McKesson Azure Lab 05

Joan Imrich

Practical Tutorial using Azure Resource Manager & Docker Microservices

```
                  ##           .
            ## ## ##          ==
         ## ## ## ##         ===
     /"""""""""""""""""\___/ ===
     ~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===-
     \______ 0          __/
       \    \        __/
```

@Nishava Inc.

>>

Docker cheatsheet / CLI syntax

November 9, 2017

>>

Connect to your VM / Install Docker ... see lecture materials

https://canvas.instructure.com/courses/1227361/pages/week-4
https://docs.docker.com/machine/install-machine/#install-machine-directly
ssh <login>@<ip-address>

You can use the **curl** command to install on several platforms:

$ curl -s https://get.docker.com/ | sudo sh
This currently works on: Ubuntu, Debian, Fedora, Gentoo -- **curl** and **git** are very useful utilities.

On Ubuntu type:
```
$ apt-get install curl
$ apt-get install git
```

On Red Hat like systems (CentOS, Fedora), type:
```
$ yum install curl
$ yum install git
```

Add the Docker group	$ sudo groupadd docker
Add ourselves to the group	$ sudo gpasswd -a $USER docker
Restart the Docker daemon	$ sudo service docker restart

First, list all machines: **docker-machine ls**
If you don't have one, you can create one with: **docker-machine create default**
Start the machine, if stopped: **docker-machine start default**
Connect with machine via SSH: **docker-machine ssh default**

On windows cygwin
```
if [[ ! -d "$HOME/bin" ]]; then mkdir -p "$HOME/bin"; fi && \
curl -L https://github.com/docker/machine/releases/download/v0.13.0/docker-machine-Windows-x86_64.exe >
"$HOME/bin/docker-machine.exe" && \
chmod +x "$HOME/bin/docker-machine.exe"
```

Test if Docker is Working

$ docker --version
$ docker-compose --version
$ docker-machine --version

Run Hello World
$ docker run hello-world
Run Hello World to make sure your docker environment is running, such as container busybox:
$ sudo docker run busybox echo hello world

Docker & Git Hub

For many operations you will need account login to the **Docker Hub**! (go to hub.docker.com to get login ID)
$ docker login
Username: yourdockerhubusername
Password: xxxxxxxxxx
Your credentials will be stored in ~/.docker/conf.json.
The auth section is Base64 encoding of your user name and password.
It should be owned by your user with permissions of 0600. ($chmod 600 conf.json)
You may, as well, be root all the time. Your main VM user (e.g. centos. ubuntu) has sudo privileges so you could do:
$ sudo su

Docker Image Commands

Search Images
$ docker search training
$ sudo docker search dockersamples/
$ docker search centos
$ docker search ubuntu
$ docker search debian

Download Images
$ docker pull training/web-app
$ docker pull debian:jessie
$ **sudo docker pull training/sinatra** ... Ruby based framework
$ docker pull tomcat:jre
$ sudo docker pull ubuntu:latest

View Current Images on Host
$ docker images

```
REPOSITORY           TAG          IMAGE ID         CREATED         SIZE
ubuntu               14.04        3aa18c7568fc     5 days ago      188MB
hello-world          latest       725dcfab7d63     6 days ago      1.84kB
training/webapp      latest       6fae60ef3446     2 years ago     349MB
training/namer       latest       902673acc741     2 years ago     289MB
training/sinatra     latest       49d952a36c58     3 years ago     447MB
```

Docker Build Images

We use the **$ docker build** command to build images
$ docker build -t web .
The -t flag tags an image

The . indicates the location of the Dockerfile being built

We can also **build Images** from other sources, such as a GitHub repository
$ docker build -t web https://github.com/docker-training/staticweb.git

Build the image with docker build:
$ docker build -t mckazure/sinatra:v2 .

Run the container for the new image:
$ docker run -t -i mckazure/sinatra:v2 /bin/bash

Tag the image:
$ docker tag <imageId> mckazure/sinatra:devel

Push Image to DockerHub:
$ docker push mckazure/sinatra

Remove an Image:
$ docker rmi mckazure/Sinatra

Container:
$ sudo docker ps -a

```
CONTAINER ID   IMAGE                        CREATED         STATUS         PORTS    NAMES
4a432e03f0c3   wurstmeister/zookeeper   3 minutes ago   Up 3 minutes   22/tcp   wonder_feynman
```

$docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' `4a432e03f0c3`
172.17.0.7
Remove ContainerID:

```
$docker rm 4a432e03f0c3
```

$ docker build -t web .
docker build -t web https://github.com/docker-training/staticweb.com
$ docker run -d -P training/webapp python app.py
$docker run -it -d -v $(pwd):/opt/namer -w /opt/namer -p 80:9292 training/namer *rackup*

Requires rackup! … For docker examples, training/sinatra & training/namer
rackup is a useful tool for running Rack applications, which uses the Rack::Builder DSL to configure middleware and build up applications easily. *rackup* automatically figures out the environment it is run in, and runs your application as FastCGI, CGI, or WEBrick—all from the same configuration.

Use Docker to build and test a web app and how to link Docker containers together

See Week 5 – Canvas website for "Lab5 – Addendum" showing a more complex example of testing a larger Sinatra-based web app instead of a static website such as above. Sinatra is a Rubybased web application framework. It contains a web application library and a simple Domain Specific Language or DSL for creating web applications. Unlike more complex web application frameworks, like Ruby on Rails, Sinatra does not follow the model–view–controller pattern but rather allows you to create quick and simple web app that takes incoming URL parameters and output them as a JSON hash.

Dockerfile

Dockerfile is a set of instructions that build up a Docker image. **Dockerfile instructions are executed in a sequence order.** Each instruction creates a new layer in the image. Instructions are cached. If no changes are detected then the instruction is skipped and the cached layer used. (see command $ docker system purge)

FROM instruction MUST be the first non-comment instruction
You can only have one FROM and ENTRYPOINT command
Lines starting with # are treated as comments

The docker build command builds an image from a Dockerfile
# Download from https://github.com/docker-training/staticweb

```
FROM ubuntu:14.04
MAINTAINER Docker Education Team <education@docker.com>
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Hi, I am in your container' \
>/usr/share/nginx/html/index.html
CMD [ "nginx", "-g", "daemon off;" ]
EXPOSE 80
```

$ cat namer/Dockerfile ... from command ... **$docker pull training/namer** ... which implements

```
FROM ubuntu:14.04
MAINTAINER Education Team at Docker <education@docker.com>
RUN apt-get update && apt-get install --no-install-recommends -y curl wget git ruby ruby-dev
RUN gem install --no-ri --no-rdoc bundler sinatra faker i18n tilt rack rack-protection
sinatra-reloader
EXPOSE 9292
WORKDIR /opt/namer
CMD ["rackup", "--host", "0.0.0.0"]
```

Data Volumes

Data that is saved during a container's execution is destroyed when the container exits. Volumes are special directories in containers that are separate from the Union File System that can persist or share data. Also supports mounting shared storage such as NFS.

Create a data volume inside a container
$ docker run -d -P --name zweb -v /webapp training/webapp python app.py

Find the volume on the host by listing the 'source' path
$ docker inspect webapp

Mount a host directory as a data volume
$ docker run -d -P --name zweb -v /src/webapp:/webapp training/webapp python app.py

Also have the ability to create Data Volume Containers to persist data and share between containers, such as database data that can be backed up or restored.

Containers, Docker Configurement & Network Tips

Create "docker group" for Docker user with Admin (sudo or su) Priviledges
$groups … man groupadd, man users / useradd ...

The docker or dockerroot user is root equivalent. It provides root level access.You should restrict access to it like you would protect root. Add the docker group if it is not there already.
$ sudo groupadd docker
Add user centos ($USER) to the group
$ sudo gpasswd -a $USER dockerroot

Restart the Docker daemon
$ sudo systemctl restart docker.service

Examine Processes Running on local computer system (*versus* docker ps -a)

$ ps -ef | grep docker
$ ps -ef | less … then type docker …hit space bar to scroll down or "n" for next, "b" for back
$ sudo docker ps -a

Get IP Address on your computer system (versus docker containers)

$ipconfig
$ curl -4 icanhazip.com
$ ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'

Get IP Address of a Single Docker Container

The command can be used to get Docker container's IP address is docker inspect with some options:
$ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' *container_id_or_name*

Get IP Address of Container

Let's see an example which we will get IP address of a Docker container which has ID 4a432e03f0c3:
$ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 4a432e03f0c3
172.17.0.2

With old version of Docker, we can use the following syntax:
docker inspect --format '{{ .NetworkSettings.IPAddress }}' container_id_or_name

Starting the Docker Daemon on Red Hat family
Once the package is installed, we can start the Docker daemon.
On Red Hat Enterprise Linux 6 and CentOS 6 you can use commands:
$ sudo service start docker
If we want Docker to start at boot we should also:
$ sudo service enable docker
On Red Hat Enterprise 7, CentOS 7, Fedora to start the Docker service and configure it to start at the boot time, type
$ sudo systemctl enable docker.service   # and
$ sudo systemctl start docker.service
Startup at the boot time is enabled by the first command:
$ sudo systemctl enable docker.service

We can stop the running container
$ docker stop <yourContainerID>
We could remove our container too
$ docker rm <yourContainerID>

Image Commands

$ docker images

```
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
debian                      jessie              25fc9eb3417f        5 days ago          123MB
hello-world                 latest              725dcfab7d63        5 days ago          1.84kB
dockersamples/visualizer    latest              fa9f23efb4f5        3 weeks ago         148MB
dockersamples/static-site   latest              f589ccde7957        20 months ago       191MB
training/webapp             latest              6fae60ef3446        2 years ago         349MB
training/namer              latest              902673acc741        2 years ago         289MB
training/sinatra            latest              49d952a36c58        3 years ago         447MB
```

Download a user image.
$ docker pull training/docker-fundamentals-image
Download the ubuntu image
$ sudo docker pull ubuntu:latest
$ docker imges
We can also build from other sources, like GitHub
$ docker build -t web https://github.com/docker-training/staticweb.git

docker rm -f $(docker ps -aq) (will remove all of your containers)
docker network rm $(docker network ls -q) (will remove all of your networks)
docker run --rm -v /var/lib/docker/network/files:/network busybox rm /network/local-kv.db

Running Docker Containers

We can show all containers by adding the -a option for the command:
$ sudo docker ps -a
By default, the command shows all running containers as the following:

```
CONTAINER ID  IMAGE                     CREATED        STATUS        PORTS    NAMES
260f108e4271  wurstmeister/kafka        2 minutes ago  Up 2 minutes           tender_gates
4a432e03f0c3  wurstmeister/zookeeper    3 minutes ago  Up 3 minutes  22/tcp   wonder_feynman
745bde570fde  dockersamples/static-site 3 minutes ago Up 3 minutes  80/tcp, 443/tcp
loving_wescoff
```

$ sudo docker run -d dockersamples/static-site
$ sudo docker stop 745bde570fde
$ sudo docker rm 745bde570fde

$docker run --name static-site -e AUTHOR="tourist" -d -P dockersamples/static-site
$docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' static-site
$docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' f0e070549384

Cleaning Up Docker

After working with Docker for some time, you start accumulating development junk: unused volumes, networks, exited containers and unused images. Prune is a very useful command (also works for volume and network sub-commands)

$ docker system prune
Remove Dangling Volumes

dangling volumes are volumes not in use by any container. To remove them, combine two commands: first, list volume IDs for dangling volumes and then remove them.

$ docker volume rm $(docker volume ls -q -f "dangling=true")
Remove Exited Containers

The same principle works here too. First, list the containers (only IDs) you want to remove (with filter) and then remove them (consider rm -f to force remove).

$ docker rm $(docker ps -q -f "status=exited")
Remove Dangling Images
dangling images are untagged images, that are the leaves of the images tree (not intermediary layers).

$ docker rmi $(docker images -q -f "dangling=true")
Autoremove Interactive Containers

When you run a new interactive container and want to avoid typing rm command after it exits, use --rm option. Then when you exit from the created container, it will be automatically destroyed

docker run -it --name CoreServerCMD microsoft/windowsservercore cmd.exe
winpty.exe docker run -it --name CoreServerCMD microsoft/windowsservercore cmd.exe

```
docker attach CoreServerCMD
docker inspect CoreServerCMD
docker inspect -f '{{.Config.Hostname}}'  CoreServerCMD
docker inspect -f '{{.Config.Image}}'  CoreServerCMD
docker inspect -f '{{.NetworkSettings.IPAddress}}'  CoreServerCMD
```

We can filter by using | operator:
After getting the ID or name of the Docker container, we can go ahead to get its IP address.
docker ps -a | grep kafka

Inspect Docker Resources

Jq Usage - jq is a lightweight and flexible command-line JSON processor. It is like sed for JSON data. You can use it to slice and filter, and map and transform structured data with the same ease that sed, awk, grep and friends let you play with text.

Test jq command:

```
$ docker run --rm --name jq realguess/jq:v1.4  sh -c 'echo "{\"foo\":\"bar\"}" | jq .'
        {
          "foo": "bar"
        }
```

Start an interactive container with jq:
$ docker run -it --rm --name jq realguess/jq:v1.4
This will drop into /bin/sh, then jq command can be run:
# jq
$docker info and **$docker inspect** commands can produce output in JSON format. Combine these commands with jq processor.

Pretty JSON and jq Processing
# show whole Docker info
$ docker info --format "{{json .}}" | jq .

# show Plugins only
$ docker info --format "{{json .Plugins}}" | jq .

# list IP addresses for all containers connected to 'bridge' network
$ docker network inspect bridge -f '{{json .Containers}}' | jq '.[] | {cont: .Name, ip: .IPv4Address}'

$ docker network ls
```
NETWORK ID          NAME                DRIVER              SCOPE
d2e4aa487976        bridge              bridge              local
19cceec4933c        host                host                local
f8b0c93b7948        none                null                local
```

$ docker network inspect d2e4aa487976

Get Docker Container's IP Address from Host Machine

Get IP Address of a Single Docker Container
The command can be used to get Docker container's IP address is docker inspect with some options:
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' container_id_or_name

Let's see an example which we will get IP address of a Docker container which has ID 4a432e03f0c3:
$ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 4a432e03f0c3
172.17.0.2

With old version of Docker, we can use the following syntax:
docker inspect --format '{{ .NetworkSettings.IPAddress }}' container_id_or_name

Let's see an example with container id:
$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' 4a432e03f0c3
172.17.0.2

Get IP Addresses of All Docker Containers

We can tune the above commands to get IP addresses of all Docker containers in just one single command:
docker inspect -f '{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)

Here is the output on my environment:

$ docker inspect -f '{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)
```
/static-web - 172.17.0.3
training/sinatra - 172.17.0.2
training/names
```

With old version of Docker, we can use the following syntax:
$ docker inspect -f '{{.Name}} - {{.NetworkSettings.IPAddress }}' $(docker ps -aq)
The output can be:
$ docker inspect -f '{{.Name}} - {{.NetworkSettings.IPAddress }}' $(docker ps -aq)

```
/wwwlogs -
/wwwdata -
/admiring_montalcini - 172.17.0.11
/mywebi - 172.17.0.10
/myweb - 172.17.0.9
/mcweb - 172.17.0.8
/elegant_hypatia - 172.17.0.7
/vigilant_hypatia - 172.17.0.6
/zweb - 172.17.0.5
/webz - 172.17.0.4
/web - 172.17.0.3
/laughing_banach - 172.17.0.2
```

Another command that can be used to get Docker IP address is $docker container inspect. In similar to the docker inspect command, it is mainly used for displaying detailed information on one or more containers. However, we can leverage it to get Docker IP address.

For example, let's get IP addresses of all Docker containers:
$ docker container inspect -f '{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)

Get IP Address of a Docker Container from Inside Container
Get in a Docker Container

To get IP address of a Docker container from inside, we can get in it first by using the docker exec command:
$ docker exec -it container_id_or_name bash

For example, let's get in a docker container which sas id: 4a432e03f0c3
$ docker exec -it 4a432e03f0c3 bash

Get IP Address of a Docker Container

After getting in the docker container, we can issue normal Linux command to get Docker container's IP address, for example:
$ ip addr
$ ip addr | grep global

```
    inet 10.0.0.4/24 brd 10.0.0.255 scope global eth0
    inet 172.17.0.1/16 scope global docker0
```

or
/sbin/ip route|awk '/default/ { print $3 }'

Watching Containers Lifecycle

Sometimes you want to see containers being activated and exited when you run certain docker commands or try different restart policies. The watch command combined with docker ps can be pretty useful.

$ docker stats command (even with --format option) doesn't allow you to see the same information as you can with the **$ docker ps** command.

Display a Table with 'ID Image Status' for Active Containers and Refresh it Every 2 Seconds

$ watch -n 2 'docker ps --format "table {{.ID}}\t {{.Image}}\t {{.Status}}"'

Enter into Host/Container Namespace

Sometimes you want to get some tool as a Docker image, but you do not want to search for a suitable image or to create a new Dockerfile (no need to keep it for future use, for example). Sometimes storing Docker image definition in a file looks like an overkill – **you need to decide how you edit, store and share this Dockerfile**. Sometimes it's better to have a single line command, that you can copy, share, embed into a shell script or create special command alias.

So, when you want to create a new ad-hoc container with a single command, try a Heredoc approach.
Create Alpine Based Container with 'htop' Tool

```
$ docker build -t htop - << EOF
FROM alpine
RUN apk --no-cache add htop
EOF
```

Network Tricks

There are times when you might want to create a new container and connect it to an existing network stack. This might be the Docker host network or another container's network. This is helpful when debugging and auditing network issues.

The **docker run --network/net** option allows you to do this.

Use the Docker Host Network Stack

```
$ docker run --net=host ...
```

The new container will attach to the same network interfaces as the Docker host.

Use Another Container's Network Stack

```
$ docker run --net=container:<name|id> ...
```

The new container will attach to the same network interfaces as the other container. The target container can be specified by id or name.

Attachable Overlay Network

Using Docker Engine running in swarm mode, you can create a multi-host overlay network on a manager node. When you create a new swarm service, you can attach it to the previously created overlay network.

Sometimes you need to attach a new Docker container (filled with different networking tools), to an existing overlay network, in order to inspect the network configuration or debug network issues. You can use the docker run command for this, eliminating the need to create a completely new "debug service".

Docker 1.13 brings a new option to the docker network create command: attachable. The attachable option enables manual container attachment.

```
# create an attachable overlay network
$ docker network create --driver overlay --attachable mynet
# create net-tools container and attach it to mynet overlay network
$ docker run -it --rm --net=mynet net-tools sh
```

Docker Command Completion

Docker CLI syntax is very rich and constantly growing: adding new commands and new options. It's hard to remember every possible command and option, so having a nice command completion for a terminal is a must have.

The command completion is a kind of terminal plugin, that lets you auto-complete or auto-suggest what to type in next by hitting the tab key. Docker command completion works for commands and options. The Docker team prepared command completion for the docker, docker-machine and docker-compose commands, for both Bash and Zsh shell.

If you are using Mac and Homebrew, then installing the Docker commands completion is pretty straight forward.

```
# Tap homebrew/completion to gain access to these
$ brew tap homebrew/completions
# Install completions for docker suite
$ brew install docker-completion
$ brew install docker-compose-completion
$ brew install docker-machine-completion
```

If you're not using Mac, read the official Docker documentation for install instructions: docker engine, docker-compose and docker-machine.

Start Containers Automatically

When running a process inside a Docker container, a failure may occur due to multiple reasons. In some cases, you can fix it by re-running the failed container. If you are using a Docker orchestration engine, like Swarm or Kubernetes, the failed service will be restarted automatically.

If not, then you might want to restart the container based on the exit code of the container's main process, or always restart (regardless the exit code). Docker 1.12 introduced the docker run command: restart for this use case.

Restart Always

Restart the redis container with a restart policy of always so that if the container exits, Docker will restart it.

```
$ docker run --restart=always redis
```

Restart Container on Failure

Restart the redis container with a restart policy of on-failure and a maximum restart count of 10.

```
$ docker run --restart=on-failure:10 redis
```