Virtual Design Master

Challenge 4- Orchestration

Robots will be sent back to Earth to attempt to secure different areas across the globe. So you will need to create an orchestrated system to rebuild the infrastructures.

Submitted by-Harshvardhan Gupta

7/28/2015

Table of Contents

1. Executive Summary

1.1.1 Project Synopsis

Now that we've established ourselves across the solar system, it is time to make plans for taking our home planet back from the zombies. Select teams of humans and robots will be sent back to Earth to attempt to secure different areas across the globe. We will of course need working infrastructures help us with this as we reclaim the planet. Chances of sending back IT people are pretty slim, so you will need to create an orchestrated system to rebuild the infrastructures. We know this is not an easy task so we are going to start small. Our test will be a simple web application that displays "Welcome Back to Earth!". Since we are evaluating the best way to accomplish orchestrating our infrastructure, you will need to use two different orchestration tools, two different operating system, and two different web servers. Because we don't know what type of infrastructure we're going to end up with (we could end up with a mix of equipment and operating systems), the web servers must run inside of a container.

1.1.2 Intended Viewers

This document is specifically written for technical people responsible for deploying container based web application "Welcome back to Earth!" Document provides guidelines /code to deploy application using Docker.

1.1.3 Project Vision

Orchestrate web application deployment using container on two supported Operating system.

1.1.4 Project Requirements

Reference	Description
R001	Need to create an Orchestrated system to rebuild the infrastructure
R002	Deploy web application that displays "Welcome Back to Earth!"
R003	Build instructions with complete walkthrough

1.1.5 Project Constraints

Reference	Description
C001	Use two different Orchestration tools
C002	Use two different Operating System
C003	Use two different Web server
C004	the Web server must run under container

1.1.6 Project Assumptions

Reference	Description

A001	64-bit OS will be used (Debian and RPM based OS)
A002	Apache and Nginx will used as preferred web server
A003	Automation workflows for a server deployment into a virtual data center.
A004	Docker central images repo and Git hub clone available on either bases
A005	Working DNS, AD, DHCP and NTP infrastructure is already established
A006	Standalone Puppet and Chef Installed Gold images for either OS are stored as VM Template
A007	A local MTA that allows the Orchestration tool to send email notifications

1.1.7 Project Risks

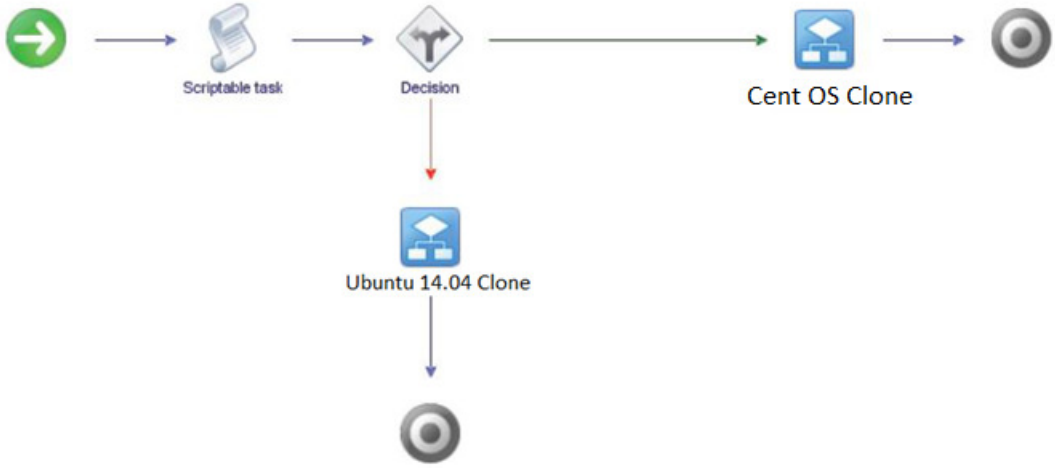
Reference	Description
I001	Container security is biggest challenge
I002	Containers are still not production ready for mission critical deployment
I003	Docker daemon attack surface-securing using SSL is only straight forward way

2. Automation workflows for a server deployment into a virtual data center

We would be deploying Cent OS 7 and Ubuntu 14.04 LTS automatically using vCenter Server Orchestrator. I have prepared two Gold Virtual Machines which would be cloned for automatic Server Deployments. Here are the steps

1. Separate Workflows created to clone the Gold Virtual Machines. One for Cent OS and second for Ubuntu Virtual Machine. If the Workflow is run, it will only ask for New Virtual Machine name, remaining all the inputs are configured as attributes.

2. To make this workflow simpler for users, both the Workflows are combined using a simple scriptable task and a decision making object. The script just maps the entered Virtual Machine name to respective cloning Workflow and the Decision making object select the appropriate cloning Workflow depending on the selection, i.e., Cent OS or Ubuntu.



3. The combined Workflow required just 2 inputs, Guest Operating type and the new Virtual Machine name and it will automatically deploy the Virtual Machine on the Virtual Data Center.

4. This workflow will clone the Gold Virtual Machine and deploy new Virtual Machine under Cloned Virtual Machines folder on the vdm-GoldImages Datastore.

2.1.1 Conceptual workflow

To deploy containerized Webserver on Ubuntu 14.04 LTS, we need to install the Docker container first. This can be automated by several ways, we could use a script to install or push some commands.

I am using below Script to install the Docker through Apt Package Manager. Docker can be installed automatically on your VM by adding this script when launching it.

```
$ curl -s https://get.docker.io/ubuntu/ | sudo sh >./install.sh
```

This is a curl script for easy installation. Looking at the individual pieces of this script will allow us to understand the process better:

1.First, the script checks whether our Advanced Package Tool (APT) system can deal with https URLs, and installs apt-transport-https if it cannot:

```
# Check that HTTPS transport is available to APT
if [ ! -e /usr/lib/apt/methods/https ]; then apt-get update apt-get install -y apt-transport-https
fi
```

2. Then it will add the Docker repository to our local key chain:

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9
```

3.Finally, it adds the Docker repository to the APT sources list, and updates and installs the lxc-docker package:

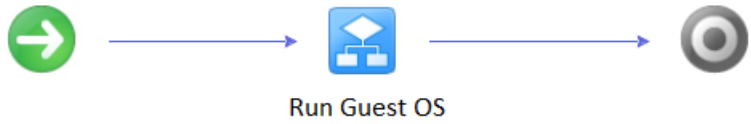
```
$ sudo sh -c "echo deb https://get.docker.io/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
$ sudo apt-get update
$ sudo apt-get install lxc-docker
```

2.1.2 Actual workflow

1. Created script file with the name install.sh.

2. The file is copied from VCO local machine to Cloned Ubuntu 14.04 LTS using the below Workflow.

3. Second Workflow is created to run the Guest OS which will execute script and install Docker.



4. Create final Workflow and combine both the Workflow to automate the complete task.

Note-Installation of Docker on Cent OS 7 can be performed in same way using curl script.

3. Docker Workflow

3.1.1 Pull-use-modify-commit-push

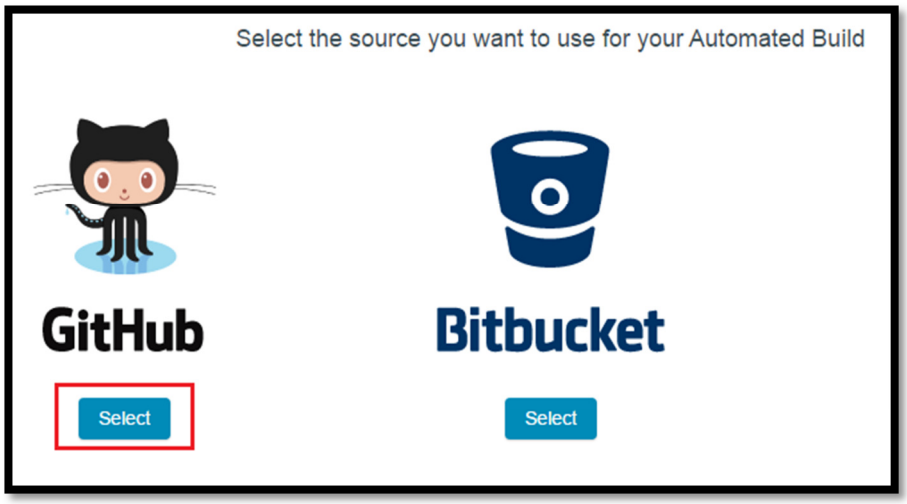
A typical Docker workflow is like:

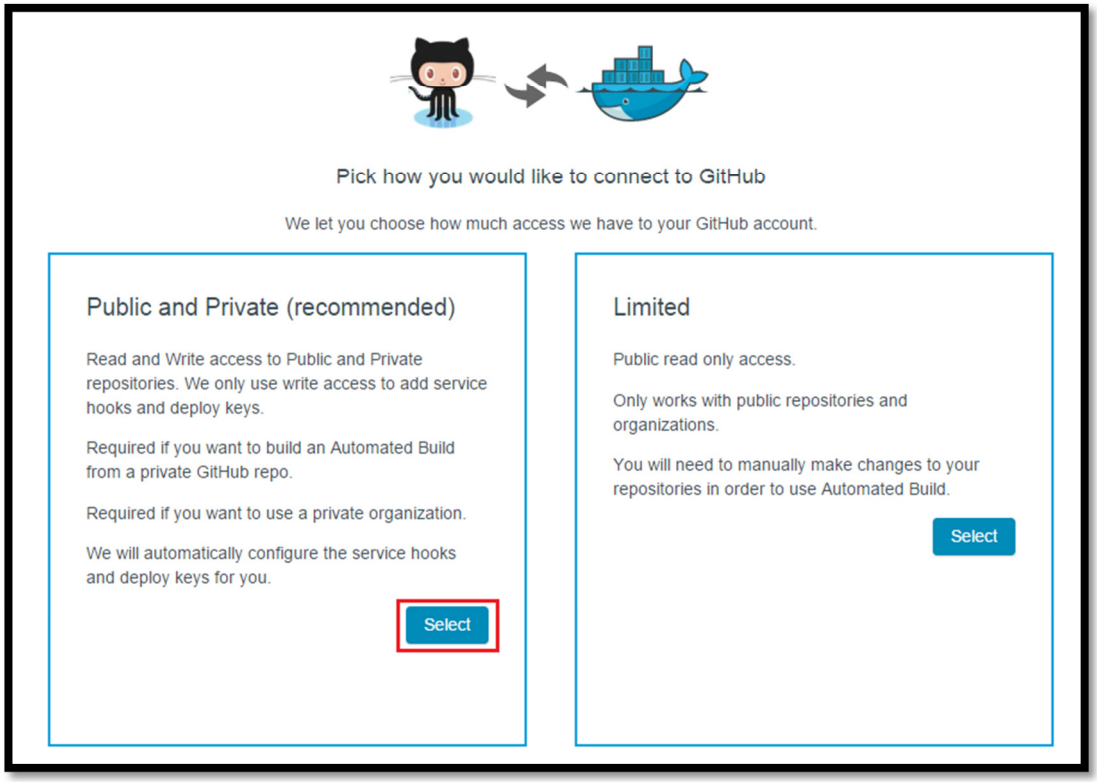
1. Prepare a list of requirements to run web application.
2. Use one of our own public images to satisfy most of these requirements.
3. Next, fulfil the remaining requirements by writing a Docker file (which is preferable since we will be able to make the build repeatable).
4. Push new image to the public Docker registry so that the Mars human community can use it too.

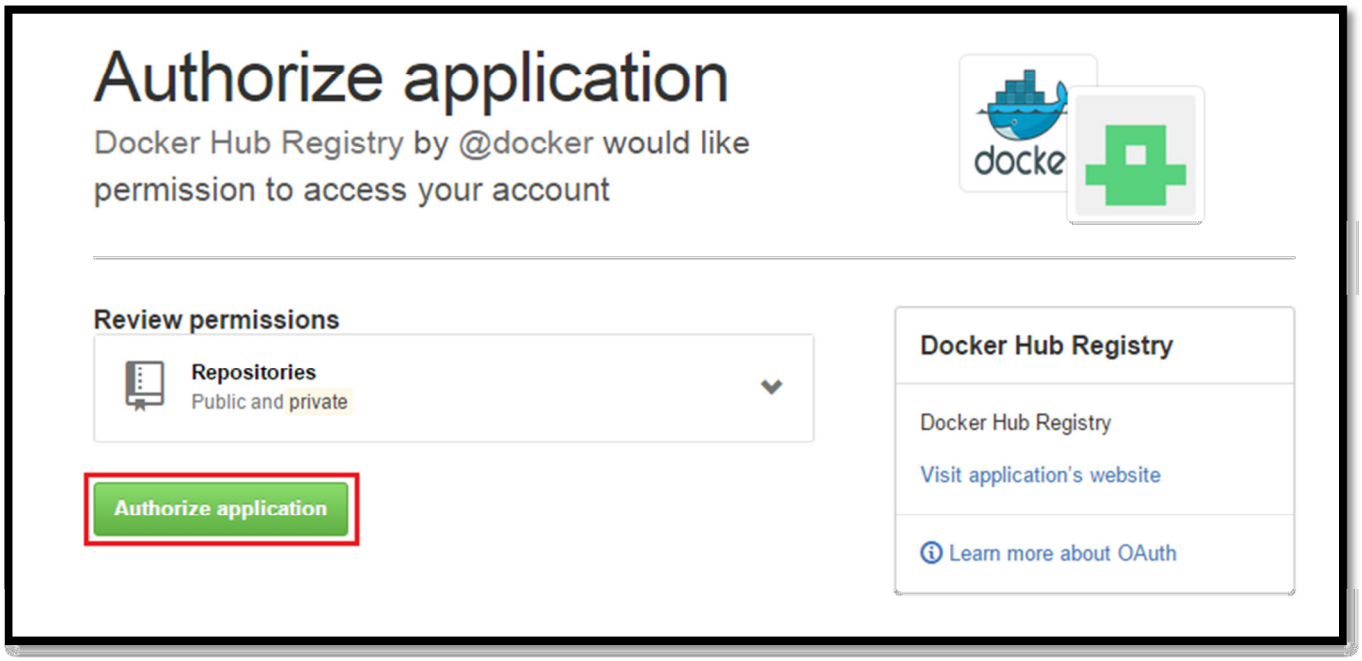
3.1.2 Automated Builds

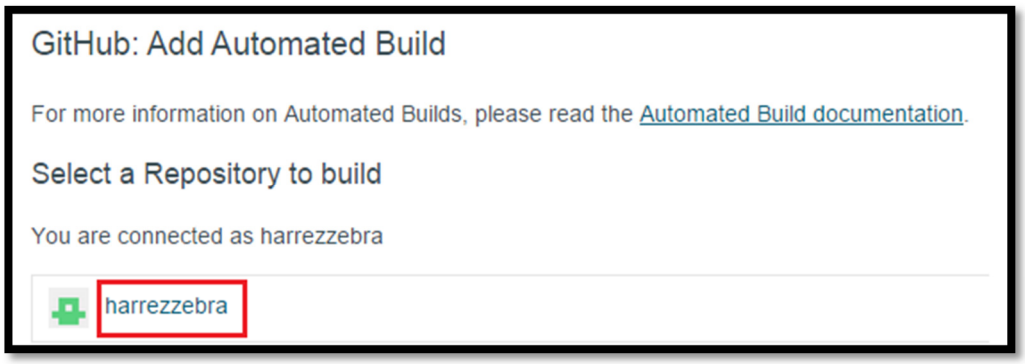
Follow steps to build automated repository on Docker Hub for later use.

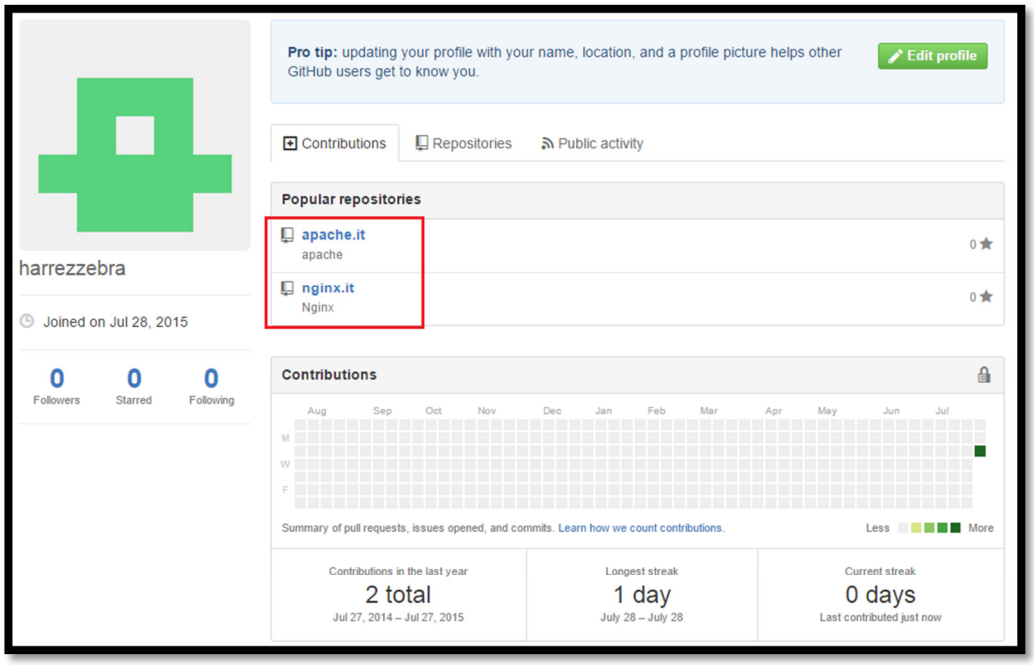
Sign-up for GitHub account.

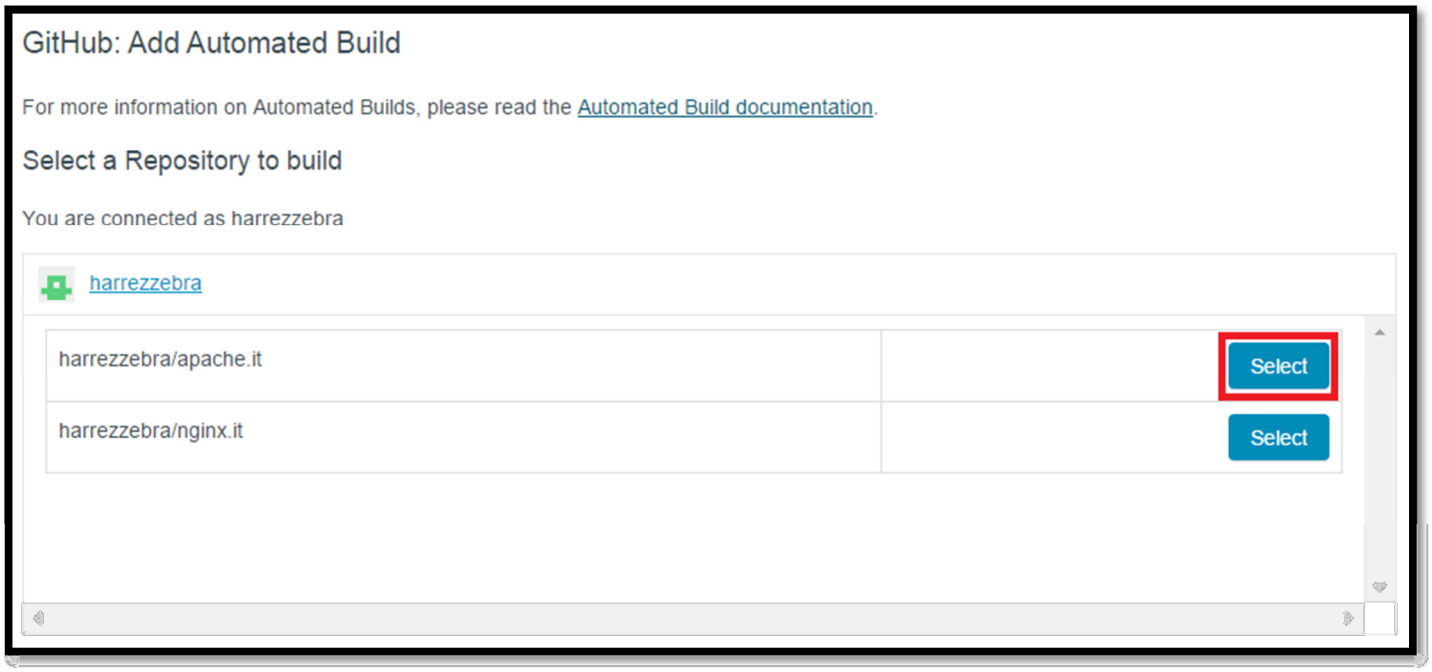

Select the source you want to use for your Automated Build
GitHub
Select
Bitbucket
Select

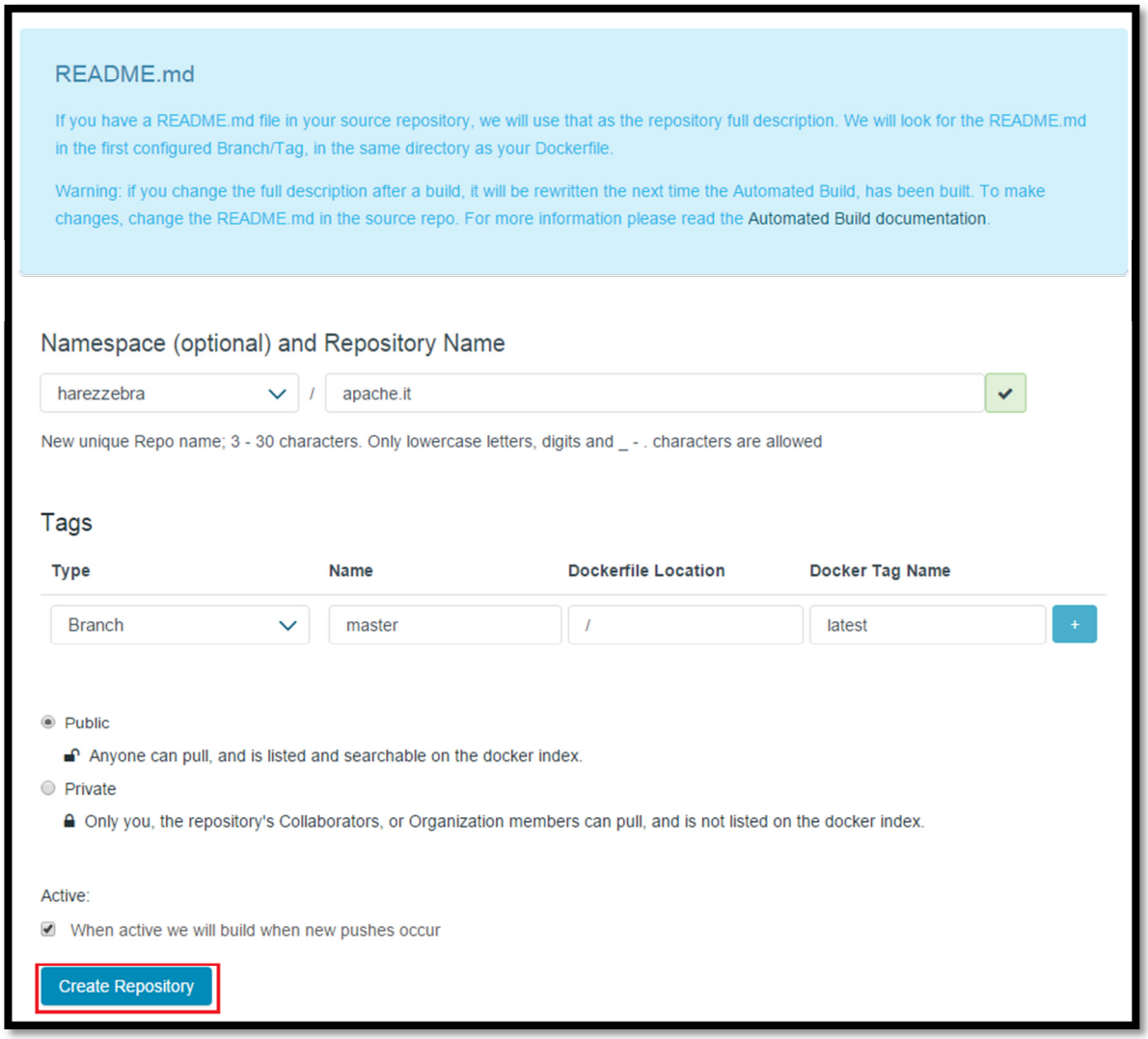

Pick how you would like to connect to GitHub
We let you choose how much access we have to your GitHub account.
Public and Private (recommended)
Read and Write access to Public and Private repositories. We only use write access to add service hooks and deploy keys.
Required if you want to build an Automated Build from a private GitHub repo.
Required if you want to use a private organization.
We will automatically configure the service hooks and deploy keys for you.
Select
Limited
Public read only access.
Only works with public repositories and organizations.
You will need to manually make changes to your repositories in order to use Automated Build.
Select

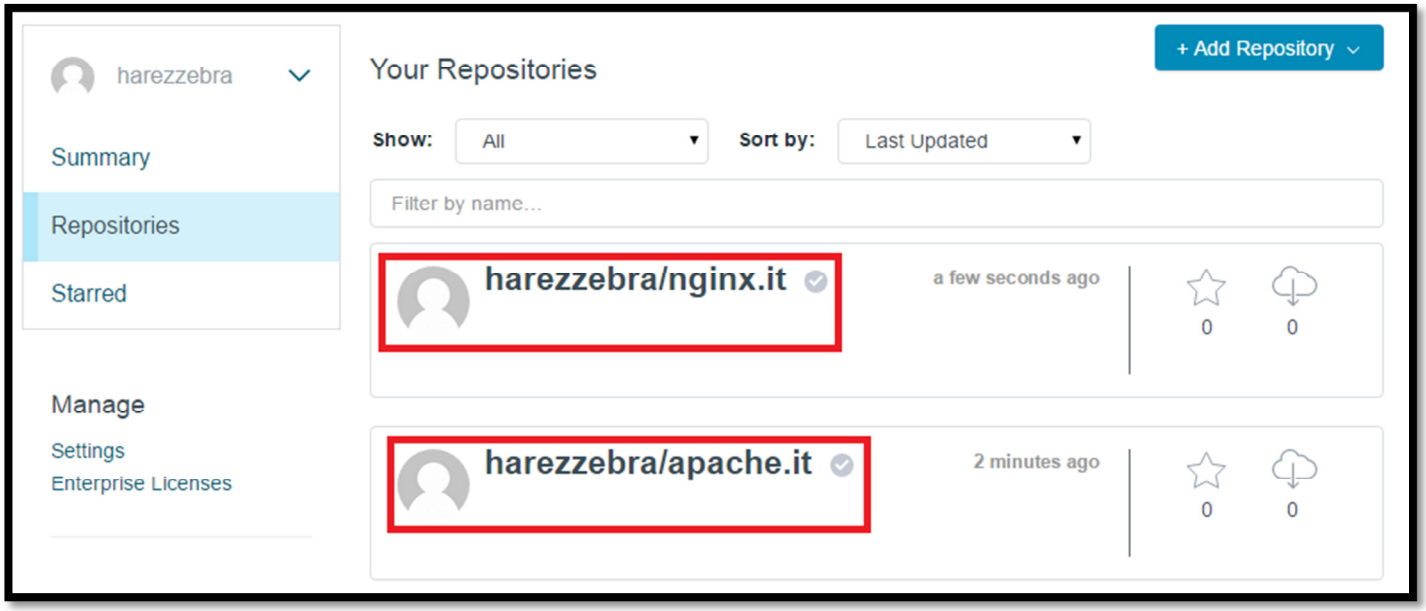

Authorize application
Docker Hub Registry by @docker would like permission to access your account
Review permissions
Repositories
Public and private
Authorize application
Docker Hub Registry
Docker Hub Registry
Visit application's website
Learn more about OAuth


GitHub: Add Automated Build
For more information on Automated Builds, please read the Automated Build documentation.
Select a Repository to build
You are connected as harrezzebra
harrezzebra


Pro tip: updating your profile with your name, location, and a profile picture helps other GitHub users get to know you.
Edit profile
Contributions
Repositories
Public activity
Popular repositories
apache.it
apache
nginx.it
Nginx
harrezzebra
Joined on Jul 28, 2015
0 Followers
0 Starred
0 Following
Contributions
Summary of pull requests, issues opened, and commits. Learn how we count contributions.
Less More
Contributions in the last year
2 total
Jul 27, 2014 – Jul 27, 2015
Longest streak
1 day
July 28 – July 28
Current streak
0 days
Last contributed just now


GitHub: Add Automated Build
For more information on Automated Builds, please read the Automated Build documentation.
Select a Repository to build
You are connected as harrezzebra
harrezzebra
harrezzebra/apache.it
Select
harrezzebra/nginx.it
Select


README.md
If you have a README.md file in your source repository, we will use that as the repository full description. We will look for the README.md in the first configured Branch/Tag, in the same directory as your Dockerfile.
Warning: if you change the full description after a build, it will be rewritten the next time the Automated Build, has been built. To make changes, change the README.md in the source repo. For more information please read the Automated Build documentation.
Namespace (optional) and Repository Name
harezzebra
/
apache.it
New unique Repo name; 3 - 30 characters. Only lowercase letters, digits and _ - . characters are allowed
Tags
Type
Name
Dockerfile Location
Docker Tag Name
Branch
master
/
latest
+
Public
Anyone can pull, and is listed and searchable on the docker index.
Private
Only you, the repository's Collaborators, or Organization members can pull, and is not listed on the docker index.
Active:
When active we will build when new pushes occur
Create Repository



4. Creating Image for Public Consumption

Now spun a test VM (either Cent OS and Ubuntu) and install docker on it.

Install with the script

Log into your machine as a user with sudo or root privileges. Make sure your existing yum packages are up-to-date.

```
$ sudo yum update
```

Run the Docker installation script.

```
$ curl -sSL https://get.docker.com/ | sh
```

This script adds the `docker.repo` repository and installs Docker. Start the Docker daemon.

```
$ sudo service docker start
```

Verify docker is installed correctly by running a test image in a container.

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from hello-world
a8219747be10: Pull complete
91c95931e552: Already exists
hello-world:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.
Digest: sha256:aa03e5d0d5553b4c3473e89c8619cf79df368babd1.7.1cf5daeb82aab55838d
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker.
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (Assuming it was not already locally available.)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

Then run below command

```
$ docker run -it ubuntu /bin/bash
```

you will end up with is a shell prompt for the Ubuntu image. Now Install Apache using below command

```
apt-get update && apt-get install -y apache2 curl
```

Once installation finishes, Setup a custom webpage to display required content. First, navigate to the directory we're going to put, by typing

```
cd /var/www/html
```

and then type

```
echo “Welcome back to Earth!” > index.html
```

then restart apache by sending below command

```
apache2ctl start
```

and type below curl oneliner to check the output you just configured

```
curl http://locahost
```

Now we need to put our image on Docker Hub public repo that we created in previous section. Before proceeding to next steps commit the changes to your docker container. Use **Docker ps** command to gather Container ID details.

```
docker commit containerID harezzebra/apache.it
```

You’ll get prompted for your Docker Hub login credentials, you can also check the activity on docker hub webpage.

Rinse and repeat steps for installing Nginx http server. You can also pull and customize official Nginx image from Docker hub and save as harezzebra/Nginx.it.

5. Puppet Configuration on Ubuntu with Apache/Nginx

We’re assuming Puppet orchestrator tool is installed as standalone in Ubuntu Gold Image template and Docker image with Apache & Nginx is stored in public repository hosted in either bases.

5.1.1 Puppet Manifest-Apache

Requirements-

- Puppet Enterprise 3.7
- Puppet 3.4

Dependencies-

- puppetlabs/stdlib (>= 4.1.0)
- puppetlabs/apt (>= 1.1.0 <= 3.0.0)
- stahnma/epel (>= 0.0.6)

Code-There is a well-supported module available for Docker. Its installation is carried out by running this command:

```
$ puppet module install garethr-docker
```

The first non-comment statement includes the docker module. The docker::image {'harezzebra/apache.it':} statement is equivalent to running the $ docker pull harezzebra/apache.it command in the console. The block of statements at the end of the recipe is equivalent to running the $ docker run --d -p '8000:80' -e 'NODE_PORT=8000' -v '/var/log/apache.it:/var/log/apache.it' harezzebra/apache.it /usr/sbin/apache2ctl command.

```
$ puppet module install garethr-docker

# Installation
include 'docker'
# Download image
docker::image {'harezzebra/apache.it':}
# Run a container
docker::run { 'apache.it-puppet':
image => 'harezzebra/apache.it',
command => '/usr/sbin/apache2ctl',
ports => '80',
volumes => '/var/log/apache.it'
}
```

5.1.2 Puppet Manifest-NginX

Requirements-

- Puppet Enterprise 3.7
- Puppet 3.4

Dependencies-

- puppetlabs/stdlib (>= 4.1.0)
- puppetlabs/apt (>= 1.1.0 <= 3.0.0)
- stahnma/epel (>= 0.0.6)

Code-

```
$ puppet module install garethr-docker

# Installation
include 'docker'
# Download image
docker::image {'harezzebra/nginx.it':}
# Run a container
docker::run { 'nginx.it-puppet':
image => 'harezzebra/nginx.it',
command => '/etc/nginx/nginx.conf:ro',
ports => '80',
volumes => '/var/log/nginx.it'
```

```
}
```

6. Chef Configuration on Cent OS with Apache/Nginx

We're assuming Chef Orchestrator tool is installed as solo in Cent OS Gold Image template and Docker image with Apache & Nginx is stored in public repository hosted in either bases.

6.1.1 Chef-Apache

Code- The first non-comment statement includes the Chef-Docker recipe. The docker_image 'harezzebra/apache.it' statement is equivalent to running the $ docker pull harezzebra/apache.it command in the console. The block of statements at the end of the recipe is equivalent to running the $ docker run --d -p '8000:8000' -e 'NODE_PORT=8000' -v '/var/log/apache.it:/var/log/apache.it' harezzebra/apache.it command.

```
# Include Docker recipe
include_recipe 'docker'
# Pull latest image
docker_image 'harezzebra/apache.it'
# Run container exposing ports
docker_container 'harezzebra/apache.it' do
detach true
port '80:8000'
env 'NODE_PORT=8000'
volume '/var/log/code.it:/var/log/apache.it'
end
```

6.1.2 Chef-NginX

Code-

```
# Include Docker recipe
include_recipe 'docker'
# Pull latest image
docker_image 'harezzebra/nginx.it'
# Run container exposing ports
docker_container 'harezzebra/nginx.it' do
detach true
port '80:8000'
env 'NODE_PORT=8000'
volume '/var/log/code.it:/var/log/nginx.it'
end
```

7. Security best practices

- Always run the docker daemon in a dedicated server.
- Unless you have a multiple-instance setup, run the docker daemon on a Unix socket.
- Take special care about bind mounting host directories as volumes as it is possible for a container to gain complete read-write access and perform irreversible operations in these directories.
- If you have to bind to a TCP port, secure it with SSL-based authentication.
- Avoid running processes with root privileges in your containers.
- There is absolutely no sane reason why you will ever need to run a privileged container in production.
- Consider enabling AppArmor/SELinux profiles in the host. This enables you to add an additional layer of security to the host.

- Unlike virtual machines, all containers share the host's kernel. So it is important to keep the kernel updated with the latest security patches.

8. Backup restore and migrate containers

Although Containers are immutable infrastructure , but we shouldn't ignore backup/restore and migration of migration of containers in a hostile environment.

8.1.1 Backing up the Containers

Run docker ps in our linux machine running docker engine with containers already created.

```
# docker ps
```

Use docker commit command in order to create the snapshot.

```
# docker commit -p Container-ID container-backup
```

See the docker image by running the command docker images as shown below.

```
# docker images
```

If you want to upload or backup the image in the docker registry hub, you can simply run docker login command to login into the docker registry hub and then push the required image.

```
# docker login
# docker tag a25ddfec4d2a harezzebra/container-backup:test
# docker push harezzebra/container-backup
```

If you don't wanna backup to the docker registry hub and wanna save the image for future use in the machine locally then we can backup the image as tarballs.

```
# docker save -o ~/container-backup.tar container-backup
```

8.1.2 Restoring the Containers

For restoring those contianers which are snapshotted as docker images. If we have pushed those docker images in the registry hub, then we can simply pull that docker image and run it out of the box.

```
# docker pull harezzebra/container-backup:test
```

If we have backed up those docker images locally as tarball file, then we can easy load that docker image using docker load command followed by the backed up tarball.

```
# docker load -i ~/container-backup.tar
```

To ensure that those docker images have been loaded successfully, we'll run docker images command.

```
# docker images
```

After the images have been loaded, we'll gonna run the docker container from the loaded image.

```
# docker run -d -p 80:80 container-backup
```

8.1.3 Migrating the Docker Containers

Migrating the containers involve both the above process ie Backup and Restore. We can migrate any docker container from one machine to another. In the process of migration, first we take the backup of the container as snapshot docker image. Then, that docker image is either pushed to the docker registry hub or saved as tarball files in the locally. If we have pushed the image to the docker registry hub, we can easily restore and run the container using docker run command from any machine we want. But if we have saved the image as tarballs locally, we can simply copy or move the image to the machine where we want to load image and run the required container.

9. References

1. Akmal Waheed's season 1 challenge 3 submission document
2. Puppetlabs.com
3. Docs.chef.io
4. Docker.io
5. Slideshare.com
6. Pluralsight.com
7. Packtpub.com
8. Blog.scottlowe.org

10. Docker Terminologies

Docker container

A Docker container can be correlated to an instance of a VM. It runs sandboxed processes that share the same kernel as the host. The term **container** comes from the concept of shipping containers.

The docker daemon

The docker daemon is the process that manages containers. It is easy to get this confused with the Docker client because the same binary is used to run both the processes. The docker daemon, though, needs the root privileges, whereas the client doesn't.

Docker client

The Docker client is what interacts with the docker daemon to start or manage containers. Docker uses a RESTful API to communicate between the client and the daemon.

Dockerfile

A Dockerfile is a file written in a **Domain Specific Language** (**DSL**) that contains instructions on setting up a Docker image. Think of it as a Makefile equivalent of Docker.

Docker registry

This is the public repository of all Docker images published by the Docker community. You can pull images from this registry freely, but to push images, you will have to register at http://hub.docker.com.