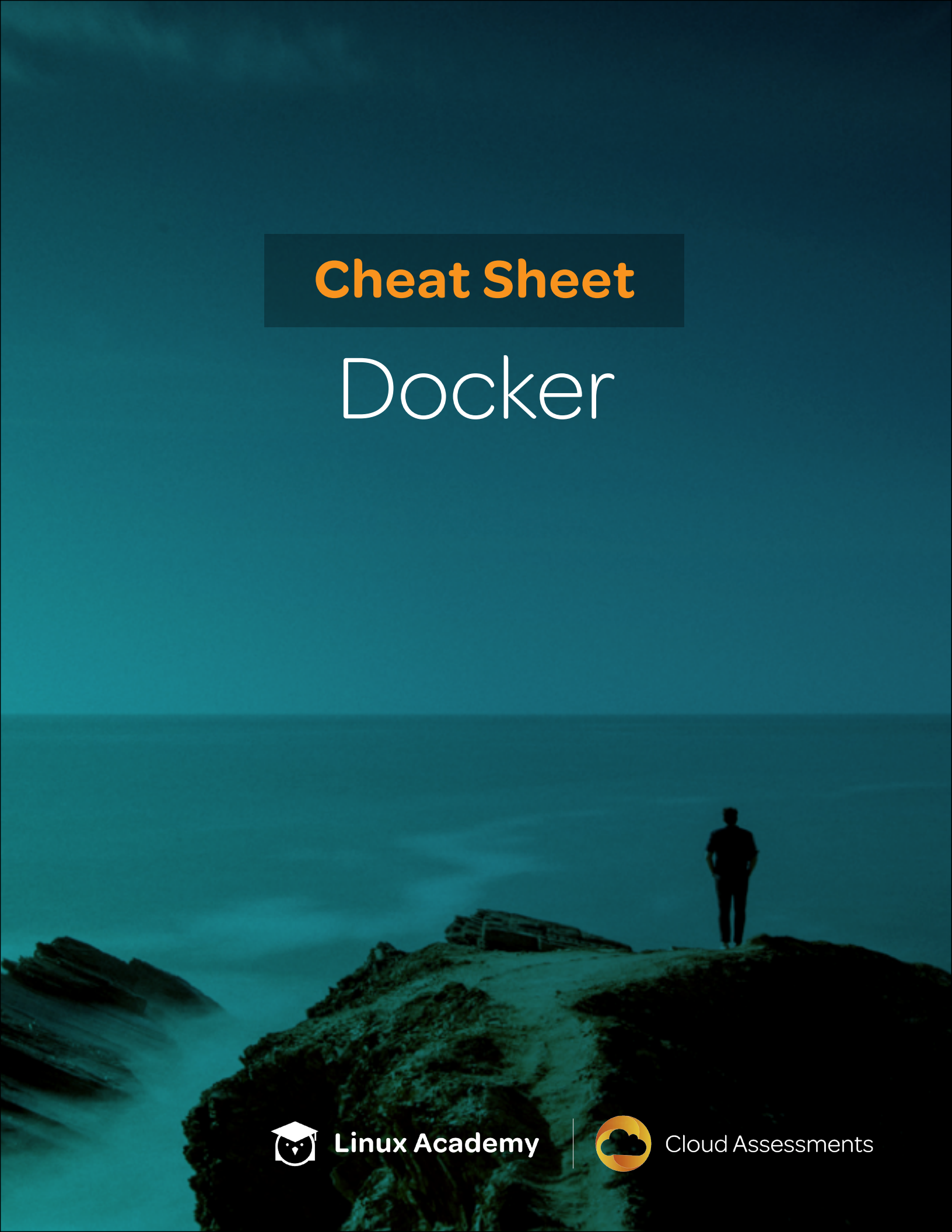
Cheat Sheet

Docker



Linux Academy | Cloud Assessments

Contents

Docker Engine Installation

- https://docs.docker.com/engine/installation/linux/docker-ce/centos/#upgrade-docker-ce-1
- CentOS/Red Hat process
  - Make sure no other/older versions of Docker are installed on the system
    - if so, `/var/lib/docker` images, containers, volumes and networks will be preserved
  - `sudo yum install -y yum-utils device-mapper-persistent-data lvm2`
  - `sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo`
    - Optionally, use `sudo yum-config-manager --enable docker-ce-edge` or `docker-ce-test`
    - Disable with `--disable`
  - `sudo yum install docker-ce`
  - `sudo systemctl enable docker && systemctl start docker && systemctl status docker`
  - NOTE: installing/upgrading covered separately from packages
  - Add a user for non-root use of Docker
    - `sudo usermod -aG user docker`
    - Restart Docker, check `/var/run/docker.sock`
    - Will need to log out and back in, then check with 'docker images'
- https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/#upgrade-docker-after-using-the-convenience-script
- Debian / Ubuntu process
  - `sudo apt-get install apt-transport-https ca-certificates curl software-properties-common`

- `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
- `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`
- `sudo apt-get update`
- `sudo apt-get install docker-ce (be careful to use docker-ce)`
  - If you need a specific version, you can do `sudo apt-get install docker-ce=[VERSION]`
- `sudo systemctl enable docker && systemctl start docker && systemctl status docker`
- NOTE: Installing/upgrading covered separately from packages
- Add a user for non-root use of Docker
  - `sudo usermod -aG user docker`
  - Restart Docker, check `/var/run/docker.sock`
  - Will need to log out and back in, then check with 'docker images'

Pull an Image from a Registry (Using Docker Pull and Docker Images)

- `docker pull`
  - `--all-tags (-a)` - download ALL tagged images in the configured repository
  - `--disable-content-trust` - skip image verification
  - HTTP_PROXY/HTTPS_PROXY - can be used in the Docker daemon configuration if needed to work with proxies for pulling images
    - `/usr/lib/systemd/system/docker.service`
      - configuration file to add those variables to
      - after updating - `systemctl daemon-reload` and then restart Docker

- `--max-concurrent-downloads [#]` - option to increase/decrease the number of layers pulled at any one time
    - passed to the Docker daemon at startup
- `docker images`
    - displays the images installed locally
    - `--all (-a)` - display all images
    - `--digests` - display digests
    - `--filter (-f) [name=value]` - display the image indicated within the filter
        - `label`
        - `dangling`
        - `before`
            - for example - `docker images --filter "before=[imagename]`
                - will display images created before indicated image name
        - since
        - reference
            - for example - 'docker images --filter=reference='[referenceimage]''
                - will display images meeting the indicated reference
    - `--no-trunc` - do not truncate output
    - `--quiet (-q)` - only show numeric IDs
        - useful to feed other docker commands typically taking one argument
            - `docker rm \docker ps -a -q\\` (removes all previously run but stopped containers)

Utilize Search in a Registry

- `docker search`
  - `--filter (-f) [conditions]` - filter the output as indicated
    - `stars=[#]` - number of stars an image has
    - `is-automated [true/false]` - is the image automated or not
    - `is-official [true/false]` - is the image an official image or not
      - for example - `docker search --filter is-official=true [search term]`
  - `--format` - uses Go template for formatted search
  - `--limit [#]` - maximum number of results to display
  - `--no-trunc` - do not truncate output

Tag an Image

- `docker tag`
  - `SOURCE[:TAG] TARGET[:TAG]`
    - tag by reference ID
    - tag by name
    - tag by name AND tag
    - to for a private repository
      - for example `docker tag [local image:tag] [registryhostname:port/registry/name:tag]`

Use CLI Commands to Manage Images (List, Delete, Prune, RMI, etc)

- `docker image`
  - `build`

    - build from Dockerfile (see Dockerfile)
- `history`
    - history of the indicated image
- `import`
    - tarball name as an argument
- `inspect`
    - options - https://docs.docker.com/engine/reference/commandline/inspect/#options
    - alone, provides JSON formatted output of every aspect of the indicated image
    - `--format (-f)` - special formatting using Go Template
    - see next video on filter and format
- `load`
    - standard input
        - `docker load < name.of.tar` (or `docker load --input name.of.tar`)
- `ls`
    - list images (same options as 'docker images')
- `prune`
    - -a (removes all images without at least one container associated)
- `pull`
    - repository pull (see earlier)
- `push`
    - repository push (see using repositories)
- `rm`
    - same options as `docker rm`

        - removes one or more indicated images
    - `save`
        - name of tar to save to
    - `tag`
        - same options as `docker tag`

Inspect Images and Report Specific Attributes Using Filter and Format

- `docker image inspect`
    - `[imagename]`
        - `docker image inspect [image] --format '{{.RepoTags}}'`
    - `--format '{{[range] .structure.substructure}} [{{end}}]'`
        - special JSON formatted output for entire substructure
            - json (keyword before range or structure)
            - `docker image inspect [image] --format '{{json .ContainerConfig}}'`

Container Basics - Running, Attaching to and Executing Commands In Containers

- `docker run`
    - `-i` - interactive - STDIN
    - `-t` - terminal/pseudo TTY
    - `-d` - detached
    - `--name [nametoassign]` - assign a name to the container, must be unique

    - `--ip [address]` - assign a valid IP on the default or indicated network
    - `--network [network name]` - connect container to indicated network
    - `--priviliged` - give privileged access to container - i.e. access to `/dev`
    - `--publish (-p) [#[:#]]` - publish container port to indicated host port - local:container
    - `--publish-all (-P)` - publish all container ports to the host port
    - `--rm` - removes container on exit
    - `--volume (-v) [local[:container]] -` - binds indicated mount point
    - `--env (-e) [name=value] - assigns variables and their values, can set multiple --env` on command line
    - `--mount type=[volume/bind,src=[path],dst=[path] OR target=[containerpath]]`
- `docker exec`
    - `-i` - interactive - STDIN
    - `-t` - terminal/pseudo TTY
    - `-d` - detached

Create an Image with a Dockerfile

- https://docs.docker.com/engine/reference/commandline/build/#options
- `docker build`
    - large number of options
    - most commonly used:
        - `.` - represents the current build context
        - `-f (--file) [PATH/DockerfileName]`
            - `docker build -f [/path/Dockerfile] -t [image[:tag]] .`

        - builds the indicated Dockerfile with the indicated tag in the current build context
    - `-m (--memory) [##]` - set memory limit of image
    - `--no-cache` - do not use cache when building the image
    - `--pull` - always attempt to pull new version of image
    - `--rm` - remove intermediate containers after successful build
    - `--squash` - squash new layers into single new layer
    - `--build-arg [var=value]`
        - `docker build --build-arg HTTP_PROXY=http://localhost:3939 .`
    - `--add-host=[hostname:IP]`
    - `-t (--tag)` - name and (optionally) tag an image (name:tag)
- `build contexts`
    - `--target [stage name]`
        - `docker build -t myimage --target [FROM/AS name in Dockerfile]`
    - git repository
        - `docker build https://github.com/path/repo.git[#branch[:directory]]`
    - tarball
        - `docker build https://server/context.tar.gz`
    - text (Dockerfile)
        - `docker build - < Dockerfile`
        - `docker build . -t name[/version][:tag]` - looks for Dockerfile in current directory context)

Dockerfile Options, Structure, and Efficiencies

(Part I and Part II)

- https://docs.docker.com/engine/reference/builder/#from
- `FROM`
  - `<image> [AS <name>]`
  - `<image>[:<tag>] [AS <name>]`
- `FROM` and `ARG`
  - `ARG SOME_VARIABLE=value`
    - NOTE: `ARG` is the only instruction that can precede `FROM` in Dockerfile
    - `FROM base:${SOME_VARIABLE}`
    - `ARG` before `FROM`, only used by `FROM`
    - `ARG` after `FROM` can be used anywhere in Dockerfile
- `RUN`
  - `<command>` - shell form - run in a shell `/bin/sh -c`
  - `["command", "parm1", "parm2"]` - exec form - does not require a shell executable, avoid shell string munging, parsed as JSON array
    - SHELL variable can be set to a different shell for the Dockerfile
  - commands can be run over multiple lines by appending a backslash (`\`) to the end of the line and continuing on the next
- `CMD`
  - `["command","parm1","parm2"]` - exec form, default and preferred
  - `["parm1","parm2"]` - provides default parameters to ENTRYPOINT
  - `command parm1 parm2` - shell form in `/bin/sh -c` by default
    - ONLY ONE CMD per Dockerfile (if more than one in the file, only last one will run)

        - provides the defaults for container execution (generally an executable, but if not an executable, provide an ENTRYPOINT)
        - NOTE: RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages. CMD sets default command and/or parameters, which can be overwritten from command line when the Docker container runs.
        - ENTRYPOINT configures a container that will run as an executable.
- LABEL
    - metadata information added to the image
    - key/value pair
    - no spaces in values unless quoted
        - `LABEL "com.sample.company"="My Sample Company"`
        - can span multiple lines with backslash (`\`)
    - when adding multiple labels in a Dockerfile, combine into one longer line with one LABEL, otherwise, each LABEL produces a new layer for the image
    - remember, they can still span multiple lines
- MAINTAINER (Deprecated)
    - `<name>`
    - `LABEL maintainer="name"` - use this format instead
- EXPOSE
    - `<port> [<port>/<protocol>...]`
    - protocol can be UDP or TCP associated with the indicated port, default is TCP with no specification
    - NOTE: does not publish the port, simply makes it available to be published and documents requirements of your image
- ENV

    - sets environment variables from <key> to <value>
    - `<key> <value>` (or `<key≧<value>`)
        - will be replaced inline when image is built
    - can appear as many times as needed
    - when adding multiple environment variables in a file, putting them all on one longer line will produce a single layer whereas multiple `ENV` statements produce a layer for each
    - NOTE: `ENV` variables persist and can be used in containers instantiated on the built image
        - can be changed at run time with `docker run --env <key≧<value>`
- `ADD`
    - copies files, directories, or URLs as indicated into the image filesystem in the path provided
    - `<src>… <dest>`
    - enclose in quotes for paths with spaces
    - wildcards are permitted
    - `WORKDIR` is where relative pathing starts, absolute directories are recommended
    - if using protected URLs that require auth, use `RUN wget` or `curl` as `ADD` does not support auth
        - NOTE: if the source is a tar archive in a recognized compression format, it will be unpacked in the destination path
        - NOTE: any other type of file will be copied, with metadata, to the indicated destination
        - NOTE: be sure the destination, when a directory, has a trailing slash (`/`) OR it will be treated as a destination file and be written as such
- `COPY`
    - copies files or directories as indicated into the image filesystem in the path provided

    - `<src>… <dest>`
    - enclose in quotes for paths with spaces
    - wildcards are permitted
    - `WORKDIR` is where relative pathing starts, absolute directories are recommended
    - `--from=<name|index>`
        - allows you to specify a file or directory (source) from a previous build stage
    - contents of directories are copied to the destination path, NOT the directory itself
        - NOTE: any other type of file will be copied, with metadata, to the indicated destination
        - NOTE: be sure the destination, when a directory, has a trailing slash (`//`) OR it will be treated as a destination file and be written as such
- `ENTRYPOINT`
    - `["command", "parm1", "parm2"]` - exec form, preferred
    - `command parm1 parm2` - shell form
    - creates a container that, when instantiated, effectively runs as an executable with the item in the `ENTRYPOINT`
    - NOTE: when used with `CMD` to provide parms, `CMD` must be AFTER the `ENTRYPOINT`
        - can be overriden at run time with `--entrypoint as part of the docker run` command
        - exec form parsed as JSON array, does not invoke a shell in exec form
    - NOTE: to ignore parms in `CMD` and to be sure `docker stop` cleanly stops the container, begin your `ENTRYPOINT` with `exec`
- `VOLUME`
    - `["/path"]`
    - creates a mount point with the indicated name and indicates it is an external mount point when instantiated from the image by a container

    - host directories are declared at run time and NOT within the Dockerfile
    - an underlying host directory is not guaranteed to be available on every host and images/containers should be portable
- `USER`
    - `<user>[:<group>]` or `<UID>[:<GID>]`
    - set username (UID) and (optionally) the GID to use when running the image and to be used with `RUN`, `CMD`, or `ENTRYPOINT` instructions in the Dockerfile
- `WORKDIR`
    - `/path/dir`
    - set the working directory for any `RUN`, `CMD`, `COPY`, `ADD`, or `ENTRYPOINT` instruction in the Dockerfile
    - NOTE: can be used/set/reset multiple times as desired
- `ARG`
    - `<name>[<default value>]`
    - defines arguments that can be passed in by user at build time using `docker build`
        - i.e., a user to run command as, a working directory, file name to use, changeable parms, etc.
    - predefined ARGs
        - HTTP_PROXY
        - HTTPS_PROXY
        - FTP_PROXY
        - NO_PROXY
    - example - `docker build -t --build-arg HTTP_PROXY=<value> .`
    - can cause additional layers to be built when the `ARG` is different on subsequent builds
- `STOPSIGNAL`

    - the specific signal (by name) to send to containers when `docker stop <name>` is issued

- `SHELL`
    - `["executable", "parms"]`
    - overrides the `/bin/sh -c` behavior for shell-based instructions in `RUN`, `CMD`, etc.
- SAMPLES IN DOCUMENTATION FOR REVIEW - MAKE SLIGHT REVISIONS

Describe and Display How Image Layers Work

- `layers`
    - `docker image history`
        - shows all layers and the history of commands on an image
    - `--no-trunc`
        - display the full command(s) run in that layer
    - `union file system`
        - allows files and directories of separate file systems (branches) to be overlaid to form a single file system

Flatten an Image to a Single Storage Layer

- flattening (or collapsing an image to save space) is not possible; however, you CAN flatten a container and then make it an image
- `docker run [image]`
    - run the image (i.e. couchbase)
    - export the container
        - `docker export [containerid] > container.tar`
    - import the container (to flatten the image, losing the history in the process)

        - `docker import [tarfile] [newimage:tag]`
        - should be smaller size and down to one layer
            - verify with `docker image history [newimage:tag]`
    - UNOFFICIAL: `docker-squash`
        - https://github.com/jwilder/docker-squash

Selecting a Docker Storage Driver

- normally, Docker volumes will be used to write data to
- using a 'pluggable' architecture, Docker supports multiple storage drivers that control how images and containers are stored and managed on your Docker host
- Docker EE (check the Product Compatibility Matrix https://success.docker.com/Policies/Compatibility_Matrix)
- Docker CE
    - Ubuntu (aufs, devicemapper, overlay2, overlay, zfs, vfs)
    - Debian (aufs, devicemapper, overlay2, overlay, vfs)
    - CentOS (devicemapper, vfs)
    - Fedora (devicemapper, overlay2, overlay, vfs)
- `docker info`
    - will tell you everything about your Docker installation
    - `| grep "Storage Driver"` - will show you the storage driver in the active configuration
- `--storage-driver` - flag to manually set when docker daemon started
- `/etc/docker`
    - `daemon.json` (may not exist, can be created to change storage driver)
    - `{ "storage-driver": "devicemapper" }`

    - restart Docker and rerun `docker info | grep "Storage Driver"`

- NOTE: once a new storage driver is enabled, it will make any images previously downloaded using the previous driver unavailable (and changing back or deleting the `daemon.json` file will cause docker.service to fail)

Prepare for a Secure Docker Registry

- create directory for the certificates in the desired context
    - i.e. `mkdir /home/user/certs`
- create the certificates using 'openssl' tools
    - `openssl req -newkey rsa:4096 -nodes -sha256 -keyout certs/dockerrepo.key -x509 -days 365 -out certs/dockerrepo.crt -subj /CN=myregistrydomain.com`
- NOTE: for our demo, because of port restrictions, we will add local host `/etc/hosts` entries on several Docker hosts - the first being the host of the registry so that 'myregistrydomain.com' points to the private IP and the second on another Docker host that points /etc/hosts 'myregistrydomain.com' to the private IP of the system hosting the registry
- create the directories that are needed for the certificates
    - `/etc/docker/certs.d/myregistrydomain.com:5000` - yes, it must include the port that the registry is running on; if redirected, use that port instead)
    - copy the certificate needed for this registry to the new directory
        - `cp /home/user/certs/dockerrepo.crt /etc/docker/certs.d/myregistrydomain.com:5000/ca.crt`
        - if not copied as root, `sudo chmod root:root /etc/docker/certs.d/myregistrydomain.com:5000/ca.crt`
- create authentication directory for necessary users
    - i.e. `mkdir /home/user/auth`
- run the container needed to create the authentication information for the registry, passing in a user
    - `docker run --entrypoint htpasswd registry:2 -Bbn testuser testpassword`

```
> auth/htpasswd
```

- confirm that the certificate and key exist in the `~/certs` directory
- confirm that the htpasswd authentication file exists in the `~/users` directory

Deploy, Configure, Log In to, Push, Pull, and Delete an Image in a Registry

- pull the registry image (if needed)
  - `docker pull registry`
- run the container that allows the private registry to work on default port 5000 with the appropriate TLS and basic user authentication requirements
  - `docker run -d -p 5000:5000 -v \pwd\\/certs:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/dockerrepo.crt -e REGISTRY_HTTP_TLS_KEY=/certs/dockerrepo.key -v \pwd\\/auth:/auth -e REGISTRY_AUTH=htpasswd -e REGISTRY_AUTH_HTPASSWD_REALM="Registry Realm" -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd registry`
- local test
  - `docker pull busybox`
  - `docker tag busybox myregistrydomain.com:5000/my-busybox`
  - `docker push myregistrydomain.com:5000/my-busybox`
    - should FAIL, no authentication provided
  - `docker login myregistrydomain.com:5000/my-busybox`
    - now succeeds
  - `docker rmi busybox`
  - `docker rmi myregistrydomain.com:5000/my-busybox`
    - note that this only affects the repository on the host NOT the registry we set up
    - test with `docker pull myregistrydomain.com:5000/my-busybox`

        - should pull the image and be visible locally with ‘docker images’

- remote Docker host
    - `mkdir -p /etc/docker/certs.d/myregistrydomain.com:5000`
    - copy remote `ca.crt` to this directory (note that each host using the repository must have this file in the right directory)
        - in our demo, make sure `/etc/hosts` points to the remote myregistrydomain.com via the private IP
    - test
        - `docker pull myregistrydomain.com:5000/my-busybox`
            - FAILS, no authentication
        - `docker login myregistrydomain.com:5000`
        - `docker pull myregistrydomain.com:5000/my-busybox`

Managing Images in Your Private Repository

- can use either `curl` or `wget` to get a list of the repositories/images on the private registry (using the HTTP/API for Docker registry)
    - `wget --no-check-certificate --http-user=testuser --http-password=testpassword https://myregistrydomain.com:5000/v2/_catalog`
        - will download a file called ’_catalog’ with the values in it
    - `curl --insecure -u “testuser:testpassword” https://myregistrydomain.com:5000/v2/_catalog`
- for images tags, use the following format
    - `GET /v2/<name>/tags/list`
- for manifests (to get the digest needed to work with images and layers)
    - `curl --insecure -u “testuser:testpassword” https://myregistrydomain.com:5000/v2/my-busybox/manifests/latest`

- deleting images can be done with
  - `DELETE /v2/<name>/manifests/<reference>`
    - where <reference> must be the digest
- registry HTTP API v2

Configure Logging Drivers (Splunk, Journald, etc)

- determined by the value in the JSON file `/etc/docker/daemon.json`
  - minimal configuration (with options)
    - `{ “log-driver”: “syslog” “log-opts”: { “labels”: “production_log”, “env”: “os,customer” } }`
  - unspecified log driver is ‘json-file’ logging driver
  - `docker info | grep “Logging Driver”`
- container log driver (customize)
  - `docker run -[options] --log-driver [driver] [image] [command]`
- find logging for a container
  - `docker inspect --format ‘{{.HostConfig.LogConfig.Type}}’ [container]`
- supported types
  - none
  - json-file
  - syslog
  - journald
  - gelf
  - fluentd

    - awslogs
    - splunk
    - etwlogs (Windows Event Log Tracing)
    - gcplogs (Google Cloud)
- the `docker logs` command ONLY works when coonfigured for json-file and journald

Complete Setup of a Swarm Mode Cluster with Managers and Worker Nodes

- manager setup
    - `docker swarm`
        - `init --advertise-addr [ip address of manager]`
        - take note (copy) the necessary `docker swarm join` command and token
    - `join-token manager` - to add another manager, following prompts
    - `join-token worker`
        - will redisplay the command and token needed for a node to join a swarm
- worker node setup
    - `docker swarm`
        - `join --token [token]`
        - repeated on each node intended for the cluster
- verify
    - on management node
        - `docker node ls`
        - will display all joined nodes

- `docker system info`
    - show state of the swarm in the output

State the Differences Between Running a Container and Running a Service

- The `docker run` command was originally the container equivalent of 'the face that launched a thousand ships'. It was responsible for the container revolution that we are in now, but times have changed.
- Although containers now give us the flexibility, portability, granularity, and abstraction that allow us to get the most out of our environments and deployments, it quickly became 'too limited'.
- We needed an easier way to deploy complex configurations in highly available and easily scalable implementations. This required the development of cluster management and control software (like Docker Swarm or Kubernetes) to work directly with Docker containers. As a result, a new paradigm was needed to address the requirements of highly scalable, clustered container environments.
- Whereas containers are limited to the single host they are started on, services are containers that live on a scalable number of 'workers' in a cluster of systems. Docker Swarm then handles access to and availability of that service across those worker nodes, eliminating the challenges of routing and accessing individual containers.
- Scalability is key in the enterprise, both up and down, to maximize your infrastructure spend, and services allow you granular control of CPU, Memory, Disk, Network, and more.

Demonstrate Steps to Lock (and Unlock) a Cluster

- why?
    - logs used by swarm managers are encrypted on disk
    - access to the nodes gives access to the TLS key used to encrypt them
    - locking a cluster further protects those keys in environments once the restarts until the unlock key is provided

- `docker swarm init --autolock --advertise-addr [ip of manager]`
  - new swarm, will be locked on initialization
  - NOTE the unlock key in password management or some other mechanism
  - will allow nodes to join, otherwise function normally UNTIL management node restarts, then nothing will run until the swarm is unlocked
  - additional managers can join, but, would need to be unlocked with same key on restart
- `docker swarm update --autolock=true`
  - will update existing swarm so it is locked, will display the unlock key
- `docker swarm update --autolock=false`
  - will prompt for unlock key and disable locking
- on restart of locked manager
  - `docker swarm unlock`
    - prompts for unlock key
- lost the unlock key but still have a manager available?
  - `docker swarm unlock-key`
    - will display the current unlock key
- key rotation - for security on regular basis
  - `docker swarm unlock-key --rotate`
    - will display NEW unlock key, NOTE IT
    - keep old key logged somewhere for a period of time until you can verify all management nodes have the update

Extend the Instructions to Run Individual Containers into Running Services Under Swarm

and Manipulate a Running Stack of Services

- replica
  - the number of threads running a service in a given swarm
- initialize a single replica
  - `docker service create [image]`
    - new default: `add --detach=[true/false]` (false will be new default)
  - `docker service update`
    - options (see below)
    - update a running service
  - `docker service ls`
    - list running services with the number of replicas on a cluster
- NOTE: run `--detached=false` to see a COOL VISUAL REPRESENTATION OF WHAT IS HAPPENING AND WHEN CONVERGED
- options for create and/or update a service
  - `--name [name of service]`
  - `--publish [#:#]` - publish indicated ports and map to underlying host - NOTE: bypasses routing mesh
  - `--env [var=value]` - environment variable published to each worker in the swarm
  - `--workdir [directory]` - working directory inside the container/service
  - `--user [username]` - runas the indicated user
  - `--network [network name]` - can be used to bypass the mesh network for an overlay
  - `--replicas [#]`
  - `--publish-add [#]` - add port to host/swarm random high port

    - `--publish [-p] published=[port],target=[port][/protocol TCP or UDP]`
    - `--secret [filename]` - allows the service access to Docker secrets, use this for each secret
- Docker (using the routing mesh) will make a service available on each node (even if a replica is not running on it) on the target port
- additional options
    - `--dns [IP[,IP]]` - sets custom DNS servers
    - `--entrypoing [command]` - overwrites the default entrypoint of the image
    - `--hostname [hostname]` - create service with indicated hostname
    - `--label [something=value]` - sets metadata on a service
    - `--mount [options see below]` - works with volumes
        - `type=[type of mount]`
            - `volume ::: source=[name of volume to use] ::: destination[containerpath to mount volume name] ::: volume-label=[labelname] ::: an anonymous volume can just be a type and destination` `— i.e. docker service create --name [name] --mount type=volume,destination=/path/container [image]`
            - `bind ::: source=[host path] ::: destination[container path]`
            - `tmpfs`
    - `-- placement-pref`
        - `‘spread=node.labels.datacenter=[value]’`
    - `--limit-cpu [#] / --reserve-cpu [#]`
        - reservation allows the specification of a ‘soft’ limit, must be set lower than the ‘hard’ limit (in limit-cpu setting), used when contention is found, the limit-cpu is the MAX CPU that the container can use
        - this is a RATIO number of the number of CPUs available (i.e. limit of 1.5 on a 2 CPU system gives one and a half max to container)

- `--limit-memory [#]/--reserve-memory [#]`
    - same as memory in practice, only in MB or GB in implementation (minimum value is 4m)

Illustrate Running a Replicated vs Global Service

- replicated service
    - a service that is replicated one or more times to a given number of instances and nodes
        - runs as many tasks as specified by your command
- global service
    - runs on each active node in the swarm, at least one
- `--mode [global]`
    - replica is default when not indicated OR when replica number is provided

Increase and Decrease the Number of Replicas in a Service

- can be set on launch
    - `docker service create --name [name] --replicas 1 [image]`
        - sets a single replica in the cluster swarm
- can be updated on running service
    - `docker service update --replicas 3 [name]`
        - sets the number of replicas running on the indicated service to 3
- note that this can be done to scale up or down as needed
- same as running `docker service scale [service]=[#] [service]=[#]`

- only way to scale multiple instances at once

Setting Up a Swarm (Backup and Restore)

- Backing Up
  - Unlock and stop a manager in the swarm
  - take a copy of the entire `/var/lib/docker/swarm` directory
  - You can restart the manager
- Restore
  - stop Docker on the node to restore the swarm to
  - remove the `/var/lib/swarm` directory on that node
  - restore the previous backup `/var/lib/docker/swarm` directory
  - reinitialize the swarm so it does not attempt to reconnect to old nodes
    - `docker swarm init --force-new-cluster`

Demonstrate the Usage of Templates with "docker service create"

- templates can be used for some flags related to service creation
- `docker service create`
  - `--hostname`
  - `--mount`
  - `--env`
    - `.Service.ID`
    - `.Service.Name`

        - `.Service.Labels`
        - `.Node.ID`
        - `.Task.ID`
        - `.Task.Name`
        - `.Task.Slot`
- examples
    - `docker service create --name hosttempl --hostname="{{.Node.ID}}-{{.Service.Name}}" --detach=false [image] [command]`
- could be used to create dynamic mount points to share with the containers
- hostnames could be pulled and autoadded to DNS if desired (of container instances in the service)
    - `docker service ls`
    - `docker service ps [serviceID]`
        - note list of nodes
    - `docker node inspect --format="{{.Config.Hostname}}" node.1.[ID]`

Apply Node Labels to Demonstrate Placement of Tasks

- look at the node information (on a manager)
- `docker node inspect --pretty [nodeID]`
    - provides a 'human readable' method of showing the JSON output from inspect commands
- labels - can control resource deployments with names
    - `docker node update --label-add [name] --label-add [var=value] [nodeid]`
        - NOTE: you can add multiple consecutive labels in the same line

        - NOTE: to USE a label in a service creation/update, you must assign it a value

- example
    - in the example below, we are creating an HTTPD service running in a two node cluster and running two replicas (which would normally run one on each host), we will set the service however, to run only on the node with the indicated label
        - i`docker service create --name constrainttest -p 80:80 --constraint 'node.labels.mynode == first-node' --replicas 2 httpd`
- valid attributes are:
    - node.id
    - node.hostname
    - node.role
    - node.labels
    - engine.labels

Convert an Application Deployment into a Stack File Using a YAML Compose File with 'docker stack deploy'

- install docker-compose
    - epel-release
    - python-pip
    - `pip install --upgrade pip`
    - `pip install docker-compose`
    - `yum upgrade python*`
    - `docker-compose -v` (so v3 of YAML files are supported)
- api-servers and load balancer example

- create Dockerfile
  - `# dockerfile for simple apache server` `LABEL maintainer="admin@linuxacademy.com" RUN yum install -y httpd RUN echo "Container Website" >> /var/www/html/index.html ENTRYPOINT apachectl -DFOREGROUND`
- version: '3' `services: apiweb1: image: myhttpd:v1 build: . ports: - "81:80" apiweb2: image: myhttpd:v1 ports: - "82:80" load-balancer: image: nginx:latest ports: - "80:80"`
- `docker-compose up -d`
- `docker-compose ps`
- `docker-compose down --volumes`
- `docker-compose push`
- deploy to the swarm
  - `docker stack deploy --compose-file docker-compose.yml [stackname]`
  - NOTE: build step will be ignored as it is NOT supported by stack deployments
  - `docker stack rm [stackname]`

Create a Docker Bridge Network for a Developer to Use for their Containers

- `docker network create`
- `docker network connect`
  - NOTE: adding a container to a network does NOT disconnect from previous, just adds to another with another IP
  - NOTE: when connecting to a USER DEFINED network, the address can be assigned with --ip=[IP]
- `docker network ls`
- `docker network rm [network]`

- `docker container inspect [container]`
  - `--format="{{.NetworkSettings.Networks.[network name].IPAddress}}"`
    - cleanly display the network IP address for the given network name and container
- bridge network creation (and common options)
  - `docker network create --driver=bridge [optional args] [interface name]`
    - `--subnet=[network/mask]` (i.e. 192.168.0.0/16)
    - `--ip-range=[range/mask]` (i.e. 192.168.1.0/24)
    - `--gateway=[IP]`
    - `-o` (`--opt`) `[option list below]`
      - `com.docker.network.bridge.name`
      - `com.docker.network.bridge.enable_ip_masquerade`
      - `com.docker.network.bridge.enable_icc`
      - `com.docker.network.bridge.host_binding_ipv4`
      - `com.docker.network.driver.mtu`

How to Configure Docker to Use External DNS

- `/etc/docker/daemon.json { "dns": ["8.8.8.8", "8.8.4.4"] }`
- `docker run -- dns [IP Address]`
- normal DNS is passed through from host
- `docker exec -it [container] /bin/bash`
  - `cat /etc/resolv.conf`

Publish a Port So That an Application is

Accessible Externally and Identify the Port and IP It Is On

- `docker run`
  - `--publish [-p] [hostport:containerport/protocol]`
    - will remap the indicated host port to the indicated container port for accessing the container service
  - `-P`
    - will map A host port (above 32000) to the default container exposed port
- `docker container ps`
  - will list characteristics of running container including ports and port mapping
- `docker container inspect [container]`
  - grep for 'IPAddress' for IP of container
  - grep for 'Ports' for port mapping information
  - `--format="{{.NetworkSettings.Networks.[network driver].IPAddress}}"`
    - for container IP
  - `--format="{{.NetworkSettings.Ports}}"`
    - for container to host port mapping output

Deploy a Service on a Docker Overlay Network

- `docker network create`
  - `--driver=overlay`
  - `--subnet=[network/mask]`
  - `--gateway=[IP]`

    - `--ip-range=[subnet/mask]` - only good to further divide network from subnet definition

- `docker service create`
    - `--network=[network name]`
- once nodes are running
    - `docker network inspect [networkname]`
        - run on any host once deployed with instance, will show the instance information for IP on that network
- NOTE: underlying hosts will not be able to see/interact with the new overlay network
    - `docker ps`
        - `docker container inspect [container]`
        - `docker exec –it [container] /bin/bash`
            - will be able to see all other IPs in that network

Troubleshoot Container and Engine Logs to Understand Connectivity Issues Between Containers

- `docker service logs`
- `docker container logs`
- `cat /var/log/messages | grep [dD]ocker`

Describe How Storage Can Be Used Across Cluster Nodes

- `docker volume create [name]`

    - `docker volume ls`
        - list created volumes
        - will propagate on service creation
    - `docker volume inspect [name]`
        - for characteristics
    - `docker volume rm [name]`
- `docker service create`
    - no volume (`--volume or -v`)
    - `--mount [source=/path],target=[path]`
- `/var/lib/docker/volumes/[volumename]/_data`
- cannot share the underlying volume (each instance uses its own storage)
    - NOTE: Docker for AWS and Azure do support shared storage using the 'Cloudstor' plugin
        - outside scope for exam

Identify the Steps You Would Take to Clean Up Unused Images On a Filesystem (Pruning)

- `docker prune`
    - `--all (-a)` - remove unused images (not just dangling)
    - `--force (-f)` - force removal without prompting
    - `--volumes` - prune unused volumes as well (not default)
- must be done on each system, not a cluster/swarm aware command
- removes objects
    - volumes

- containers
- images
- networks
- build caches

Describe the Process of Signing an Image

- deploy a registry with certificates (self signed is fine, see registry creation and deployment)
- enable Docker trust in a shell
  - `export DOCKER_CONTENT_TRUST=1`
    - advantage of shell export is just for THAT shell environment, allowing flexibility
- builds of content then (Dockerfiles) will need to have it explicitly disabled to build successfully
  - `docker build`
    - `--disable-content-trust`
- push the trusted content (which will create the required signature keys, creating them if needed on first push)
  - self-signed will error
- push the content after resetting the variable
- set trust variable on the client
  - pull the image
    - should error on the self signed certificate, untrusted (unsigned)

Set Up and Configure Universal Controller Plane and Docker Trusted Registry for Secure Cluster Management

- configure Docker and desired cluster state FIRST
- UCP INSTALL
- `docker pull docker/ucp`
  - on manager node
- `docker container run --rm -it --name ucp \ -v /var/run/docker.sock:/var/run/docker.sock \ docker/ucp:2.2.4 install \ --host-address <node-ip-address> \ – MUST be same IP used in swarm setup for manager --interactive`
  - `--rm` as this conteiner can be removed automatically, others will spink up
- obtain Docker EE free 30 day trial license from store.docker.com, download to system with browser
- `/etc/hosts`
  - sample from test environment `172.31.22.32 tcox4.mylabserver.com 172.31.22.32 ucp.example.com 172.31.28.127 tcox5.mylabserver.com 172.31.28.127 dtr.mylabserver.com 172.31.29.156 tcox6.mylabserver.com`
- DTR INSTALL
- `docker run -it --rm docker/dtr install --ucp-node tcox5.mylabserver.com \ – where the DTR is going to be --ucp-username admin --ucp-url https://tcox4.mylabserver.com \ – where UCP itself is --ucp-insecure-tls` - trust the certs that are used
- same login/password
- apply the same license

Complete Configuration of Backups for UCP and DTR

- UCP
  - `docker container run --log-driver none --rm -i --name ucp \ -v /var/run/docker.sock:/var/run/docker.sock \ docker/ucp backup > ucp-backup.tar`
  - NOTE: UCP will be offline during backup but it will NOT affect running services

    - `--debug (-D)` - verbose debugging output
    - `--passphrase [passphrase]`
        - used to encrypt tar file

- DTR
    - `docker run -i --rm --ucp-insecure-tls --ucp-username admin docker/dtr backup > dtr-backup.tar`
    - NOTE: DTR will be offline during backup and will not be available for pulls/pushes at that time
    - `--debug` - verbose debugging output
    - `--ucp-insecure-tls` - disables certificate verification
    - `--ucp-username[password]` - the username or password for UCP

Create and Manage Users and Teams

- UCP Console demo
- DTR Console demo

COnfigure RBAC in UCP and Enable LDAP for Authentication in UCP

- UCP Console demo
- RBAC = Role Based Access Control