifacts from the code build stage and checks the image into a Docker image registry.

All going well with the build process for code and Docker image, the canary workload is updated with the new image. The process includes de-registration from a load balancer pool, updating the Docker image, restarting the Docker container and re-connecting the new workload to the load balancing service.

After reviewing the performance of the canary workload, the rest of production workloads can be updated using a one line Ansible playbook command.

1.1 Resource Configuration

The minimum compute and network requirements to set up your own demonstration environment based on this reference architecture include:

Compute requirements:

- 4 x Ubuntu 14.04 instances
  - AWS: T2.Micro
  - Azure: Standard A1
  - Google Compute Platform: Small
  - vSphere: 1 Core, 1-2 GB RAM

Network Requirements:

- Internet accessible network range
- Optional: private network address range

Load Balancer Service (must be supported by the CM tool)

- AWS ELB (AWS only)
- HAProxy
- F5

1.2 DNS Service

This reference architecture requires the use of fully qualified domain names (FQDN) for communication between the services. The domain name used is **proserveau.local**. There must a DNS service available to host this domain, which could be a preexisting DNS service that you control, or alternatively you can set up a DNS service for purpose. This document will include instructions to configure BIND to host the domain zone for **proserveau.local**.

1.3 CVS Service

This reference architecture requires the of a code versioning system (CVS). **Github** has been selected for this purpose. **Github** is a publically accessible service provided over the Internet. In order to use **Github** you will need to register an account with **Github** and load source code into a **Github** repository. This document does not cover this procedure.

1.4 Continuous Integration Service

This reference architecture requires the use of Continuous Integration system to retrieve source code from the CVS system, and build it into an executable module. **Jenkins** has been selected for this purpose. This document will provide instructions for installing and configuring **Jenkins** for this purpose.

1.5 Configuration Management Service

This reference architecture requires the use of a configuration management tool (CM). **Ansible** has been selected for this purpose. This document will provide instructions for installing and configuring **Ansible** for this purpose.

1.6 Application Containerisation Service

This reference architecture requires the use of an application containerisation service to host the compile applications. **Docker** has been selected for this purpose. This document will provide instructions for installing and configuring the **Docker Engine**, **Docker Swarm**, **Docker Registry**, **Etcd**, and **Registrator** services.

1.7 Load Balancer Service

This reference architecture requires the use of a load balancer service to ensure that the applications is always available during updates. Use the most appropriate load balancing technology available, as long as it is supported by the CM tool. This document will provide instructions for installing and configuring **HAProxy**, and will provide general instructions for using **AWS ELB**.

2. Instance Configuration

The first step is to provision the four Ubuntu 14.04 instances that will be used by the reference architecture. The procedure for doing this will be unique to each cloud provider and is not covered here. Once the instances are up and running the following post configuration tasks will need to occur.

2.1 Instance Hostnames

Throughout the document the four instances required will be referred to with the following FQDNs.

1. cfgmgr.proserveau.local
2. docker0.proserveau.local
3. docker1.proserveau.local
4. docker2.proserveau.local

2.2 IP Addresses

All instances should have static private IP addresses. The cfgmgr.proserveau.local instance should have a static public IP address. The

2.3 Firewalls/Security Groups

Apply the following firewall rules to the instances using the appropriate method for your cloud provider, substituting the correct IP addresses and subnet ranges for the environment you have deployed.

Source	**Source Port**	**Destination**	**Destination Port**	**Purpose**
0.0.0.0/0	Any	cfgmgr.proserveau.local docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	22/TCP	SSH
cfgmgr. proserveau.local docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	Any	cfgmgr.proserveau.local docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	2376/TCP	Docker Engine TLS
docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	Any	cfgmgr.proserveau.local	4001/TCP	Etcd
docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	Any	cfgmgr.proserveau.local	5000/TCP	Docker Registry
cfgmgr.proserveau.local	Any	docker1.proserveau.local docker2.proserveau.local	7000/TCP	Docker Swarm

Source	Source Port	Destination	Destination Port	Purpose
0.0.0.0/0	Any	cfgmgr.proserveau.local	8443/TCP	Jenkins HTTPS
docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	Any	cfgmgr.proserveau.local	53/UDP	DNS
0.0.0.0	Any	docker0.proserveau.local docker1.proserveau.local docker2.proserveau.local	3000/TCP	Application LB End Points
0.0.0.0	Any	cfgmgr.proserveau.local	80/TCP	Application LB Front End

2.4 Set Instance Hostnames

Edit the **/etc/hostname** and **/etc/hosts** files to set the FQDN of each instance and make sure that each host can resolve its own name to its own static IP.

2.5 Upload Software

Using SCP upload the script bundle <details here> to each of the instances. OR Git pull repository?

2.6 Update Instances

Run the script **install-ubuntu-updates.sh** to update all packages on each instance.

```
# sudo ./install-ubuntu-updates.sh
```

2.7 DNS Service Option 1: Install BIND DNS Service

Run the script **install-dns.sh** to install and configure the BIND server on cfgmgr server only, specify the IP addresses of the Jenkins server, the upstream DNS forwarder (usually provided by the cloud provider), and the each of the Docker engine hosts.

```
# sudo ./install-dns.sh --dns <cfgmgr IP> --fowarder <DNS
forwarder IP> --docker0 <Docker0 IP> --docker1 <Docker1 IP> --
docker2 <Docker2 IP>
```

2.8 DNS Service Option 2: Configure External DNS Service

If you choose to use an external DNS service create A records in the appropriate domain zone for the host names that you have selected (cfgmgr, docker0, docker1, docker2), and create a CNAME record for the name “swarm” pointing to docker1.

2.9 DNS Service Option 3: Configure /etc/hosts files

A third option is to configure the /etc/hosts files of all four instances to include IP addresses and hostnames of all instances. Insert records in the /etc/hosts file as shown below replacing the x.x.x.x with the appropriate IP addresses.

```
    x.x.x.x        cfgmgr.proserveau.local  cfgmgr
    x.x.x.x        docker0.proserveau.local docker0
    x.x.x.x        docker1.proserveau.local docker1
    x.x.x.x        docker2.proserveau.local docker2
    x.x.x.x        swarm.proserveau.local   swarm
```

2.10 Configure Client DNS

Run the script **client-dns.sh** on each instance to configure the DNS client settings to use the selected DNS service.

```
# sudo ./client-dns.sh --dns <DNS Server IP> --domain <DNS
DOMAIN NAME>
```

2.11 Setup Passwordless SSH Login

The **cfgmgr** instance needs to be able to logon to each Docker instance using SSH and a certificate, but without any password.

Create and public/private keypair for use with SSH on each of the docker0, docker1 and docker2 instances. Use the same keypair for each server. If you are using AWS or Azure you will already have a keypair available to you. If you do not have a keypair available you will need to generate one for this purpose.

Copy the private key file from the keypair to the **~/.ssh** directory on the **cfgmgr** instance. Change the name of the private key file to devops.key, and use the command

```
# chmod 600 devops.key
```

to set the file permissions correctly.

From the **cfgmgr** instance perform a login to each of the Docker instances using the following commands:

```
# ssh -i ~/.ssh/devops.key ubuntu@docker0.proserveau.local
# ssh -i ~/.ssh/devops.key ubuntu@docker1.proserveau.local
# ssh -i ~/.ssh/devops.key ubuntu@docker2.proserveau.local
```

At each login you will be prompted to add each Docker host to the list of known hosts, once that is complete exit from each host.

2.12 Cloud Provider DNS Settings

Some Cloud Providers (Azure, AWS) may automatically change the contents of **/etc/resolv.conf** back to their defaults. This functionality can be disabled from the Cloud Provider management interface.

2.13 Generate Certificates

Run the script **create-tls.sh** on cfgmgr to create the certificates required to enable TLS throughout the system. This script will generate the certificates, and copy them from cfgmgr to the Docker hosts.

```
# sudo ./create-tls.sh
```

2.14 Install Docker Engine

Run the script **install-cfgmgr-docker.sh** on **cfgmgr** to install and configure the Docker Engine, Docker Registry, and Etcd.

```
# sudo ./install-cfgmgr.docker.sh
```

Run the script **install-docker.sh** on each of the Docker instances to install Docker Engine and Registrator service.

```
# sudo ./install-docker.sh
```

2.15 Install Docker Swarm

Run the script

Install Ansible

Run the script <script name> to install and configure Ansible on cfgmgr.
Copy Ansible patches.

Install Jenkins

Run the script <script name> to install Jenkins on cfgmgr.
Configure Jenkins

Configure Load Balancing

Optional: Install HAProxy

Run the script <script name> to install and configure HAProxy on cfgmgr.

Options: AWS ELB

In AWS Management create an ELB listening on port 80, assign the docker0, docker1, and docker2 instances to the ELB redirecting to port 3000.