Docker Deployment System Built On

Amazon Web Services

Chris Fortier

CSCI E-90 Cloud Computing

Harvard University Extension School

December 2014

Project Description:

Docker and its associated ecosystem is one of the most hyped about technologies in the computing world. However many of the examples only show simple examples that may not be of practical use. In this project we will build a Virtual Private Cloud suitable for serving a web application using CloudFormation. We will then explore an innovative technique of running Docker containers on each server instance behind an HA Proxy process.

There are several important benefits to this approach. By running instances in an auto-scaling group we have elastic scalability to meet any amount of demand. By using HA Proxy on each host server, we will be able to deploy a new version of our application container to new visitors while simultaneously allowing all current connections to finish for current users. This affords the ability to do a rolling deployment without any visitor interruption.

YouTube Links:

Full Presentation:

http://www.youtube.com/watch?v=M3o5x8tb5eY&feature=youtube_gdata

Two-minute summary:

http://www.youtube.com/watch?v=f5bXkZycxqA&feature=youtube_gdata

Table of Contents

Objectives:

This goal of this project is to demonstrate how to build a full-scale infrastructure on Amazon Web Services (AWS) that would be suitable for running a production web application. As the primary focus is to discuss Docker, I will make assumptions that the reader is already well versed in all of the underlying AWS technologies. Specific technologies include: Virtual Private Cloud (VPC), Auto-Scaling Groups, Elastic Load Balancers (ELB), Route53, Elastic Compute Cloud (EC2), and CloudFormation.

All source code will be made available on github.com and we will only use open source tools as needed.

Virtual Private Cloud

We need to have to infrastructure in place to begin our demonstration. For maximum scalability and fault tolerance we will build a VPC with the following requirements:

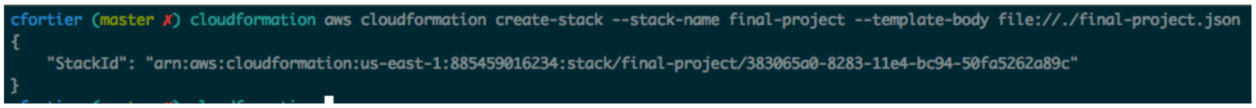
- (3) Availability Zones
- Auto-scaling group of "web-tier" instances spanning all three availability zones
- Auto-scaling group Consul server instances spanning all three availability zones
- Elastic Load Balancer to balance traffic to the auto scale group
- Security Groups allowing access for:
  - HTTP traffic from all addresses to the ELB
  - All HTTP traffic from the ELB to the auto scale instances
  - Traffic between all instances for our distributed key/value store

CloudFormation with Troposphere

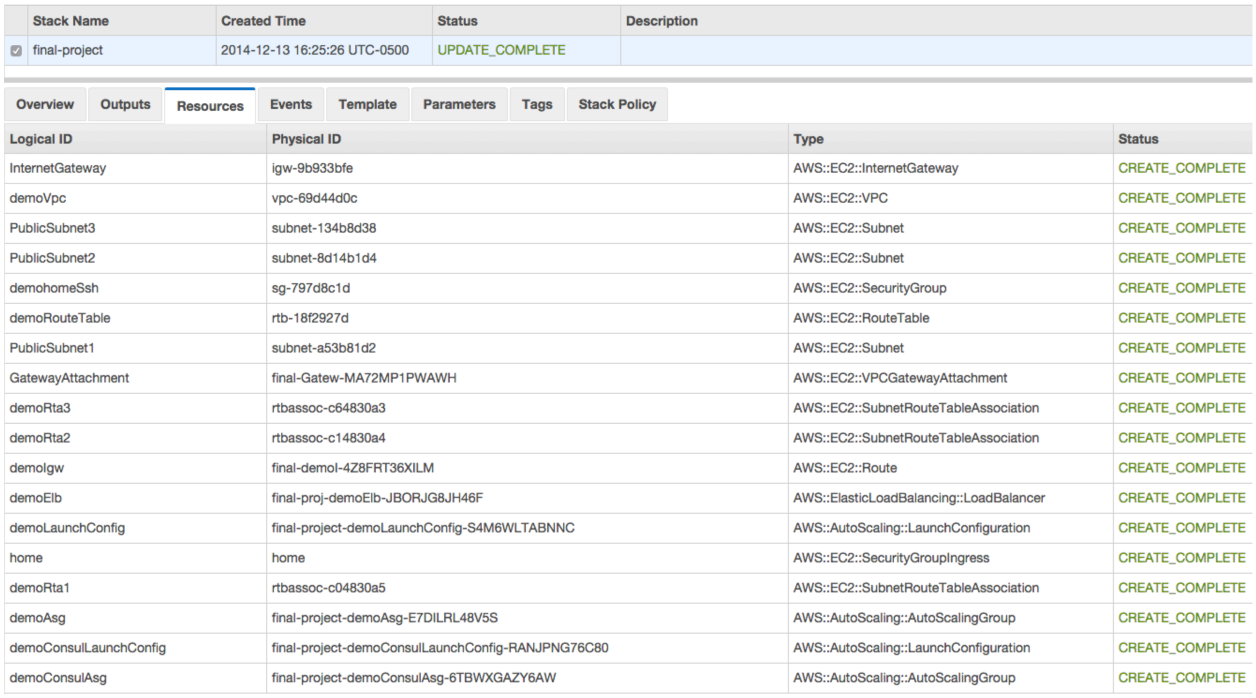
To build this VPC we will use an open source tool called *Troposphere* (https://github.com/cloudtools/troposphere), which is a Python library to automate the creation of CloudFormation templates. The source code used can be found at (https://github.com/cfortier2/cs90-final/source_code/cloudformation).

The Troposphere file is written to generate a CloudFormation template file named `final-project.json` in the same directory. Once the CloudFormation template is generated, the stack can be launched using the AWS command line tools.

Example:



```
cfortier (master ✗) cloudformation aws cloudformation create-stack --stack-name final-project --template-body file://./final-project.json
{
    "StackId": "arn:aws:cloudformation:us-east-1:885459016234:stack/final-project/383065a0-8283-11e4-bc94-50fa5262a89c"
}
```



	Stack Name	Created Time	Status	Description
✓	final-project	2014-12-13 16:25:26 UTC-0500	UPDATE_COMPLETE	

Logical ID	Physical ID	Type	Status
InternetGateway	igw-9b933bfe	AWS::EC2::InternetGateway	CREATE_COMPLETE
demoVpc	vpc-69d44d0c	AWS::EC2::VPC	CREATE_COMPLETE
PublicSubnet3	subnet-134b8d38	AWS::EC2::Subnet	CREATE_COMPLETE
PublicSubnet2	subnet-8d14b1d4	AWS::EC2::Subnet	CREATE_COMPLETE
demohomeSsh	sg-797d8c1d	AWS::EC2::SecurityGroup	CREATE_COMPLETE
demoRouteTable	rtb-18f2927d	AWS::EC2::RouteTable	CREATE_COMPLETE
PublicSubnet1	subnet-a53b81d2	AWS::EC2::Subnet	CREATE_COMPLETE
GatewayAttachment	final-Gatew-MA72MP1PWAWH	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE
demoRta3	rtbassoc-c64830a3	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
demoRta2	rtbassoc-c14830a4	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
demoIgw	final-demoI-4Z8FRT36XILM	AWS::EC2::Route	CREATE_COMPLETE
demoElb	final-proj-demoElb-JBORJG8JH46F	AWS::ElasticLoadBalancing::LoadBalancer	CREATE_COMPLETE
demoLaunchConfig	final-project-demoLaunchConfig-S4M6WLTABNNC	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE
home	home	AWS::EC2::SecurityGroupIngress	CREATE_COMPLETE
demoRta1	rtbassoc-c04830a5	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
demoAsg	final-project-demoAsg-E7DILRL48V5S	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE
demoConsulLaunchConfig	final-project-demoConsulLaunchConfig-RANJPNG76C80	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE
demoConsulAsg	final-project-demoConsulAsg-6TBWXGAZY6AW	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE

Server Installation

We are using base Ubuntu:14.04 images. We will need to install several programs for this to work correctly. The `install_all.sh` can be run on all three Consul Servers and Demo Host instances to install everything. These scripts are located in the `source_code/scripts/setup` directory.

Consul

In a distributed system one of the challenges is to make sure each node has the correct configuration values to operate. One option is to have a central command server that you can use to `ssh` in to each server and write configuration files when they change. This may work fine for a small number of fixed servers but isn't particularly efficient for a large number of servers, especially when they reside in an auto scale group and are rather ephemeral. For this example we will use a distributed key/value store called Consul (https://consul.io).

As described on its own site, "Consul has multiple components, but as a whole, it is a tool for discovering and configuring services in your infrastructure. It provides several key features: Service Discovery, Health Checking, Key/Value Store, Multi Datacenter"[i] In the scope of this demonstration we will only be concerned with the key/value store, however the service discovery and health checking are vital components for any true production system.

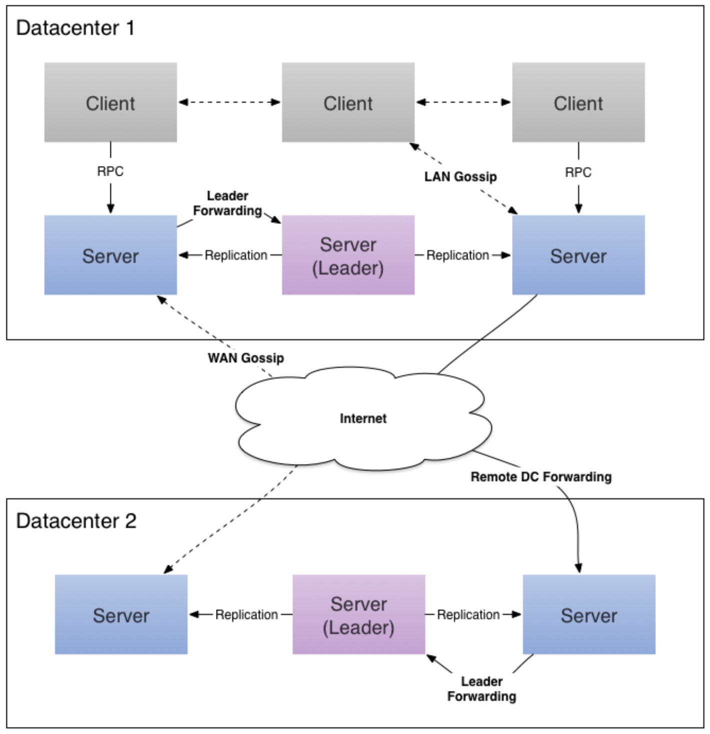
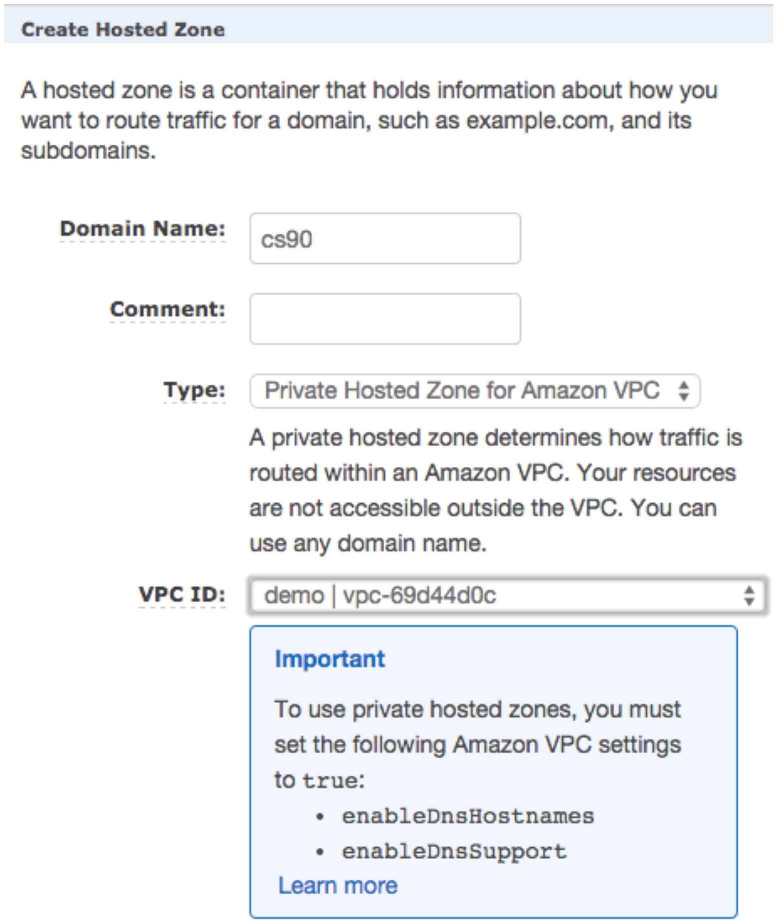


Figure 1Consul Architecture[ii]

Above is a diagram showing the Consul architecture for a multi datacenter deployment. For this demonstration we will have a cluster of three Consul Servers that are separate from the three client servers. Since we are bootstrapping a Consul server cluster we need a convention to locate all of them. The convention we will use is to create a Route53 Private DNS zone named `.cs90` then create a `CNAME` record based on the availability zone that the server resides in. Specifically, we will create: `consul.us-east-1a.cs90, consul.us-east-1b.cs90, consul.us-east-1d.cs90`. Since private DNS hosted zones are a new feature we will have to create the zone manually. The record creation is handled by: (https://github.com/cfortier2/cs90-final/blob/master/source_code/scripts/route53.py)


Create Hosted Zone

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com, and its subdomains.

Domain Name: cs90

Comment:

Type: Private Hosted Zone for Amazon VPC

A private hosted zone determines how traffic is routed within an Amazon VPC. Your resources are not accessible outside the VPC. You can use any domain name.

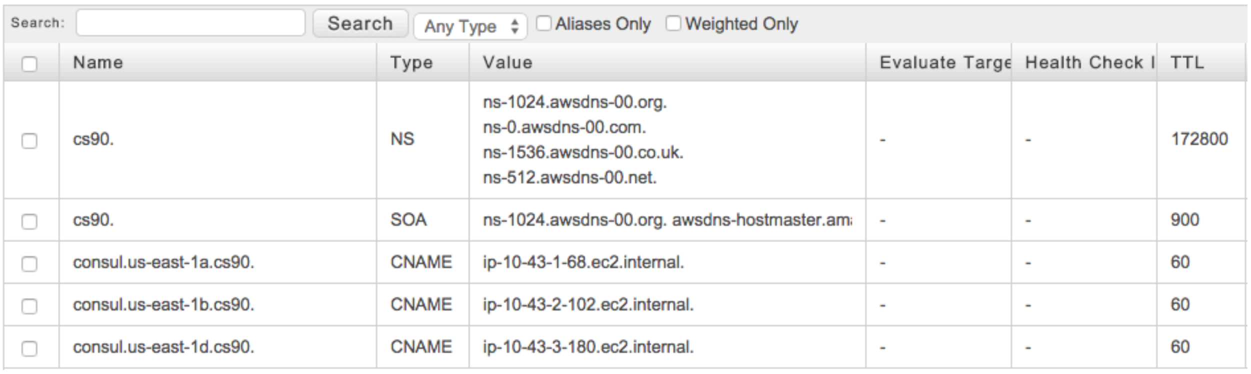
VPC ID: demo | vpc-69d44d0c

Important

To use private hosted zones, you must set the following Amazon VPC settings to `true`:

- `enableDnsHostnames`
- `enableDnsSupport`

Learn more



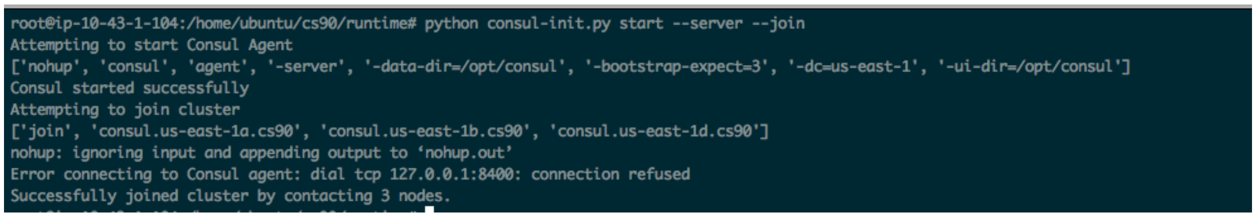
Search: [ ] Search | Any Type | Aliases Only | Weighted Only

	Name	Type	Value	Evaluate Targe	Health Check I	TTL
☐	cs90.	NS	ns-1024.awsdns-00.org. ns-0.awsdns-00.com. ns-1536.awsdns-00.co.uk. ns-512.awsdns-00.net.	-	-	172800
☐	cs90.	SOA	ns-1024.awsdns-00.org. awsdns-hostmaster.am	-	-	900
☐	consul.us-east-1a.cs90.	CNAME	ip-10-43-1-68.ec2.internal.	-	-	60
☐	consul.us-east-1b.cs90.	CNAME	ip-10-43-2-102.ec2.internal.	-	-	60
☐	consul.us-east-1d.cs90.	CNAME	ip-10-43-3-180.ec2.internal.	-	-	60

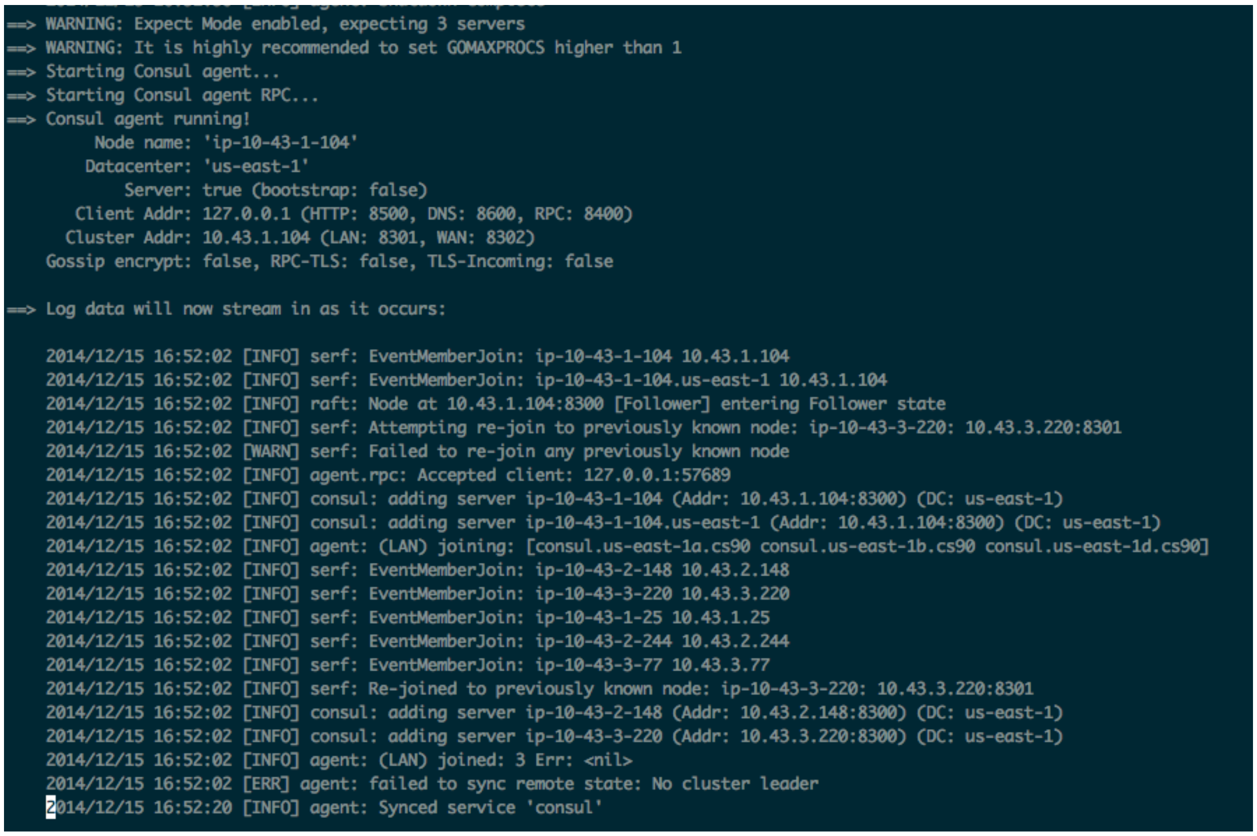
To install Consul I created a simple bash script, which can be found at: (https://github.com/cfortier2/cs90-final/blob/master/source_code/scripts/install_consul.sh). I also created a Consul init script which can is located at: (https://github.com/cfortier2/cs90-final/blob/master/source_code/scripts/consul-init.py). This init script is currently written to be run manually but should be configured to launch as an init.d or systemd type script.

We need to initialize the three servers and also the three hosts. The same script can be run for both types of servers, but with different values. Starting the servers is accomplished by running ***`python consul-init.py start --server --join`.*** This will start consul in server mode and attempt to join the cluster.

Example output from one of the servers produces:

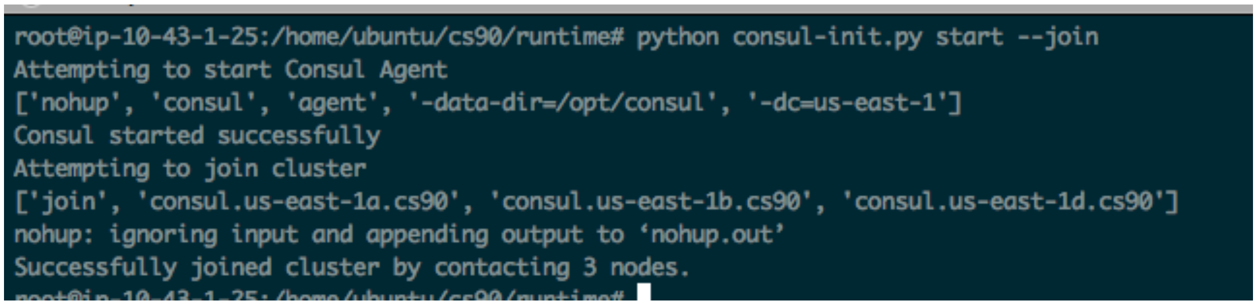


```
root@ip-10-43-1-104:/home/ubuntu/cs90/runtime# python consul-init.py start --server --join
Attempting to start Consul Agent
['nohup', 'consul', 'agent', '-server', '-data-dir=/opt/consul', '-bootstrap-expect=3', '-dc=us-east-1', '-ui-dir=/opt/consul']
Consul started successfully
Attempting to join cluster
['join', 'consul.us-east-1a.cs90', 'consul.us-east-1b.cs90', 'consul.us-east-1d.cs90']
nohup: ignoring input and appending output to 'nohup.out'
Error connecting to Consul agent: dial tcp 127.0.0.1:8400: connection refused
Successfully joined cluster by contacting 3 nodes.
```

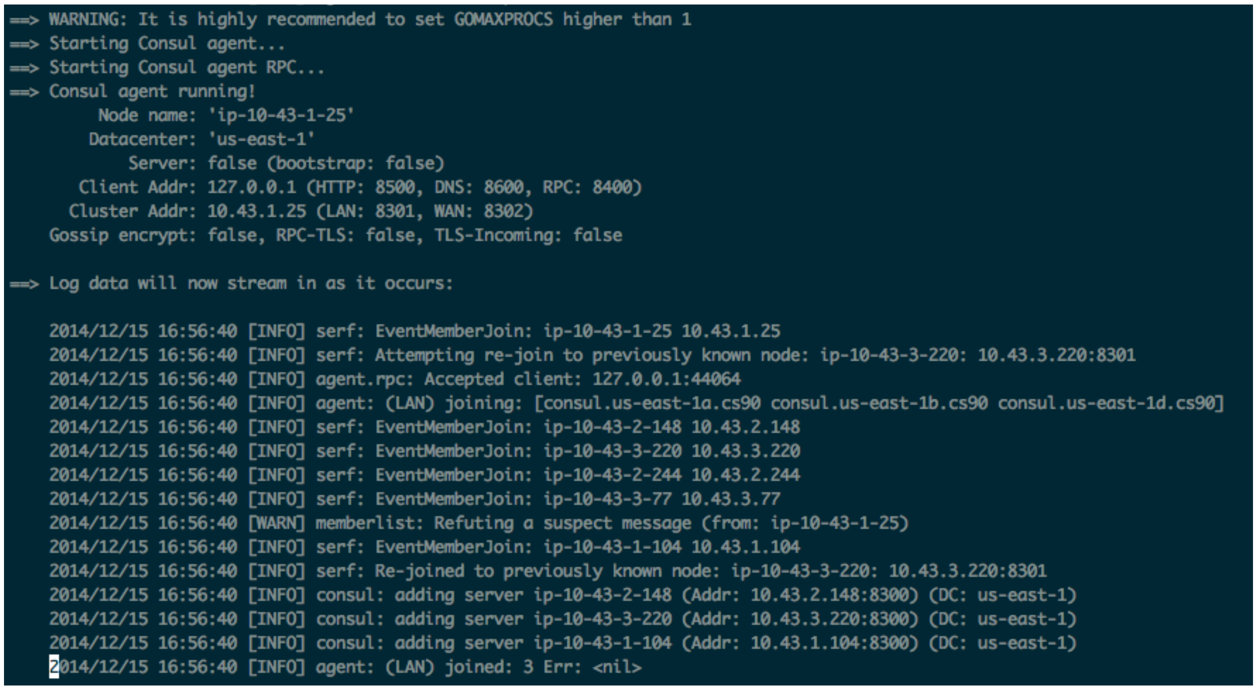


```
==> WARNING: Expect Mode enabled, expecting 3 servers
==> WARNING: It is highly recommended to set GOMAXPROCS higher than 1
==> Starting Consul agent...
==> Starting Consul agent RPC...
==> Consul agent running!
         Node name: 'ip-10-43-1-104'
        Datacenter: 'us-east-1'
            Server: true (bootstrap: false)
       Client Addr: 127.0.0.1 (HTTP: 8500, DNS: 8600, RPC: 8400)
      Cluster Addr: 10.43.1.104 (LAN: 8301, WAN: 8302)
    Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false

==> Log data will now stream in as it occurs:

    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-1-104 10.43.1.104
    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-1-104.us-east-1 10.43.1.104
    2014/12/15 16:52:02 [INFO] raft: Node at 10.43.1.104:8300 [Follower] entering Follower state
    2014/12/15 16:52:02 [INFO] serf: Attempting re-join to previously known node: ip-10-43-3-220: 10.43.3.220:8301
    2014/12/15 16:52:02 [WARN] serf: Failed to re-join any previously known node
    2014/12/15 16:52:02 [INFO] agent.rpc: Accepted client: 127.0.0.1:57689
    2014/12/15 16:52:02 [INFO] consul: adding server ip-10-43-1-104 (Addr: 10.43.1.104:8300) (DC: us-east-1)
    2014/12/15 16:52:02 [INFO] consul: adding server ip-10-43-1-104.us-east-1 (Addr: 10.43.1.104:8300) (DC: us-east-1)
    2014/12/15 16:52:02 [INFO] agent: (LAN) joining: [consul.us-east-1a.cs90 consul.us-east-1b.cs90 consul.us-east-1d.cs90]
    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-2-148 10.43.2.148
    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-3-220 10.43.3.220
    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-1-25 10.43.1.25
    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-2-244 10.43.2.244
    2014/12/15 16:52:02 [INFO] serf: EventMemberJoin: ip-10-43-3-77 10.43.3.77
    2014/12/15 16:52:02 [INFO] serf: Re-joined to previously known node: ip-10-43-3-220: 10.43.3.220:8301
    2014/12/15 16:52:02 [INFO] consul: adding server ip-10-43-2-148 (Addr: 10.43.2.148:8300) (DC: us-east-1)
    2014/12/15 16:52:02 [INFO] consul: adding server ip-10-43-3-220 (Addr: 10.43.3.220:8300) (DC: us-east-1)
    2014/12/15 16:52:02 [INFO] agent: (LAN) joined: 3 Err: <nil>
    2014/12/15 16:52:02 [ERR] agent: failed to sync remote state: No cluster leader
    2014/12/15 16:52:20 [INFO] agent: Synced service 'consul'
```

The last step to setting up the Consul cluster is to join the host nodes to the Consul cluster. This time we run the same command without the server option: ***`python consul-init.py start --server --join`***. The output should show:



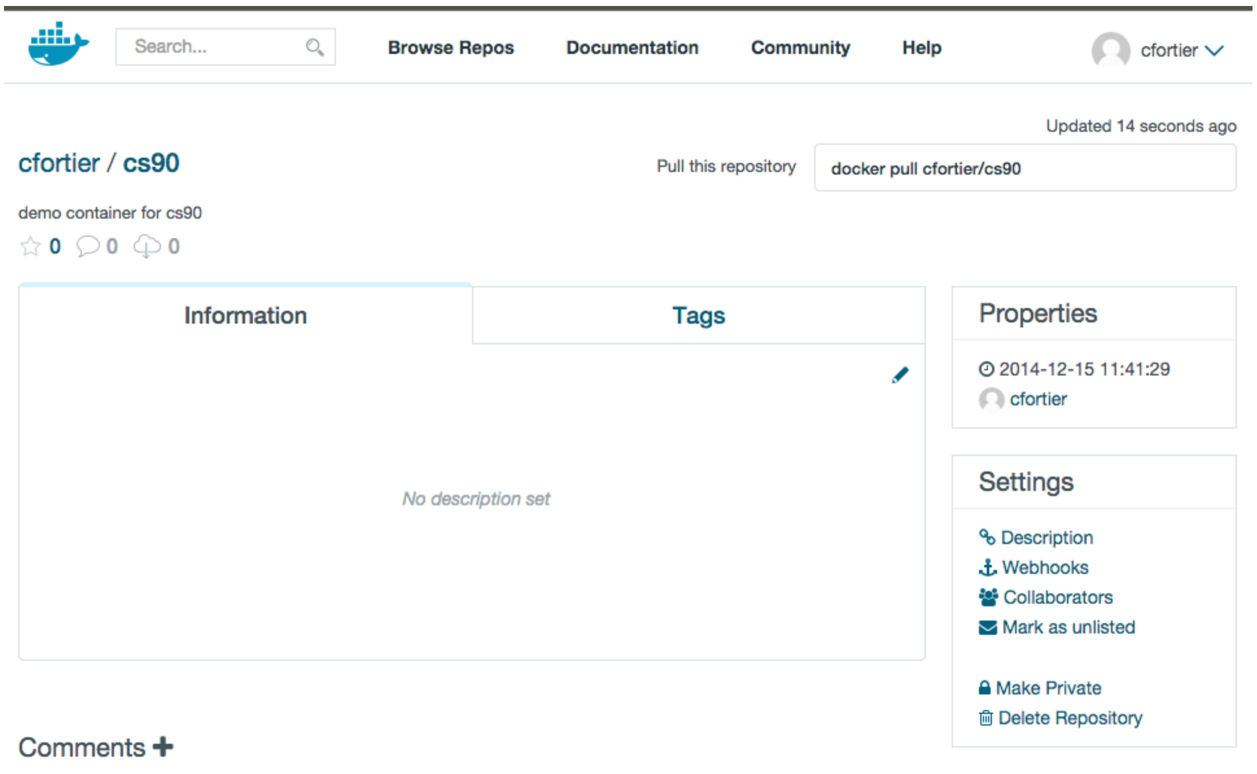
```
root@ip-10-43-1-25:/home/ubuntu/cs90/runtime# python consul-init.py start --join
Attempting to start Consul Agent
['nohup', 'consul', 'agent', '-data-dir=/opt/consul', '-dc=us-east-1']
Consul started successfully
Attempting to join cluster
['join', 'consul.us-east-1a.cs90', 'consul.us-east-1b.cs90', 'consul.us-east-1d.cs90']
nohup: ignoring input and appending output to 'nohup.out'
Successfully joined cluster by contacting 3 nodes.
```



```
==> WARNING: It is highly recommended to set GOMAXPROCS higher than 1
==> Starting Consul agent...
==> Starting Consul agent RPC...
==> Consul agent running!
         Node name: 'ip-10-43-1-25'
        Datacenter: 'us-east-1'
            Server: false (bootstrap: false)
       Client Addr: 127.0.0.1 (HTTP: 8500, DNS: 8600, RPC: 8400)
      Cluster Addr: 10.43.1.25 (LAN: 8301, WAN: 8302)
    Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false

==> Log data will now stream in as it occurs:

    2014/12/15 16:56:40 [INFO] serf: EventMemberJoin: ip-10-43-1-25 10.43.1.25
    2014/12/15 16:56:40 [INFO] serf: Attempting re-join to previously known node: ip-10-43-3-220: 10.43.3.220:8301
    2014/12/15 16:56:40 [INFO] agent.rpc: Accepted client: 127.0.0.1:44064
    2014/12/15 16:56:40 [INFO] agent: (LAN) joining: [consul.us-east-1a.cs90 consul.us-east-1b.cs90 consul.us-east-1d.cs90]
    2014/12/15 16:56:40 [INFO] serf: EventMemberJoin: ip-10-43-2-148 10.43.2.148
    2014/12/15 16:56:40 [INFO] serf: EventMemberJoin: ip-10-43-3-220 10.43.3.220
    2014/12/15 16:56:40 [INFO] serf: EventMemberJoin: ip-10-43-2-244 10.43.2.244
    2014/12/15 16:56:40 [INFO] serf: EventMemberJoin: ip-10-43-3-77 10.43.3.77
    2014/12/15 16:56:40 [WARN] memberlist: Refuting a suspect message (from: ip-10-43-1-25)
    2014/12/15 16:56:40 [INFO] serf: EventMemberJoin: ip-10-43-1-104 10.43.1.104
    2014/12/15 16:56:40 [INFO] serf: Re-joined to previously known node: ip-10-43-3-220: 10.43.3.220:8301
    2014/12/15 16:56:40 [INFO] consul: adding server ip-10-43-2-148 (Addr: 10.43.2.148:8300) (DC: us-east-1)
    2014/12/15 16:56:40 [INFO] consul: adding server ip-10-43-3-220 (Addr: 10.43.3.220:8300) (DC: us-east-1)
    2014/12/15 16:56:40 [INFO] consul: adding server ip-10-43-1-104 (Addr: 10.43.1.104:8300) (DC: us-east-1)
    2014/12/15 16:56:40 [INFO] agent: (LAN) joined: 3 Err: <nil>
```

Docker Hub[iii]

To work with Docker images we need a central location to store them. The open source solution is to use the Docker Hub, however many enterprise customers may want to consider any number of alternatives that provide private storage. Docker Hub operates very similar to github.com and uses very similar commands. For this demo we need to create a repository for our image:



Updated 14 seconds ago

cfortier / cs90

Pull this repository: docker pull cfortier/cs90

demo container for cs90

0 0 0

No description set

Properties

2014-12-15 11:41:29

cfortier

Settings

- Description
- Webhooks
- Collaborators
- Mark as unlisted
- Make Private
- Delete Repository

Comments

Dockerfile[iv]

Now that we have a registry, we can work on building a Docker image. Dockerfiles are the essential building declaration of a Docker container. In our example we are going to build a simple container running the Apache server with a PHP application. Containers are designed to be composable and build upon the open source community. We will start with an official open source image for PHP: (https://registry.hub.docker.com/_/php/). Our Dockerfile will reside at (https://github.com/cfortier2/cs90-final/source_code/docker/Dockerfile).

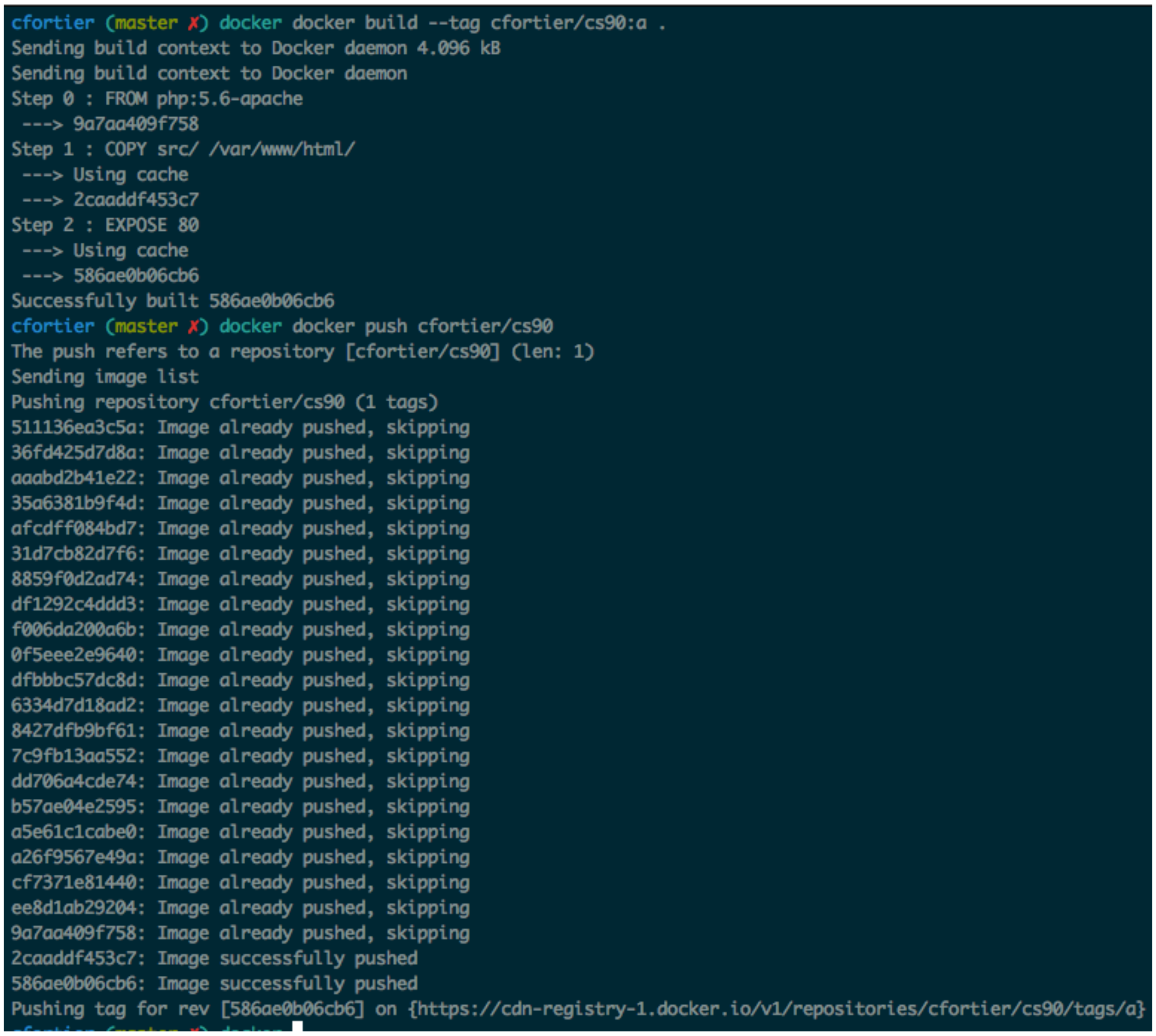
To give a quick overview of the Dockerfile: the first line states: ***`FROM php:5.6-apache`*** which is telling Docker to get the PHP image that is tagged with `5.6-apache`. The next line will copy the entire `src` folder in to the image as `/var/www/html`. The final line will expose port 80 on the container.

Boot2Docker

Since we are developing on a Mac OSX machine, we need to build the Docker container somehow. This is accomplished by running a Linux virtual machine on the Mac. Boot2docker (http://boot2docker.io/) is an open source project dedicated to solving this problem. They provide all the tools necessary to run Docker on a Mac.

Build Application

At this stage, we are finally ready to build our application. The application will consist of one PHP page that displays the host name and time. This is important to test the HA Proxy setup. The build and push output is:



```
cfortier (master ✗) docker docker build --tag cfortier/cs90:a .
Sending build context to Docker daemon 4.096 kB
Sending build context to Docker daemon
Step 0 : FROM php:5.6-apache
 ---> 9a7aa409f758
Step 1 : COPY src/ /var/www/html/
 ---> Using cache
 ---> 2caaddf453c7
Step 2 : EXPOSE 80
 ---> Using cache
 ---> 586ae0b06cb6
Successfully built 586ae0b06cb6
cfortier (master ✗) docker docker push cfortier/cs90
The push refers to a repository [cfortier/cs90] (len: 1)
Sending image list
Pushing repository cfortier/cs90 (1 tags)
511136ea3c5a: Image already pushed, skipping
36fd425d7d8a: Image already pushed, skipping
aaabd2b41e22: Image already pushed, skipping
35a6381b9f4d: Image already pushed, skipping
afcdff084bd7: Image already pushed, skipping
31d7cb82d7f6: Image already pushed, skipping
8859f0d2ad74: Image already pushed, skipping
df1292c4ddd3: Image already pushed, skipping
f006da200a6b: Image already pushed, skipping
0f5eee2e9640: Image already pushed, skipping
dfbbbc57dc8d: Image already pushed, skipping
6334d7d18ad2: Image already pushed, skipping
8427dfb9bf61: Image already pushed, skipping
7c9fb13aa552: Image already pushed, skipping
dd706a4cde74: Image already pushed, skipping
b57ae04e2595: Image already pushed, skipping
a5e61c1cabe0: Image already pushed, skipping
a26f9567e49a: Image already pushed, skipping
cf7371e81440: Image already pushed, skipping
ee8d1ab29204: Image already pushed, skipping
9a7aa409f758: Image already pushed, skipping
2caaddf453c7: Image successfully pushed
586ae0b06cb6: Image successfully pushed
Pushing tag for rev [586ae0b06cb6] on {https://cdn-registry-1.docker.io/v1/repositories/cfortier/cs90/tags/a}
```

HA Proxy[v]

HAProxy is a free, ***very*** fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. In our application we will run HA Proxy on each instance to serve traffic on port 80. HA Proxy will be configured at run time to know which Docker containers should be enabled and which ones will be disabled. The benefit of this is that a server can be placed in maintenance mode in HA Proxy while it is serving traffic. This has the effect of allowing any current connection to continue but will not server new traffic to it. This gives us the ability to deploy the new container while simultaneously running the current container until its transactions finish.

Deployment Process

The process of deployment is rather detailed and needs to be designed to avoid any race conditions. With that in mind, we will view a deployment as a single transaction and will wait for all nodes to complete the transaction. For this demonstration we will manually handle this, however it should be done programmatically for an actual production system.

The process to initialize a node (on first boot) will be:

1. Query Consul `cs90/active/image` for the active Docker image
2. Pull that image from the Docker Registry
3. Run that image
4. Update HA Proxy to serve traffic to that node

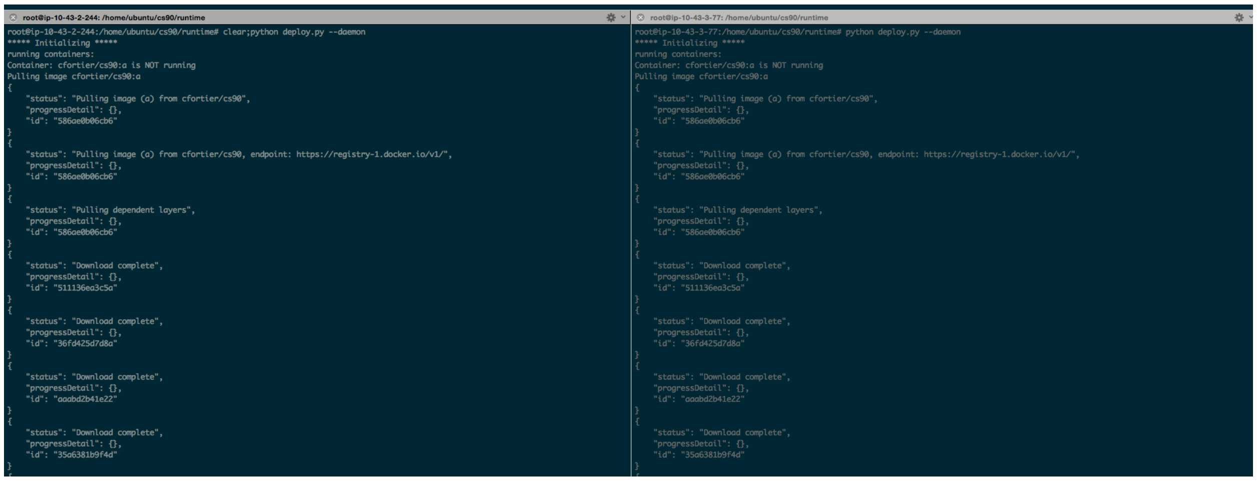
The process to deploy a new version to a running node will be:

1. Poll Consul every second for changes to the `cs90/stage/image`
2. If a change is detected, pull and run the designated image

3. Wait for the `cs90/active/image` value to be updated to the stage value
4. Update HA Proxy to server traffic to the new container while placing the original in maintenance mode
5. End transaction

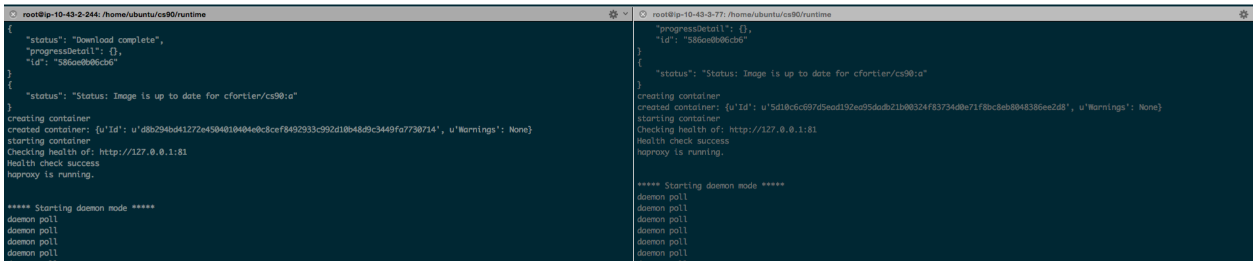
In the following screenshots we will go through an initialization and then a deployment. The top panel is one of the cluster members and will be used to update the Consul values. The two bottom panels are host nodes.

Demo Initialization



```
root@ip-10-43-2-244: /home/ubuntu/cs90/runtime
root@ip-10-43-2-244:/home/ubuntu/cs90/runtime# clear;python deploy.py --daemon
***** Initializing *****
running containers:
Container: cfortier/cs90:a is NOT running
Pulling image cfortier/cs90:a
{
    "status": "Pulling image (a) from cfortier/cs90",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Pulling image (a) from cfortier/cs90, endpoint: https://registry-1.docker.io/v1/",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Pulling dependent layers",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "511136ea3c5a"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "36fd425d7d8a"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "aaabd2b41e22"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "35a6381b9f4d"
}
```

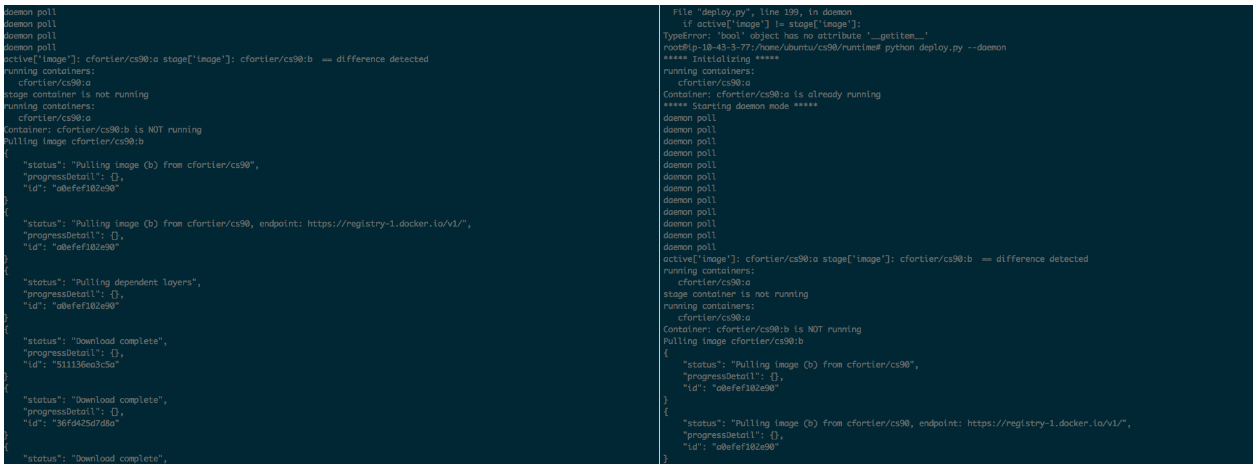
```
root@ip-10-43-3-77: /home/ubuntu/cs90/runtime
root@ip-10-43-3-77:/home/ubuntu/cs90/runtime# python deploy.py --daemon
***** Initializing *****
running containers:
Container: cfortier/cs90:a is NOT running
Pulling image cfortier/cs90:a
{
    "status": "Pulling image (a) from cfortier/cs90",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Pulling image (a) from cfortier/cs90, endpoint: https://registry-1.docker.io/v1/",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Pulling dependent layers",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "511136ea3c5a"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "36fd425d7d8a"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "aaabd2b41e22"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "35a6381b9f4d"
}
```



```
root@ip-10-43-2-244: /home/ubuntu/cs90/runtime
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Status: Image is up to date for cfortier/cs90:a"
}
creating container
created container: {u'Id': u'd8b294bd41272e4504010404e0c8cef8492933c992d10b48d9c3449fa7730714', u'Warnings': None}
starting container
Checking health of: http://127.0.0.1:81
Health check success
haproxy is running.


***** Starting daemon mode *****
daemon poll
daemon poll
daemon poll
daemon poll
```

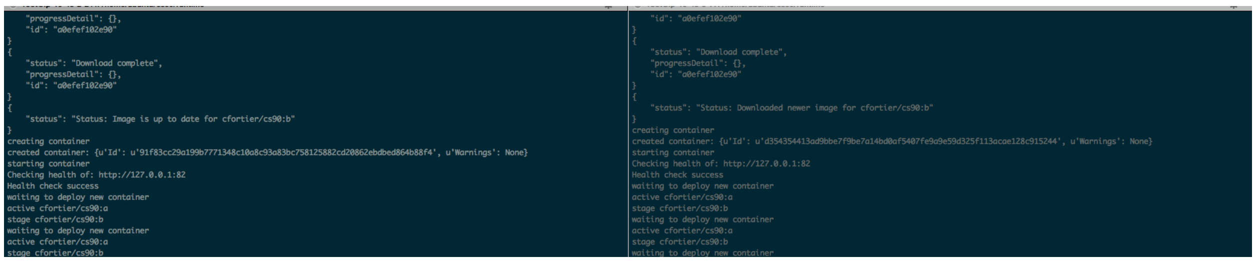
```
root@ip-10-43-3-77: /home/ubuntu/cs90/runtime
    "progressDetail": {},
    "id": "586ae0b06cb6"
}
{
    "status": "Status: Image is up to date for cfortier/cs90:a"
}
creating container
created container: {u'Id': u'5d10c6c697d5ead192ea95dadb21b00324f83734d0e71f8bc8eb8048386ee2d8', u'Warnings': None}
starting container
Checking health of: http://127.0.0.1:81
Health check success
haproxy is running.


***** Starting daemon mode *****
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
```

Demo Deployment of B



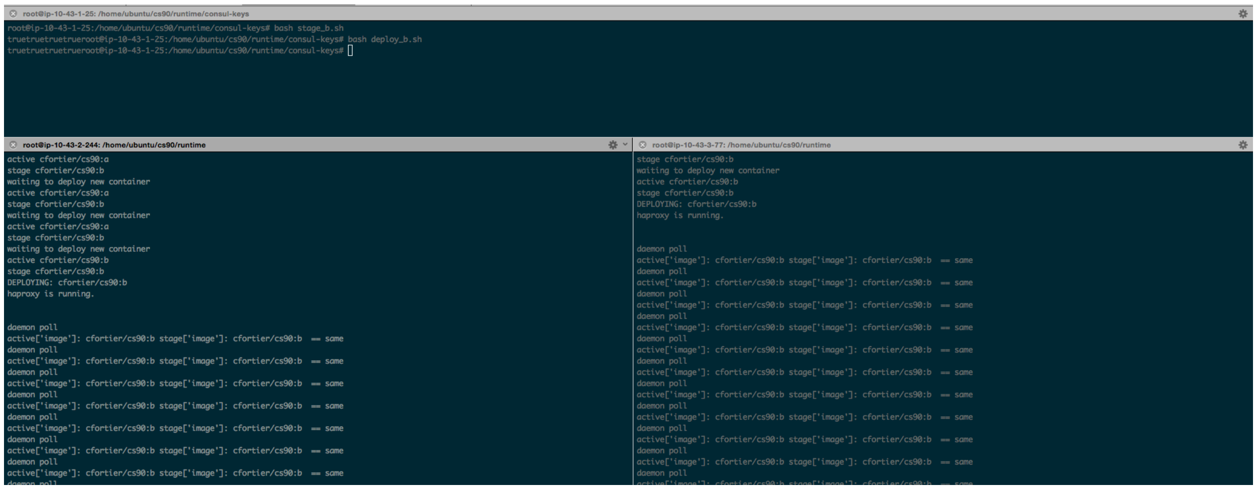
```
daemon poll
daemon poll
daemon poll
daemon poll
active['image']: cfortier/cs90:a stage['image']: cfortier/cs90:b  == difference detected
running containers:
   cfortier/cs90:a
stage container is not running
running containers:
   cfortier/cs90:a
Container: cfortier/cs90:b is NOT running
Pulling image cfortier/cs90:b
{
    "status": "Pulling image (b) from cfortier/cs90",
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Pulling image (b) from cfortier/cs90, endpoint: https://registry-1.docker.io/v1/",
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Pulling dependent layers",
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "511136ea3c5a"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "36fd425d7d8a"
}
{
    "status": "Download complete",
```

```
  File "deploy.py", line 199, in daemon
    if active['image'] != stage['image']:
TypeError: 'bool' object has no attribute '__getitem__'
root@ip-10-43-3-77:/home/ubuntu/cs90/runtime# python deploy.py --daemon
***** Initializing *****
running containers:
   cfortier/cs90:a
Container: cfortier/cs90:a is already running
***** Starting daemon mode *****
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
daemon poll
active['image']: cfortier/cs90:a stage['image']: cfortier/cs90:b  == difference detected
running containers:
   cfortier/cs90:a
stage container is not running
running containers:
   cfortier/cs90:a
Container: cfortier/cs90:b is NOT running
Pulling image cfortier/cs90:b
{
    "status": "Pulling image (b) from cfortier/cs90",
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Pulling image (b) from cfortier/cs90, endpoint: https://registry-1.docker.io/v1/",
    "progressDetail": {},
    "id": "a0efef102e90"
}
```



```
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Status: Image is up to date for cfortier/cs90:b"
}
creating container
created container: {u'Id': u'91f83cc29a199b7771348c10a8c93a83bc758125882cd20862ebdbed864b88f4', u'Warnings': None}
starting container
Checking health of: http://127.0.0.1:82
Health check success
waiting to deploy new container
active cfortier/cs90:a
stage cfortier/cs90:b
waiting to deploy new container
active cfortier/cs90:a
stage cfortier/cs90:b
```

```
    "id": "a0efef102e90"
}
{
    "status": "Download complete",
    "progressDetail": {},
    "id": "a0efef102e90"
}
{
    "status": "Status: Downloaded newer image for cfortier/cs90:b"
}
creating container
created container: {u'Id': u'd354354413ad9bbe7f9be7a14bd0af5407fe9a9e59d325f113acae128c915244', u'Warnings': None}
starting container
Checking health of: http://127.0.0.1:82
Health check success
waiting to deploy new container
active cfortier/cs90:a
stage cfortier/cs90:b
waiting to deploy new container
active cfortier/cs90:a
stage cfortier/cs90:b
waiting to deploy new container
```



root@ip-10-43-1-25: /home/ubuntu/cs90/runtime/consul-keys

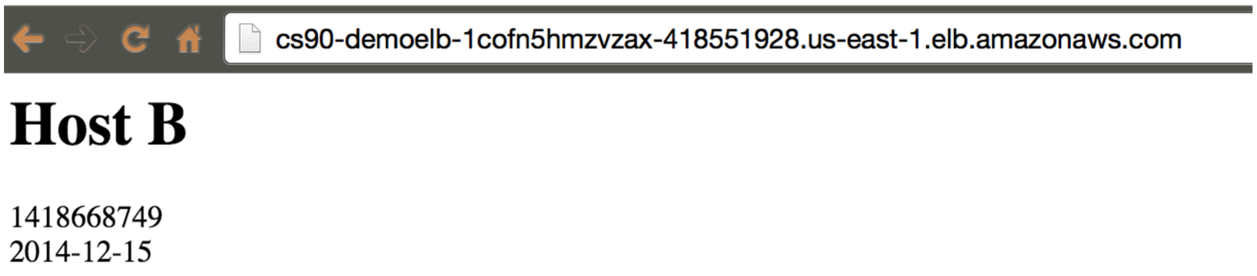
```
root@ip-10-43-1-25:/home/ubuntu/cs90/runtime/consul-keys# bash stage_b.sh
truetruetruetrueroot@ip-10-43-1-25:/home/ubuntu/cs90/runtime/consul-keys# bash deploy_b.sh
truetruetruetrueroot@ip-10-43-1-25:/home/ubuntu/cs90/runtime/consul-keys# 
```

root@ip-10-43-2-244: /home/ubuntu/cs90/runtime

```
active cfortier/cs90:a
stage cfortier/cs90:b
waiting to deploy new container
active cfortier/cs90:a
stage cfortier/cs90:b
waiting to deploy new container
active cfortier/cs90:a
stage cfortier/cs90:b
waiting to deploy new container
active cfortier/cs90:b
stage cfortier/cs90:b
DEPLOYING: cfortier/cs90:b
haproxy is running.


daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
```

root@ip-10-43-3-77: /home/ubuntu/cs90/runtime

```
stage cfortier/cs90:b
waiting to deploy new container
active cfortier/cs90:b
stage cfortier/cs90:b
DEPLOYING: cfortier/cs90:b
haproxy is running.


daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
active['image']: cfortier/cs90:b stage['image']: cfortier/cs90:b  == same
daemon poll
```


cs90-demoelb-1cofn5hmzvzax-418551928.us-east-1.elb.amazonaws.com

Host B

1418668749
2014-12-15

Conclusion and Next Steps

This paper has covered a wide range of topics but effectively shows how to build an entire AWS infrastructure from scratch, build a Docker Image, then execute a rolling deploy of those containers. There are several areas of improvement needed before this process would be considered "production ready". Specific things that would need to be implemented:

1. Each node should report itself as providing a service in Consul. This would allow for real-time monitoring of the status of each node and would allow us to detect inconsistencies.
2. A command and control center would need to be implemented to execute all key updates (as opposed to the manual bash scripts we used).
3. Some mechanism would need to be implemented if the runtime scripts fail for some reason. Though it isn't documented in pictures here, I

encountered some random issues that caused a script to fail. However re-running the script worked fine.

References

[i] https://consul.io/intro/index.html
[ii] https://consul.io/docs/internals/architecture.html

[iii] https://docs.docker.com/userguide/dockerhub/

[iv] https://docs.docker.com/reference/builder/
[v] http://www.haproxy.org/