Kevin Chen

7/10/17

Docker Notes

**Introduction**

What is Docker?

* Docker:
  + Carves up a computer into sealed, separate containers that run your code.
  + Gets the code to and from your computers.
  + Builds the containers
  + A social platform for you to find and share containers
  + It is NOT virtual machines…there is a single OS running, but it’s curved up into isolated little spaces.
* Containers:
  + A self-contained sealed unit of software.
  + Contains everything required to run the code. Includes: code, configs, processes, networking, dependencies, and some of the OS to run the code.
  + It takes all the services that make up a Linux server and makes a copy of that in the Linux kernel for each container.
  + Each container has its own little world that it can’t see out of and other containers can’t see in.
  + It doesn’t matter which Linux each container is running on; Docker manages this.
* Docker is a client program. It’s also a server program that d manages a Linux system. It builds containers from code. A service that distributes containers.

**Installing Docker**

Docker Toolbox

* Docker needs a Linux server to manage. Can use a Linux Virtual Machine, which Docker Toolbox can help manage.
* Docker Toolbox will install Docker and VirtualBox (if it doesn’t already exist). It then will install Docker Machine, which manages a Linux VM. Inside the virtual machine is the server side of Docker. When you type Docker at the command prompt, it will send the command into the Linux VM over the network to the Docker server running there.
* How to use Docker
  + Run Docker Quickstart Terminal to start up Docker. This brings up a terminal configured to connect to your VM. Now you can run Docker commands.
  + “docker-machine start” starts the machine
  + “docker-machine stop” stops the machine.
  + “docker-machine ip” prints out the IP address of the machine it is connected to.
  + “docker-machine scp” copies files into and out of your VM.
  + “docker-machine upgrade” installs the latest version of Docker.
  + “docker-machine restart” restarts Docker.

Installing Docker on Windows

* (Refer to online resources or the Lynda course to install Docker on other OS)
* Virtualization needs to be enabled.
* Install the Docker Toolbox.
* Install Netcat using the command: docker run –net=host -ti ubuntu:14.04 bash
* Run lc -lp 1234 to tell it to start listening on a port 1234.
  + Press Ctrl + C or Ctrl + Break to get back to the prompt

**Using Docker**

The Docker Flow: Images to Containers

* Begin with an image: every file that makes up just enough of the OS to do what you need to. Much more efficient than installing a whole OS.
* “docker images” takes a look at your Docker images.
  + Repository for the image is where it came from.
  + Tag is the version of the image.
  + Image ID is the internal Docker representation of the image.
* You can refer to any image using either combining the repository and tag, separating them by a colon, or the image ID
  + If you don’t specify a tag, Docker will by default assume it is “latest”
* “docker run *image command*” takes the *image* and turns it into a running container and does the specified *command*.
  + -ti option (terminal interactive) causes it to have a full terminal within the image so you run the shell. Useful for running commands inside the image via keyboard.
  + Type “exit” or Ctrl + D to exit the image.
* “docker ps” lets you take a look at the running containers.
  + Has an ID. This is different from the corresponding image ID.
  + If you don’t give the container a name, Docker gives you a creative one.
* Any changes you make to the running container does not change the image.

The Docker flow: Containers to images

* When you exit from a running container, it becomes a stopped container.
* “docker -ps -a” will list all containers, so you can see stopped ones as well.
* You can open up the last container that exited with “docker -ps -l”.
* Can see its ID, status (contains exit code, useful for debugging), ports (for networking), and its name (which Docker creates if not specified).
* You can create a new image from a stopped container using the command “docker commit *containerID*”
  + This returns an Image ID. But this image has no name.
  + It does not overwrite the image that the container was made from.
* “docker tag *imageID* *name*” assigns a *name* to *imageID*.
* This command creates an image out of a stopped container AND assigns the image a name: “docker commit *containerName* *imageName*”

Run processes in containers

* Containers have a main process. “docker run” starts a container and its main process.
* The container stops when the main process stops.
* Containers have names. If you don’t give it one, Docker will make one up.
* “docker run --rm” throws away the container after you stop it.
* Ubuntu Commands:
  + “sleep *n*” will cause the container to sit there for *n* seconds after opening and then exit.
  + “bash -c “*command1*; *command2*; …; *commandN*”” runs each command separated by semicolons on the command line.
* Detached containers
  + Adding the “-d” option for “docker run” starts the container as a detached container. This runs in the background.
  + You can then enter a detached container by typing “docker attach *containerName*”
  + You can then exit that detached container but leave it running by using the key combination Ctrl + P followed by Ctrl + Q.
* Running more processes in a container
  + “docker exec *containerName*” starts another process in an existing container.
  + Can’t be used to add ports, volumes, or any of the “fancy stuff”.
  + If the terminal exits a container, then this container will also be stopped inside any other container it is in.

Manage containers

* Docker logs
  + Docker keeps the output of a container to help you debug. Access the logs of a particular container with the command “docker logs *containerName*”
  + “docker run --name *name*” assigns *name* to the running container.
  + Don’t let the output get too large, or else the system will become slow/unresponsive.
* Killing and removing containers
  + “docker kill *containerName*” kills the container
  + “docker rm *containerName*” removes the container
* Resource constraints
  + Add the tag “--memory *maxAllowedMemory*” to “docker run” to restrict memory.
  + Add the tag “--cpu-shares *share*” to “docker run” to restrict CPU usage relative to other containers. “--cpu-quota *limit*” restricts it to a general limit (e.g. 10%)
  + Orchestration generally requires resource limiting
* Some advice
  + Don’t let containers fetch dependencies when they start. Make your containers include the dependencies inside the container themselves.
  + Don’t leave important things in unnamed stopped containers. You tend to delete these pretty quickly.

Network between containers

* Programs in containers are isolated from the Internet by default.
* Docker provides a private network for use by the containers.
* There are multiple private networks, so you can group containers into private networks so that unrelated services don’t interfere with each other.
* You set which containers can talk to which containers on whichever ports by exposing ports and linking containers.
* Docker helps you find other exposed ports with compose services.
* Can explicitly specify the port on the inside of the container to a particular port outside of the container. Can expose as many ports as you want per container.
* To expose a port, add the “-p *insidePort*:*outsidePort*” tag to “docker run” for each port you want to expose.
* Some netcat commands:
  + “nc -lp *portNumber*” tells Netcat to listen on that port and echo whatever it says.
  + The pipe operator “|” forwards the data to another copy of Netcat. Syntactically, the pipe operator goes in between the two Netcat commands.
  + Use the command “nc *ipAddress* *portNumber*” to get Netcat to connect to the container, where *ipAddress* is the IP Address of the Linux VM (found using the command “docker-machine ip”) and *portNumber* is the port exposed outside of the container.
* Exposing ports dynamically
  + The port inside the container is fixed.
  + Get rid of *outsidePort* in “-p *insidePort*:*outsidePort*” tag to expose outside port dynamically.
  + “docker port *containerName*” shows the ports that the container was dynamically exposed to.
  + This allows many containers running programs with fixed ports without conflicting each other. This often is used with a service discovery program.
* Exposing UDP ports
  + By default, docker assumes you are using tcp connection. If you want to use UDP, use “docker run -p *outsidePort*:*insidePort*/*protocol*”
  + Example: “docker run -p 25124:25124/udp”

Link Containers

* Approach #1: expose both containers externally on the same host port.
* Approach #2: link containers so data directly goes between the containers while staying within Docker.
  + Generally used with orchestration
  + Links all ports, though only one way
  + Only for services that cannot ever be run on different machines
  + Don’t need to do anything special for the receiving container. The sending container needs the additional tag “--link *containerName*” to directly transmit data to the container with the name *containerName*.
  + Docker automatically assigns a hostname to the server (receiving container).
  + Docker links the IP Address with the *containerName* in /etc/hosts.
  + Note that links can break when containers restart.

Dynamic and legacy linking