Docker: Beyond the Basics

CI/CD

Instructor

Sean P. Kane

@spkane

Follow Along Guide

Textual Slides

Prerequisites

- A recent computer and OS
  - Recent Linux, OS X, or Windows 10
  - root/admin rights
  - Sufficient resources to run one 2 CPU virtual machine (VM)
  - CPU Virtualization extensions MUST be enabled in your BIOS/EFI
  - Reliable and fast internet connectivity
- Docker Community Edition

Prerequisites

- A graphical web browser
- A text editor
- A software package manager
- Git client
- General comfort with the command line will be helpful.
- [optional] tar, wget, curl, jq, SSH client

A Note for Windows Users

This class was written from a largely Unix based perspective, but everything can be made to work in Windows with very little effort.

- Unix Variables
  - `export MY_VAR=test`
  - `echo ${MY_VAR}`
- Windows 10 Variables (powershell)
  - `$env:my_var = "test"`
  - `Get-ChildItem Env:my_var`

A Note About Proxies

Proxies can interfere with some Docker activities if they are not configured correctly.

If required, you can configure a proxy in Docker: Community Edition via the preferences.

Instructor Environment

- **Operating System**: Mac OS X (v10.12.X+)
- **Terminal**: iTerm2 (Build 3.X.X+) - https://www.iterm2.com/
- **Shell Customization**: Bash-it - https://github.com/Bash-it/bash-it
- **Shell Prompt Theme**: Zork - `export BASH_IT_THEME="zork"`
- **Shell Prompt Font**: Adobe Source Code Pro - https://github.com/adobe-fonts/source-code-pro
- **Text Editor**: Visual Studio Code (v1.X.X+) - https://code.visualstudio.com/

Docker in Translation

- **Docker client**
  - The docker command used to control most of the Docker workflow and talk to remote Docker servers.
- **Docker server**
  - The dockerd command used to launch the Docker daemon. This turns a Linux system into a Docker server that can have containers deployed, launched, and torn down via a remote client.

Docker in Translation

- **Virtual Machine**
  - In general, the docker server can be only directly run on Linux. Because of this, it is common to utilize a Linux virtual machine to run Docker on other development platforms. Docker Community Edition makes this very easy.

Docker in Translation

- **Docker images**
  - Docker images consist of one or more filesystem layers and some important metadata that represent all the files required to run a Dockerized application. A single Docker image can be copied to numerous hosts. A container will typically have both a name and a tag. The tag is generally used to identify a particular release of an image.

Docker in Translation

- **Docker containers**
  - A Docker container is a Linux container that has been instantiated from a Docker image. A specific container can only exist once; however, you can easily create multiple containers from the same image.

Testing the Docker Setup

```
$ docker images
$ docker run -d --rm --name quantum -p 18080:8080 spkane/quantum-game:latest
$ docker ps
```

- In a web browser, navigate to port 18080 on your Docker server.
  - (e.g.) http://127.0.0.1:18080/

```
$ docker kill quantum
$ docker ps
```

Exploring the Docker VM

- Based on Alpine Linux (apk)

```
$ docker run -it  --privileged --pid=host debian nsenter -t 1 -m -u -n -i sh
# cat /etc/os-release
# exit
```

- http://man7.org/linux/man-pages/man1/nsenter.1.html

```
$ docker run -ti --rm  spkane/quantum-game:latest cat /etc/os-release
```

Setting the Stage

```
$ cd ${HOME}
$ mkdir class-docker-cicd
$ cd class-docker-cicd
$ mkdir code
$ git clone https://github.com/spkane/docker201.git layout --config core.autocrlf=input
$ cd layout
$ ls
```

Automating Workflow

- Datastore
  - Postgres
- Collaborative Source Code Repository
  - Gogs
- Docker Image Repository
  - Docker Distribution
- Build, Test, and Deploy
  - Jenkins

Iterative Workflow

- Core Technology - Docker

User develops code locally (Docker)
User commits code (Gogs backed by Postgres)
Pipeline builds & tests code (Jenkins & Docker Distribution)
Pipeline deploys code to production.
and then iterate...

Composing a Docker Service

For unix: `$ alias dc='docker-compose'`
For Windows Powershell: `PS C:\> New-Alias dc docker-compose.exe`

- Open & explore docker-compose.yaml in your text editor
- Full Documentation:
  - https://docs.docker.com/compose/compose-file/

Creating a Datastore

```
$ cd compose/review/1st
$ vi docker-compose.yml
```

- Note: DB user & password

Creating a Source Repo

```
$ cd ../2nd
$ vi docker-compose.yml
```

Docker Distribution

```
$ cd ../3rd
$ vi docker-compose.yml
```

Important Note For Windows Users

- In the next section you might see:
  - a Windows Security Alert for `vpnkit.exe`, be sure and select `Allow access`.
  - Multiple `Docker for Windows - Share drive` alerts. Be sure and select `Share it` for each prompt.
- If you have problems with file mounts you may need to set: `$Env:COMPOSE_CONVERT_WINDOWS_PATHS=1`

Jenkins

```
$ cd ../../final/{unix,windows}
$ vi docker-compose.yml
$ docker-compose config
$ docker-compose up -d
$ docker-compose ps
$ docker-compose logs -f
2017/07/01 20:06:31 [ INFO] Listen: http://0.0.0.0:3000
LOG: database system is ready to accept connections
msg="debug server listening localhost:5001"
Please use the following password to proceed to installation
```

Important

Note: Don't run `docker-compose down` until class is over.

Configure Gogs

- Navigate web browser to:
  - http://127.0.0.1:10080/install
- **Database Type**: `Postgres`
- **Host**: `postgres:5432`
- **User**: `postgres`
- **Password**: `myuser-pw!`
- **SSH Port**: `10022`
- **Application URL**: `http://127.0.0.1:10080/`

Create Gogs User

Username: `myuser`
Password: `myuser-pw!`
Confirm Password: `myuser-pw!`
Email Address: `myuser@example.com`
Click: `Install Gogs`

Create GIT Repo

Click: `+`
Click: `+ New Repository`
Repository Name: `outyet`
Click: `Create Repository`

A Note For Windows Users

- In the next section you might see a GUI based password prompt from git. Be sure to provide your gogs username and password for the prompt.

Explore the Code

```
$ cd ~/class-docker-cicd/code/outyet
```

- Explore with your favorite code editor
  - Dockerfile
  - main.go
  - main_test.go

```
docker-compose up -d
```

Examine Application

- Navigate web browser to:
  - http://127.0.0.1:10088/

```
$ docker-compose down
```

First Code Commit

```
$ cd ../../..
$ cp -a outyet ../code/
$ cd ../code/outyet/
$ git init
$ git config core.autocrlf input
$ git add .
$ git commit -m "first commit"
$ git remote add origin http://127.0.0.1:10080/myuser/outyet.git
$ git push -u origin master
```

- username: `myuser`
- password: `myuser-pw!`

Test Docker Distribution

```
$ docker login 127.0.0.1:5000
```

- username: `myuser`
- password: `myuser-pw!`

```
$ docker pull spkane/quantum-game:latest
$ docker image ls spkane/quantum-game:latest
$ docker tag ${IMAGE_ID} 127.0.0.1:5000/myuser/quantum-game:latest
$ docker push 127.0.0.1:5000/myuser/quantum-game:latest
```

Configure Jenkins

```
cat ../../layout/jenkins/data/secrets/initialAdminPassword
```

- Navigate web browser to:
  - http://127.0.0.1:10081/
- Paste Administrator Password

  Click: `Continue`
  Click: `Select plugins to install`
  Click: `None`
  Click: `Install`

Configuring Jenkins

- Create Admin User
  Username: `myuser`
  Password: `myuser-pw!`
  Confirm password: `myuser-pw!`
  Full Name: `My User`
  E-Mail Address: `myuser@example.com`
  Click: `Save and Continue`

Configuring Jenkins

- Final Details
  Jenkins URL: `http://127.0.0.1:10081/`
  Click: `Save and Finish`
  Click: `Start Using Jenkins`

Components Assembled

- Postgres Database
  - https://www.postgresql.org/
- Gogs - Source Code Manager
  - https://gogs.io/
- Docker Distribution
  - https://github.com/docker/distribution
- Jenkins CI
  - https://jenkins.io/

Getting Started with Jenkins

- Navigate web browser to:
  - http://127.0.0.1:10081/
- Login to Jenkins
  Click: `create new jobs`

Note: If you have not configured Jenkins, you can login using the `admin` user and the `initialAdminPassword`.

Creating The Jenkins Job

Enter an item name: `outyet`
Click: `Freestyle project`
Click: `OK`

Configuring the Job

Description: `build and test outyet`

Gogs Webhook

- None

Source Code Management

Repository URL: http://gogs:3000/myuser/outyet.git
Branch Specifier (blank for 'any'): ``

Build Triggers

- None

Build Environment

Check: `Delete workspace before build starts`
Check: `Mask passwords and regexes`
Name/Password Pairs:

- **Name**: `DOCKER_PW`
- **Password**: `myuser-pw!`

The Build Script

```
docker login --username=myuser --password=${DOCKER_PW} 127.0.0.1:5000
docker build -t 127.0.0.1:5000/myuser/outyet:${GIT_COMMIT} .
docker push 127.0.0.1:5000/myuser/outyet:${GIT_COMMIT}
```

Post-Build Actions

- None

Click: Save

Build The Code

Click: `Build Now`

Build The Code

Click: `#1`
Click: `Console Output`

Build Results

Looking for:

```
Finished: SUCCESS
```

Automate Builds

- Navigate web browser to:
  - http://127.0.0.1:10080/
    Click: `outyet`
    Click: `Settings`
    Click: `Webhooks`

Automate Builds

Click: `Add Webhook`
Click: `Gogs`
Payload URL:
`http://jenkins:8080/gogs-webhook/?job=outyet`
Content Type: `application/json`
When should this web hook be triggered?: `Just the push event`
Check: `Active`
Click: `Add Webhook`

Automate Builds

Click: `http://jenkins:8080/gogs-webhook/?job=outyet`
Click: `Test Delivery`

- Confirm Green Checkmark

Add a Bug

```
$ cd ~/class-docker-cicd/code/outyet
$ vi main.go
```

- Modify to look like this:
  - Add `"net/url"`

Result

```
import (
    "expvar"
    "flag"
    "fmt"
    "html/template"
    "log"
    "net/http"
    "net/url"
    "os"
    "sync"
    "time"
)
```

Commit the Bug

- git add .
- git commit -m "Introducing bug"
- git push origin master

Failed Tests

```
./main.go:28: imported and not used: "net/url"
[0mThe command '/bin/sh -c \
  go get -v -d && go install -v && go test -v && \
  go build -ldflags "-s" -a -installsuffix cgo -o outyet .' \
  returned a non-zero code: 2
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Fix The Error

```
$ cd ~/ class-docker-cicd/code/outyet
$ vi main.go
```

- Modify to look like this:
  - Remove `"net/url"`

Result

```
import (
    "expvar"
    "flag"
    "fmt"
    "html/template"
    "log"
    "net/http"
    "os"
    "sync"
    "time"
)
```

Commit the Fix

```
$ git add .
$ git commit -m "Removing bug"
$ git push origin master
```

Successful Tests

```
deploy_e0ebf86decf4795cee332523e68017ce7952e094:
  digest:
  sha256:a31bc49ececbac7d79b1dd080b5167ee55b34c385e967e48bfd107f8ba5afbee
  size: 738
Finished: SUCCESS
```

What We Have Learned

- Docker Compose Basics
- Gogs (w/ Postgres)
- Docker Distribution
- Jenkins
- Creating Jenkins Jobs
- Building with Jenkins & Docker
- Testing with Jenkins & Docker
- Automating builds with web hooks

Additional Reading

- The 12-Factor App
  - http://12factor.net/
- Official Docker Documentation
  - https://docs.docker.com/
- Docker: Up and Running
  - http://shop.oreilly.com/product/0636920153566.do

Additional Learning Resources

https://www.safaribooksonline.com/

Student Survey

Please take a moment to fill out the class survey linked to in the chat channel.

O'Reilly and I value your comments about the class.

Thank you!

Any Questions?