Narrative – Introduction to Docker

Slide 1 – Introduction

- General introduction
- Note I'm a JEE guy
- Work for American Electric Power
- Will be approaching this subject from an enterprise perspective

Slide 2 – It's all about the containers

- This is what people usually think of when they think of Docker – the container, specifically a shipping container (after all it's in the Docker logo)
- Containers have a uniform interface, doesn't matter what's inside
  - JEE applications
  - Rails
  - Node.js
- Containers have many transports – environments in which they can be deployed
  - Linux (native)
  - OS X, Windows, BSD (with help)

Slide 3 – It's not all about the containers

- Containers are for the DevOps folks

Slide 4 – It's about the stuff inside the containers

- Developers & Architects care about the applications
- Containers are the means for packaging, deploying and running our applications
- We control what goes inside the container – the image
- We'll be discussing images later

Slide 5 – Don't care where our applications run

- Cloud landscape is continually changing
- Hosting considerations change over time
  - Evolve with business
- We just want to easily package, deploy and run our applications

Slide 6 – Docker is the solution

- Package our applications
  - Including dependencies
    - Shared libraries

    - Configuration
        - Environment variables
        - IP ports
- Deploy & run our packaged applications anywhere
- Docker abstracts infrastructure
    - Doesn't care about virtual machines
    - Not a virtual machine replacement, more on that later

Slide 7 – Docker overview

- The Docker objects
    - Docker registry
        - This is where Docker images are stored
        - You can create private repositories too
        - DockerHub is public Docker repository
            - Anyone can pull any public image from the Docker repository
    - Local image cache
        - Images must be in the local image cache to be run
    - Local containers
        - Images running in local Docker instance
    - Dockerfiles
        - Used to create Docker images
- The Docker object relationship arrow depicts the Docker command and its related objects
    - Examples
        - Running a Docker image produces a Docker containers
        - Building a Docker buildfile produces a Docker image
- This diagram contains nearly all the Docker commands

Slide 8 – Running busybox

- I like to run the busybox image to confirm my local Docker installation is running properly
- Parsing the command line
    - docker – this is how you invoke the Docker client
    - run – the Docker command to run an image
        - the image will be pulled down from the repository if it's not found locally
    - busybox – the name of the Docker image to be run
        - a new Docker container will be created in which the busybox image will be run
    - cat /etc/os-release – the command to be run inside the newly-created container

running the busybox image
- Because the cat command exits, the container exits at this point
  - A container that exits cannot be restarted – exit is a terminal state
- Demo Docker at this point
  - docker run busybox cat /etc/os-release
    - note that Docker runs
  - docker ps -a
    - note the status of the container
  - Discuss that some images have default commands that will be run when the image is run others don't

Slide 9 – What are images?
- Docker images contain the following:
  - OS core and shared libs
    - glibc
  - Application runtime environments
    - Node.js
    - Apache
    - Nginx
    - WebLogic
  - Application configuration
    - Environment variables
    - IP ports to open
  - Application artifacts
  - Images also contain whatever files are need for the proper running of the application

Slide 9 – Images & containers
- Think of Docker images as being like classes and Docker containers being like objects
- When programming you use the new operator to create an object from a class, with Docker you use the run command to create a container for an image
- Like objects, containers are runtime artifacts
- Images are created using docker build
  - Like classes, you tend not to create a lot of images and image creation is kind of special
  - Likes classes, you'll tend to create a lot of instances of images
- Containers are created using docker run
  - Like objects are created from classes, containers are created from images
  - Containers are always named

        - By default is a name isn't specified
            - A rather whimsical name
        - Explicitly named using –name option
        - Every container has an id
            - A container can be referred to by name or by id
- The set of currently running containers are obtained using docker ps
    - Demonstrate docker ps command
        - docker ps -a shows all running images, including those that have exited
        - we see the name of the image running the container
        - we see the command running
        - we see when the container was created
        - we see the status (Exited, Running, Stopped)
        - we see the IP ports opened by the container, if any
        - we see the name of the container
            - a silly but unique name is assigned if not specified when creating the container
    - Demonstrate removing a container
        - docker rm `docker ps -aqf exited=0`
        - this removes all Docker images that have successfully exited
        - the current version of Docker only allows you to filter on command exit status
        - the -q is the quiet option, returning only the container id
            - which is what we need to remove a container
- The set of currently cached images are obtained using docker images
    - Demonstrate listing images
        - docker images
            - images having the same image id are the same image
            - the same image can have multiple tags
- Demonstrate how to search for images
    - docker search centos
    - docker search nginx
    - docker search wls
    - docker search derby

Slide 10 – Creating Docker images

- Left brain strategy
    - Use a dockerfile
    - docker build
    - This is a repeatable process

        - Anyone can use the dockerfile to build the image
    - The dockerfile can be versioned in your source control management system
- Right brain strategy
    - Create using existing image
    - Run image
    - Modify however needed
    - Commit the container into a new image
        - You cannot modify an already-existing image with this technique

Slide 11 – Dockerfile

- left brain approach
- '+', '*' modifers are like regex
    - '+' means 0 or at most 1
    - '*' means 0 or many
    - the modifier isn't part of the command, it's telling you how many times the command may appear in a dockerfile
- common commands
    - used for building most Docker images
    - FROM tells you the base image, it's like specifying the base class
        - this image inherits everything from the base image
    - ADD files to the image, you can add tarballs too and they're automatically untarred
        - you can also add from URLs similarly to wget or curl
    - ENV is used for adding environment variables to the image. They can be referred to by subsequent commands in the dockerfile.
    - EXPOSE is used to expose IP ports to the containers
    - CMD specifies the command to run by default if not specified using docker run
        - if a command is specified using docker run then it overrides the CMD specified in the dockerfile
- less common commands
    - RUN specified command using the image as existing up to the point the RUN command is encountered in a new container, committing the result in a new image to be used for the next dockerfile commands. This is a good time to run groupadd, useradd or usermod commands if needed.
    - WORKDIR the most recent WORKDIR prior to RUN determines the working directory from which the command will be run. The most recent WORKDIR prior to CMD determines the working directory from which the command will be run.
    - USER the most recent USER prior to RUN determines the user under which

the command is run. The most recent USER prior to CMD determines the user under which the command is run.
    - COPY isn't as useful as ADD, generally I prefer using ADD
    - ENTRYPOINT allows you to specify a new entrypoint into the image, which by default is /bin/sh -c. The command that gets run then is ENTRYPOINT CMD. For example, if CMD were bash then the final command executed to launch the container would be /bin/sh -c bash.
- Dockerfile processing
    - When running a dockerfile command the image up to that point is run in a new container, the command is run, the container is committed back to a new temporary image from which the next dockerfile command will be run
    - The temporary containers are removed by default upon a successful build
        - This option can be changed
        - Makes building similar Docker images very fast

Slide 12 – Building the image

- cd to directory containing Dockerfile file
- docker build -t <image name:version> .
- If successful the image will be named <image name>
    - By default the version will be 'latest' if not specified
    - Image names are kinda tricky in Docker – they include the repository location. That way there's no ambiguity concerning the origin of an image: it's name includes the repository location.
- Show examples of Dockerfiles

Slide 13 – Right brained way

- To make things easy remove all the containers that have successfully exited
    - docker rm `docker ps -aqf exited=0`
    - this allows us to easily find our modified container when we're finished
- docker run the image you wish to use as your base, running bash
    - this is similar to FROM if you were using Dockerfile
- Make changes as you see fit, installing and configuring your applications
- Exit your container using exit from bash
- The container still exists even though you've exited, docker ps -af exited=0 will show the info about this container
- User docker commit to create a new image from the image used in this container
- docker commit `docker ps -aqf exited=0` image name:version
- That's it!
- Problem is you can't repeat this process, but it's very good technique for developing your Dockerfile

- Demo using right-brained technique
  - docker rm `docker ps -aqf exited=0`
  - docker run -i -t busybox
  - echo export TEST=true > ~/.profile
  - cat ~/.profile
  - exit
  - docker ps -af exited=0
    - note our container still exists
  - docker commit `docker ps -aqf exited=0` test
  - docker images
    - note 'test' was just created a few seconds ago
  - docker run -i -t –entrypoint=/bin/sh test -l
  - env
    - note TEST=true
    - you've successfully made a change to an existing image and committed it to a new image
    - cleanup:
      - docker rmi -f test
      - docker rm `docker ps -aq`

Slide 14 – IP port forwarding

- Use the -p local-port:container-port option for the Docker run command
  - left hand is local IP port, right hand is container's exposed IP port
- We don't have to reconfigure software to run on different ports
  - Less chance to mess things up
- Demo
  - Run a couple instance of taylodl/wls-mydomain on ports 7001 & 8001
  - Open browser and navigate to localhost:7001/console
    - Open Environment/Servers and note the Listen Port
    - Also note Deployments is empty
  - Open browser and navigate to localhost:8001/console
    - Open Environment/Servers and note the Listen Port
    - Also note Deployments is empty
  - docker ps
    - See our two containers running
    - Also look at the PORTS
      - We can tell which container is which by looking at the PORTS
  - docker pause
    - Pause one of the taylodl/wls-mydomain containers

        - docker ps
            - The status now says Paused
        - Note the other container isn't affected
        - Try to reach the admin console of the paused container
            - Note it times out
        - Unpause the container
        - docker ps
            - Status returns to normal
        - Resume using the admin console of the unpaused container
        - Stop all running containers
            - docker stop `docker ps -q`
            - docker ps returns an empty list
            - docker ps -a shows our taylodl/wls-mydomain containers
        - Restart all containers that were running
            - docker start `docker ps -aq`
            - Try accessing the admin console of one of the taylodl/wls-mydomain containers
        - Cleanup
            - Stop all running containers
                - docker stop `docker ps -q`
                - docker ps -a to confirm both containers have stopped
            - Remove all containers
                - docker rm `docker ps -aq`
                - docker ps -a to confirm all containers have been removed

Slide 15 – Volume mapping

- Use the -v local-volume:container-volume option for the Docker run command
    - left hand is local directory, right hand is where it will be mounted in the container
        - The directory doesn't have to exist in the image
            - If the directory does exist in the image its contents will be replaced
- Demo
    - run taylodl/wls-mydomain
        - docker run -p 7001:7001 -v /home/developer/docker/wls-autodeploy:/appl/oracle/middleware/wls/wls12120/user_projects/domains/mydomain/autodeploy taylodl/wls-mydomain
    - Open browser and navigate to localhost:7001/console
        - Note deployments present and active
    - docker inspect `docker ps -q`

        - Note all the information we can obtain about a running container
        - Especially note
            - IP port mappings
            - volume mappings

Slide 16 – More to explore

- Coordinating multiple Docker containers
    - Application server & database, for example
    - Fig
- Private repository
    - Many enterprises have a policy where they can't store images in the could
- Debugging
    - Attach
    - Exec
    - Logs