**Docker**

(Learning from KodeKloud’s YouTube video *Docker for Beginners: Full Course*)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Why do we need Docker?** | | | | | | | | | |
| **🡪 Issues**  1) Compatibility/Dependency  - Using many different components in an end-to-end application  - Version of operating system, dependent libraries etc in architecture  - Compatibility matrix issue  2) Long setup time  - For new developers  - Many instructions and command to set up the application  3) Different developer/test/production environments  - Due to difference preferences in OS, versions etc  - Need to ensure that the application is running the same way in different environments  **🡪 Features**  - Runs each component in a separate container with its own dependencies, libraries etc  - Manages these components  - Same VM or OS  - Build Docker configuration once to set up the environment  **🡪 Benefits**  - Solves compatibility issue  - Allows the modification of components without affecting the other components  - Lightweight, boots up faster  - Less isolations, more resources are shared between the containers | | | | | | | | | |
| **Containers** | | | | | | | | | |
| - Completely isolated environments (e.g. file system etc)  - Have their own processors, network interfaces, mounts etc  - All share the same OS kernel (Linux, Windows etc)  - Container OS needs to match host kernel  **🡪 How they work**  - Not mean to host an operating system  - Meant to run a specific task or process  - For example, host an instance of a web server, application server, database or compute/analysis something  - Once task completed, container exits (only lives as long as processes in it are running)  - If service in container is stopped/crash, container exits  - Does *not* listen to a standard input (i.e. input from console), unlike a regular CLI app  - Runs in an non-interactive mode  🡪 **How to users access the application**  - By using the port which the application is listening on  - To access from web browser:  1) Use internal IP of Docker container (only accessible within Docker host)  2) Do *port mapping*  🡪 **Running a container for an OS**  More precisely if image is just a base image for other images (e.g. Ubuntu),  - Runs an instance of image  - Exits immediately  - Container status will be shown as “*Exited*”  **Why?**  - By default, Docker does not attach a terminal to a container when it is run  - Bash does not find a terminal and exits  - Process started when terminal created (i.e. Bash) finishes, therefore container exits | | | | | | | | | |
| **Operating systems** | | | | | | | | | |
| - Consists of 2 things: OS kernel + set of software | | | | | | | | | |
| **OS Kernel** | | | | | | **Set of software** | | | |
| - Responsible for interactions with underlying hardware | | | | | | - Makes operatings systems different  - User interface, drivers, compilers, file managers, developer tools etc | | | |
| **Docker Image** | | | | | | **Docker Container** | | | |
| - Package/template/plan  - Used to create one or more containers  - Can be created with a Docker file of requirements from both Developers and Operations (based on the Developer Guide and instructions for the application)  - Can be deployed in production  - Examples: mongoDB, nodejs, redis etc | | | | | | - Running instance of image  - Isolated, have their own environments, set of processors | | | |
| **Volume mapping** | | | | | | | | | |
| 🡪 **How data is persisted in a Docker container**  - Example: Running a MySQL container  - Creates databases and tables  - Data stored in /var/lib/mysql in container  - Docker container has its own isolated filesystem (any changes in files happen in container)  - Directory name in “/var/lib/docker/” folder is the same as container ID for files related to container | | | | | | | | | docker run mysql |
| - All data in container is gone | | | | | | | | | docker stop mysql  docker rm mysql |
| - To keep data, map directory outside the container on Docker host to directory inside container  - **\*\* Implicitly mounts external directory to a folder inside the docker container**  - All data will be stored at external volume in the outside Docker directory  - Remains even if container is deleted | | | | | | | | | docker run -v <docker host dir>:<container dir> <container name> |
| **Creating an image** | | | | | | | | | |
| 1. Create Dockerfile and write down instructions for setting up your application in terms of CLI commands you would have used (like a bash script) | | | | | | | | |  |
| Example | **FROM** Ubuntu  **RUN** apt-get update  **RUN** apt-get install python  **RUN** pip install flask  **RUN** pip install flask-mysql  **COPY** . /opt/source-code  **ENTRYPOINT** FLASK\_APP=/opt/source-code/app.py flask run | | | | | |  | | |
| 1. Build image using docker build command and tag the application version   **Layer 1. Base Ubuntu layer**  **Layer 2. Changes in apt packages**  **Layer 3. Changes in pip packages**  **Layer 4. Source code**  **Layer 5. Update Entrypoint of image with “flask” command** | | | | | | | | | - Docker builds images in an *layered architecture*  - Each line of instruction creates new layer in image  - *Docker build output:* Can see each step and output of each command/layer/task |
|  | docker build Dockerfile -t mmumshad/my-custom-app | | | | | | | | |
| 1. Make local image available on Docker registry   - Meant to be small and lightweight | | | | | | | | |  |
|  | docker push mmumshad/my-custom-app | | | | | | | | |
| **If a layer fails during the image build process** | | | | | | | | | |
| - After fixing issue and running “docker build”, reuse previous layers from cache and continue to build the remaining layers  - Faster rebuilding of image, no need to start from first layer of build process  - Helpful if updating source code of application (only layers above updated layers need to be rebuilt) | | | | | | | | |  |
| **DockerFile** | | | | | | | | | |
| **Argument** | | | |  | | | | |  |
| CMD | | | | Overrides default command (specified in image’s *“CMD”*) to run when container starts | | | | |
| ENTRYPOINT | | | | Appends param to command indicated in “*ENTRYPOINT*” | | | | |
| **Arguments *CMD* and *ENTRYPOINT*** | | | | | | | | | |
| - An *ENTRYPOINT* command can be a bash script (.sh) | | | | | | | | | |
| Examples of Dockerfile | | # Overrides default command  **FROM** Ubuntu  # Shell format  **CMD** sleep 5  # or JSON format: CMD [“command”, “param1”]  **CMD**[“sleep”, “’5]  ## CMD: command line params override completely | | | **FROM** Ubuntu  # Specify program to run when container starts  **ENTRYPOINT**[“sleep”]  ## ENTRYPOINT: command line params get appended  # > docker run ubuntu-sleeper 10  # is the same as  # > docker run ubuntu sleep 10 | | | # Always specify in JSON format  **FROM** Ubuntu  **ENTRYPOINT**[“sleep”]  # Default value if no value is specified in command line, if have then that value overrides this  **CMD**[“5”]  # sleep 5 sec  # > docker run ubuntu-sleeper  # sleep 10 sec  # > docker run ubuntu-sleep 10 | |
| Examples in CLI | | # Modify ENTRYPOINT command during runtime  # sleep2.0 is the new command  docker run **--entrypoint** ubuntu-sleeper 10 | | |  | | |  | |
| **Networks** | | | | | | | | | |
| **🡪 Default networks** (when Docker is installed)   1. Bridge 2. Null/None 3. Host   - Docker has a built-in DNS server  - Helps containers resolve each other using the container names  - Always runs at address 127.0.0.11  **🡪 How are containers isolated within host?**  - Use network name spaces  - Create separate network name space for each each container  - Use virtual ethernet pairs to connect containers together | | | | | | | | |  |
| **Bridge** | | | - Default network that a container gets attached to  - Private internal network  - Created by Docker on the host  - Containers attached to this network usually get IP address in the range **172.17** series (172.17.0.1 … 172.17.05 etc)  - Containers can access each other using internal IP if required  - To access these containers to outside world,  Method 1) Map ports on containers to ports on docker host  - By default, only **one internal network** is created | | | | | |  |
| **None** | | | Containers:  - Are not attached to any network  - Does not have access to external network or other containers  - Run in an isolated network | | | | | |
| **Host** | | | - To access these containers to outside world,  Method 2) Associate container to host network  - Take out any network isolation between Docker host and Docker container (no need port mapping, container uses the host’s network)  - No longer able to run multiple containers on same host and same port (ports now common to *all* containers in host network) | | | | | |
| **Storage drivers** | | | | | | | | | |
| - Responsible for  - Maintaining layered architecture  - Creating a writable layer  - Moving files across to enable copy-on-write etc  - Selection of driver depends on underlying OS being used  - Ubuntu: AUFS driver  - Different driver, different performance and stability characteristics  - Choose base on needs  - Common drivers  - AUFS, ZFS, BTRFS, Device Mapper, Overlay, Overlay2 | | | | | | | | |  |
| **Docker compose** | | | | | | | | | |
| - When setting up a complex application such as running (*docker run …)* multiple services  - Create a configuration file in *yaml* format, put together different services and options specific to the file  - Only applicable to running all containers on a *single* docker host | | | | | | | | | docker-compose.yml  # Bring up entire app stack  **docker-compose up** |
| 🡪 How to do so  - Individually *docker run* the necessary containers  - Naming containers is important  - Link all the containers together (tell them to use each other specifically)  - Use command-line option to link container to container it is depending on | | | | | | | | |
| Structure of *yaml* file:  - Dictionary of container names  - Under each item, specify image to use  - **\*\* Basically translating all the docker runcommands and their parameters into this file** | | | | | | | | |
| **Docker registry** | | | | | | | | | |
| - Central repo of Docker images  - For example, image nginx  image: nginx  actually refers to  image: nginx/nginx  **(user or account/image or repository)**  and is stored at Docker hub  image: docker.io/nginx/nginx  **(registry/user/image)** | | | | | | | | |  |
| **Private registries**  - Good for internal private in-house applications  - Accessible only with credentials  - If not logged in, Docker will say that image cannot be found | | | | | | | | | docker login private-registry.io  docker run private-registry.io/apps/internal-app |
| **Docker engine** | | | | | | | | | |
| - Docker engine = host + docker installed on it  - Related to docker architecture  - how it runs applications in isolation  - how it runs under the hood  - What is installed when Docker is installed  **1) Docker CLI**  - Command-line interface  - Uses REST API to interact with with Docker Daemon  - Does not need to be on the same host, can be on another system like a laptop  **2) REST API**  - API interface that programs and used to talk to daemon and provide instructions  - Own tools can be created using this API  **3) Docker Daemon**  - Background process that manages Docker objects (e.g. images, containers, volume and networks etc) | | | | | | | | |  |
| **Containerisation** | | | | | | | | | |
| - Docker uses *namespaces* to isolate workspaces  - Workspaces such as  - Processing IDs  - Network  - Inter-process communication  - Mounts  - Unix time-sharing systems  - Each workspace created in their own namespace, provides isolation between containers | | | | | | | | |  |
| **Process ID (PID) namespaces**  - When Linux system boots up, starts with one process with a process ID of one  - Root process  - Kicks start all the other processes  - By the time system starts up completely, there are a handful of processors running  - Check processes with ps command  - Processors are unique (no two processors have the same process ID)  - A container/child system within Linux system  - Thinks it is an independent system of its own  - Has its own set of processes  - No hard isolation between containers and underlying host, so processors running within container are actually running on host  - Unique PID  - With PID namespaces, each process can have multiple process IDs, associated with it  - Process that starts in container gets next available PID on underlying system | | | | | | | | | Linux system  PID: 1  PID: 2  PID: 3  PID: 4 (same)  Child system (container)  PID: 1  PID: 2 (same) |
| **Container Orchestration** | | | | | | | | | |
| **🡪 Situation**  - If number of users increase, and need to manually run new instances of container multiple times  - Need to deploy additional instances and keep watch of load and performance of application  - Need to also keep watch if a container fails, or if the host crashes  **🡪 Solution**  - Avoid manual monitoring to some extent  **- A set of tools and scripts that can help host containers in a production environment**  - Typical solution: multiple Docker hosts that can host containers  - Even if one fails, application is still accessible  - Deploy 100s, 1000s of instances of applications with a single command  - Some solutions:  - Scale up when demand increase, scale down when demand decrease (**scaling**)  - Automatically add additional hosts to support user load (**clustering**)  - Advanced networking between containers across different hosts  - **Load balancing** across different hosts  - Provide sharing storage between host, support management and security within cluster  **🡪 Well-known solutions today**  - Docker Swarm  - lacks advanced complex scaling features  - Kubernetes  - a bit difficult to set-up  - provides a lot of options to customize deployments  - has support for many different vendors (e.g. public cloud service providers)  - popular  - MESOS (“mez-o-s”)  - difficult to set-up and get started, supports many advanced features | | | | | | | | |  |
|  | | | | | | | | |  |
|  | | | | | | | | |  |
|  | | | | | | | | |  |
|  | | | | | | | | |  |
|  | | | | | | | | |  |

|  |  |
| --- | --- |
| **Docker Swarm (docker swarm)** | |
| - Combine multiple Docker machines together into a single cluster  - Will take care of distributing services/application instances into separate hosts for high availability and balancing across different systems and hardware |  |
| - Requires multiple hosts with docker installed on them  - Designate one to be *swarm manager/master* and the rest as *workers/slaves/nodes* |  |
| # on docker swarm manager host  docker **swarm init** … # initialize manager  # on workers  docker **swarm join --token <token>** # command provided by manager during init | |
| **Docker service**  - Key component of swarm orchestration  - Definition: one or more instances of a single application/service that runs across the nodes in the swarm cluster |  |
| # run on the **manager node\*\***  # indicate number of instances of image to run across the cluster  docker **service create --replicas=<int>** <image name>    # can use arguments lie **--network, -p** for ports etc | |
|  |  |
|  |  |
| **Kubernetes (kubectl)** | |
| - Instances can automatically scale up or down based on user load  - Can create more instances than Docker Swarm  - kubernetes open architecture provides support for  - many storage networks  - a variety of authentication and authorization mechanisms  - All major cloud service providers have native support for Kubernetes  - **kubernetes uses Docker hosts to host applications in the form of Docker containers**  - or alternative containers (e.g. rkt, cri-o) |  |
| **🡪 kubernetes architecture**  - kubernetes cluster consists of a set of *nodes*  - Even if one node fails, application still accessible from other nodes  - A *node*  - physical or virtual machine (a *worker machine*)  - where ubernetes software/set of tools are installed  - where containers are launched by kubernetes  - if node where application is running on fails, node goes down |  |
| **🡪 Managing cluster**  - Store information about cluster  - How are nodes monitored  - When node fails, how to move workload of failed node to another worker node  - Done by a *master node*  - Has Kubernetes control plane components installed  - Watches over nodes in cluster  - Responsible for actual orchestration of containers on the worker nodes |  |
| **🡪 Installing kubernetes**  - Multiple components installed  **1) API server**  - For front-end for user management devices, CLI interfaces to talk to API, interact with cluster  **2) etcd server**  - Reliable key-value store  - Stores all data to manage the cluster  - Stores all information about multiple nodes in a cluster in a distributed manner (considering that there are multiple clusters as well)  - Responsible for implementing a law within the cluster to ensure no conflicts between master nodes  **3) kubelet service**  - Agent that runs on each node in the cluster  - Responsible for making sure that the containers are running on the nodes as expected  **4) Container Runtime**  - Underlying software that is used to run containers  - E.g. an engine like Docker  **5) Controllers**  - The “brain” behind orchestration  - Responsible for noticing and responding when nodes, containers or endpoints goes down  - Makes decisions to bring up new containers in cases of crashed nodes  **6) Scheduler**  - Responsible for distributing work/containers across multiple nodes  - Looks for newly created containers and assigns them to nodes |  |
| **🡪 kubernetes CLI**  - Or called “cube control tool”  - Used to deploy or manage applications on cluster to get cluster-related information, status of nodes in cluster etc |  |
| # Examples  # deploy application on cluster  **kubectl run** hello-minikube  # view information about cluster  **kubectl cluster-info**  # list all nodes part of cluster  **kubectl get nodes** |  |
| # Examples  # run many instances of the same application  **kubectl run --replicas=1000** my-web-server  # scale up the number of instances (based on user load)  **kubectl scale --replicas=2000** my-web-server  # upgrade 2000 instances in a rolling upgrade fashion one-at-a-time  **kubectl rolling-update** my-web-server **--image=web-server:2**  # roll back images if something goes wrong  **kubectl rolling-update** my-web-server **--rollback**  # test new features of application, upgrade only some instances (A/B testing)  … | |
|  |  |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Term** | **Meaning** | | | **Notes** |
| *Docker hub/store* | public Docker repository | | |  |
| *Load-balancer* |  | | |  |
| *DevOps* | Developer + Operations team | | |  |
| *Hypervisor* | - Also known as “virtual machine monitor” (VMM)  - Computer software, firmware or hardware that creates and runs virtual machines | | |  |
| *Host machine* | A computer on which a *hypervisor* or moreruns | | |  |
| *Guest machine* | Each virtual machine running on the host machine | | |  |
| *Attached mode* | - Attached to console/standard output of Docker container  - Able to see output of application on console  - Won’t respond to user inputs  - Stop container and exit: CTRL+C | | | When running a container with docker run |
| *Detached mode* | - Container runs in back end | | |  |
| *Supported tags* | - Associated with an image at dockerhub.com  - Each version of a software can have multiple short and long tags | | |  |
| *IP of Docker host* | - Local host port/IP  - *“application is available on port … on the host”* | | |  |
| *IP of Docker container* | - Internal IP  - Assigned by default (each container has one)  - Only accessible within Docker host  *- “application listens on port …”* | | | <http://…> |
| *Port mapping/publishing* | - Routes all traffic on local port to internal port  - **\*\*** Note: **Cannot map to same port on Docker host more than once**  **Benefits**  - Run multiple instances of application and map them to different ports on docker host  - Can run as many applications and map them to as many ports desirable | | | docker run **-p**  **<local port>:<internal port>** <image name> |
| *Volume mapping* | - Saves all data in container to external directory on Docker host  - Refer to notes above | | |  |
| *Logs* | Contents written to standard output of the container | | |  |
| *Dockerfile* | - Text file written in a specific format that Docker can understand  - In an *instruction-and-argument* format  - All Dockerfiles must start with a “*from*” instruction  - Possible naming convention if there are multiple Dockerfiles, “***Dockerfile-<app name>****”* | | FROM <base OS/image>  RUN <command to run on base image>  # Copy files from local system to Docker image  COPY <src code dir> <dest src code dir in Docker>  # Port which application is run within container  EXPOSE <port no.>  # Specify a command to run when image is a container (command to run app)  ENTRYPOINT … | |
| **Instruction**:  FROM, RUN, COPY, ENTRYPOINT, EXPOSE, WORKDIR, CMD etc  **Argument**: <bash command> | |
| *Docker build output* | - When running “docker build” command  - See various steps involved and their task output  - All layers built are cast, layered architecture helps to start “docker build” from a particular step in case it fails or if new steps were added in the build process (no need to start all over again) | | |  |
| *File system* | - How docker stores data on local file system when installed  - Folder structure, where data is stored by default  /var/lib/docker  | aufs  | containers  | image  | volumes  - Data related to images and containers running on docker host | | |  |
| *Layered architecture* | 🡪 Typical layer   1. Base OS 2. Packages 3. Dependencies 4. Source code of application 5. Entry point 6. **Container layer** (when container of image created)   - A new **writable** layer on top of image layer  - Exists only as long as container is alive  🡪 Benefits  - Reuses layers from cache if another application has the same layers or rebuilding image  - Builds images faster, save disk space | | | - Once build is complete, files of an **image** are **read-only**  - Modify files of application by initiating a new build |
| *Copy-on-write mechanism* | - Read-only file in image layer is copied to read-write container layer  - This copied file can be modified in container layer | | |  |
| *control groups/”cgroups”* | Control or restrict the amount of hardware resources allocated to each container | | |  |
|  |  | | |  |
|  |  | | |  |
|  |  | | |  |
|  |  | | |  |
|  |  | | |  |
|  |  | | |  |
|  |  | | |  |
| **Phrases** | | | | |
| *Containerised-versions of applications* | | Applications that can be containerized are databases, developing tools, operating system, browsers, utilities | | |
| *Instances of web services* | |  | | |
| *Ports exposed on container (****internal****)* | | - Port on the right of arrow  For example: 0.0.0.0:**3456->3456**/tcp  - Exposed port is *3456* | | |
| *Ports published on* ***host*** | | - Port on left of arrow  For example: 0.0.0.0:**38080->80**/tcp  - Published port is *38080* | | |
| *Package application into Docker image* | |  | | |
| *Publish port XXXX to YYYY on host* | | Container to map host port YYYY to internal port XXXX | | |
|  | |  | | |
|  | |  | | |
|  | |  | | |
|  | |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Commands** | **Purpose** | | **Notes** |
| **docker version** | List version of Docker used by host | |  |
| **docker -H=<remote docker engine address>:<port>** | Run a container using a remote docker host | |  |
| **docker service create --replicas=<int> <image name>** | Create multiple docker hosts for containers | | - Helps prevent inaccessible application if a Docker host crashes |
| **Example Command** | | **Explanation** | |
| docker -H=remote-docker-engine:2375 | |  | |
| docker service create --replicas=100 nodejs | |  | |
| **docker images** | | | |
| **docker pull** <app name> | - Only pull/download image and store on host  - Does *not* create a container | |  |
| **docker images** | List all available **images** on host | | Output column names:  REPOSITORY (image name)  TAG  IMAGE ID  CREATED  SIZE |
| **docker rmi** <image name> | - Remove images for good  - Make sure no containers are running off of that image first (*delete all dependent containers*)  - Image name is in the form “*REPOSITORY:TAG*” | | - Remove multiple images by adding more image names |
| **docker build** Dockerfile -t <tag name for image> | - Build image from application (“dockerise app”)  - Creates image locally on system | |  |
| **docker push** <image name> | Make local image available on Dockerhub registry | | Example of image name:  account\_name/app\_name |
| **docker history** <image name> | See layers in image | |  |
| **Example** | | **Explanation** | |
| docker build Dockerfile -t mmumshad/my-custom-app | | Build an image from an application with a tag of the app version | |
| docker build -t webapp-color . | | Build a docker image using the Dockerfile (currently in directory of file) with no tag specified  - Fullstop (.) represents Dockerfile | |
| docker build -t webapp-color:lite . | | Build an image, name it *webapp-color* and tag it *lite* (use colon) | |
| mmumshad/my-custom-app | | An example of an image name | |
| **docker run** | | | |
| **docker run** <image name> | - Run a container from an image  - If image not present locally, it will be pulled from Docker hub and stored  - Runs in **attached** mode (foreground)  More precisely if image is just a base image for other images (e.g. Ubuntu),  - Runs an instance of image  - Exits immediately  - Container status will be shown as “*Exited*” | |  |
| docker run **-d** … | - Run a container from an image in **detached** mode (background) | |  |
| docker run **-i** … | - To enable application to **read inputs** (interactive mode)  - Map standard input of host to Docker container  - Short for “interactive” | | Note: Application prompt on terminal will be missing (e.g. “*Please enter your name:*”*)* |
| docker run **-t** … | - Enable output by application as prompts  - Attach terminal to container  - Short for “pseudo terminal” | |  |
|  | **Mapping** | | |
| docker run **-p <local host port>:<internal port>** … | - \*\* **Map local host port to internal port** (i.e. map port *inside* Docker container to a **free** port on the Docker host/expose application on hosts port)  - All traffic on local port will be routed to internal port | | - Allows access for users outside Docker host |
| docker run **-v <docker host dir>:<container dir>** <container name/ID> | - **\*\* Volume mapping**  - Store data of application, prevent its deletion upon stopping of container | |  |
|  | **Storage** | | |
| **docker run --mount type=<volume/bind>, source=<file path>, target=<file path>** <image name> | - **\*\* Volume/bind mounting**  - Allows data to persist by storing data written by container into a read-write folder | |  |
|  | **docker compose** | | |
| **docker run --link <container depended upon>:<host name>** <image name> | Allows a container to link to a container it needs/depends on  - Deprecated feature and may be removed in the future in Docker | |  |
|  | **Networking** | | |
| docker run <image name> **--network=<network name>** | Associate container with non-default network | | Network names are  - “Bridge” (default)  - “none”  - “host” |
|  | **docker Engine** | | |
| docker **-H=<engine address>:<port>** run … | Run a container on a remote engine host | |  |
| **docker run --cpus=<0-1>** … | Restrict CPU access to container | |  |
| **docker --memory=<limit> …** | Restrict memory usage by a container | |  |
| docker run **-e <var name>=<value> .**... | - Set/configure value to **environment variable**  - Value will be passed into application and can used | |  |
| docker run **--name** <given name> <image name> | Run a container from an image and **name** it | |  |
| docker run <image name>**:<version no.>** | - Run another version of an image (e.g. older ver, another tagged version)  - Specify version to use in a **tag** (identified with a colon)  - (Default) No tag means latest tag (i.e. version) | |  |
| docker run <image name> <command> | - **Override** default command in “**CMD**” instruction of image (command to run when container starts)  - Specify a different command to start the container  - When container starts, it runs the command | | docker run ubuntu sleep 5 |
| docker run **--entrypoint <new command>** <image name> [<params…>] | Override command specified in “*ENTRYPOINT*” **during runtime** | |  |
| **Example Command** | | **Explanation** | |
| docker run ubuntu sleep 5 | | Container sleeps for 5 seconds | |
| docker run -it <image name> | | In interactive mode and attached terminal to container | |
| docker run -p 80:5000 <image name> | | Routes traffic from local port 80 to internal port 5000 (users access with **<IP>:<local port>** like <http://192>.168.1.5:80) | |
| docker run -v /op/datadir:/var/lib/mysql mysql | | Map data directory in docker host to directory (/var/lib/mysql) in container | |
| docker run -e APP\_COLOR=blue simple-webapp | | Set environment variable *APP\_COLOR* as blue for application to have blue background | |
| docker run -e MYSQL\_ROOT\_PASSWORD=db\_pass123 mysql | |  | |
| docker run python:3.6 cat /etc/\*release\* | | Find out base OS used by image | |
| docker run Ubuntu --network=none | | Associate container with another network other than “Bridge” | |
| docker -H=10.123.2.1:2375 run nginx | | Run *nginx* container on a remote docker host | |
| docker run --cpus=.5 ubuntu | | Ensure that container does not take up more than 50% of host CPU at any given time | |
| docker run --memory=100m ubuntu | | Limit the amount of memory container can use to 100 MB | |
| **docker containers** | | | |
| **docker attach** <container ID/name> | Attach running detached container | | For container ID, providing the first few characters is enough |
| **docker start** <container ID/name> | Start a stopped container | |  |
| **docker ps** | - List all **running** **containers** and their basic information  - Container ID and name (randomly created by Docker)  - Name of image  - Current status (*Up*, *Exited*) | | Output column names:  CONTAINER ID  IMAGE (name of image) COMMAND  CREATED  STATUS  PORTS (will show ports published on the container)  NAMES (name of container) |
| docker ps **-a** | - List **all** running, previously stopped and exited containers (all present containers) | |  |
| **docker inspect** <container ID/name> | - Gives all details about a specific container  - E.g. states, mounts, configuration, data, network settings etc  - Returns all details in a JSON format  - Inspect values of environment variables under **“*Config”: “Env”***section (of a running container) | |  |
| **docker stop** <container ID/name> | Stop a container | | - Stop multiple containers by adding more ID/names |
| **docker rm** <container ID/name> | - Remove a stopped/exited container for good  - Success: prints container ID/name | | - Remove multiple containers by adding more ID/names |
| **docker exec** <running container ID/name> <command> | Execute command on running container | |  |
| **docker logs** <container ID/name> | See logs of container running/which ran in *detached mode* (background) | |  |
| **Example Command** | | **Explanation** | |
| docker exec mysql-db mysql -pdb\_pass123 -e ‘use foo; select \* from myTable’ | | View information in database where container is mysql-db with password db\_pass123 | |
| **networking** | | | |
| **docker run** <image name> **--network=<network name>** | Associate container with non-default network | | Note: duplicated from *docker run* section |
| **docker run --link <src container name>:<container alias name>** | - Allow container being created to be linked to another container *(“src container”)*  - Source container will be associated with alias name and can be referenced through it | | Note: Linking is a **LEGACY** feature  - Use the above command, connect containers to the same network to link them |
| **docker network create** \  --driver <network> \  --subnet <IP>  --gateway <IP>  <custom isolated network name> | - Create a new network  - Helps to isolate internal network (i.e. only certain containers share certain networks)  - Not all fields compulsory | | docker network create \  --driver bridge \  --subnet 182.18.0.0/16 custom-isolated-network |
| **docker network ls** | List all networks | |  |
| **docker network inspect** <network> |  | |  |
| docker inspect <container ID/name> | - See network settings and IP addressed assigned to existing network container  - In the JSON, inspect **“Networks*”: “<bridge/none/host>”*** and **“IPAddress*”, “MacAddress”*** | | Not guaranteed that the container will get the same IP address when system reboots |
| Connect containers of the same internal network | Use container name instead of IP address | |  |
| **Example Command** | | **Explanation** | |
| mysql.connect(172.17.0.3) | | **NOT** recommended as IP address may change | |
| mysql.connect(mysql) | | **Recommended**,when web container is trying to connect to database container name “*mysql*” with internal IP address 172.17.0.3 | |
| docker network inspect bridge | | Gives details and information such as the subnet configured on the *bridge* network | |
| **storage** | | | |
| - Mounting direction is <folder mounting>:<folder mounted on>  - Preserve data created by container to image (e.g. database)  - Once mounted, any changes will be stored and just do *volume mapping* to get back the data when restarting a new instance of the container with *docker run -v …* (e.g. if current container, database crashed) | | | |
| **Method 1 (Step 1)** | | |  |
| **docker** **volume create** <folder name> | Preserve data created by container to image/ add persistent volume to container | | - New folder created  /var/lib/docker  | aufs  | containers  | image  | volumes  **| <folder\_name>** |
| **Method 1 (Step 2)** | | |  |
| **docker run -v** <folder name>:  /var/lib/<image name>  <image name> | Mount folder created by *docker volume create* onto folder in image directory  - Executed after *docker volume create*  - Called ***volume mounting****:* mounts volume from /volumes directory | | - If this command is not run before the *docker run* command, docker will automatically create the folder |
| **Method 2** | | |  |
| **docker run -v** <complete file path>:<image dir> <image name> | - **Alternative:** not store data on default /var/libs/ folder  - Called ***bind mounting:***mounts a directory from any location onto docker host | | - Provide complete file path of folder to mount onto container |
| **Method 2 (alternative)** | | |  |
| **docker run --mount type=<volume/bind>, source=<file path>, target=<file path>** <image name> | - **Alternative:** better recent alternative to “-v” command | |  |
| **Example Command** | | **Explanation** | |
| docker volume create data\_volume | | - Run codes in succession | |
| docker run -v data\_volume:**/var/lib/mysql** mysql | |
| docker run -v **/data/mysql**:/var/lib/mysql mysql | | - Method 2 | |
| docker run –mount type=bind, source=/data/mysql, target=/var/lib/mysql mysql | | - Better alternative of Method 2 | |
| **docker compose** | | | |
| **- Levels:**  **1) container names**  **2) properties (e.g. image, ports, links, depends\_on, networks, environment etc)**  **3) values to properties** | | | |
| **docker run --link <container depended upon>:<host name>** <image name> | Allows a container to link to a container it needs/depends on  - Deprecated feature and may be removed in the future in Docker | | Example of Python application that requires a host:  def get\_redis():  if not hasattr(g, ‘redis’):  g.redis = Redis(**host=”redis”,**  db=0,  socket\_timeout=5)  return g.redis |
| **docker-compose up** | Brings up entire application stack, based on **docker-compose.yml**  - Aggregates the output of each container | | - When command exits, all containers are stopped |
| **Example Command** | | **Explanation** | |
| docker run -d –name=vote -p 5000:80 **–link redis:redis** voting-app | | Creates an entry in /etc/hosts file in voting-app container, adds an entry with host named redis, with internal IP of redis container | |
| docker run -d –name=worker **–link db:db –link redis:redis** worker | | In worker application:  try {  **Jedis redis = connectToRedis(“redis”);**  **Connection dbConn = connectToDB(“db”);**  System.err.println(“Watching vote queue”);  … | |
| docker-compose up -d | | Starts the containers in the background and leaves them running | |
| **docker compose format (VER 1)** | |  | |
| - Disadvantages:  - Unable to deploy application on networks other than default *bridge* network  - Cannot specify dependency/start-up order (e.g. database needs to be set up first for other app to run properly) | | | |
| **docker-compose.yml**  redis:  image: redis  db:  image: postgres:9.4  **vote**:  **image**: voting-app  **ports**:  - 5000:80  **links**:  -redis  result:  image: result-app  ports:  - 5001:80  links:  -db  worker:  image: worker  links:  - redis  - db | | Rather than running one by one:  docker run -d --name=redis redis  docker run -d --name=db postgres:9.4  **docker run -d --name=vote -p 5000:80 --link redis:redis voting-app**  docker run -d --name=result -p 5001:80 --link db:db result-app  docker run -d --name=worker --link db:db --link redis:redis workers  - Note that here,  db:db == db | |
| **OR**  …  **build: ./vote**  … | | **\*\* Instruct Docker to do a build instead of pulling an image**  - When image is your own application  - Use file path to application source code and *Dockerfile* | |
| **docker compose format (VER 2)** | |
| - No longer display stack information as directly as before (all encapsulated in *services* section)  - Can ndefine networks to use | | | |
| **docker-compose.yml**  **version: ‘2’**  **services:**  redis:  image: redis  **networks:**  **- back-end**  db:  image: postgres:9.4  **networks:**  **- back-end**  **vote**:  **image**: voting-app  **ports**:  - 5000:80  **~~links~~**~~:~~  ~~-redis~~  **depends\_on:**  **-redis**  **networks:**  **- front-end**  **- back-end**  result:  image: result-app  ports:  - 5001:80  ~~links:~~  ~~-db~~  **networks:**  **- front-end**  **- back-end**  worker:  image: worker  ~~links:~~  ~~- redis~~  ~~- db~~  **networks:**  **- back-end**  **networks:**  **front-end:**  **back-end**: | | Shift all of Version 1 under *services*  **1) \*\* Specify version** of docker compose file if using version 2 or higher  - Write version number as **string**  2.1) No need to use **link** in version 2 and higher since docker will automatically create a *dedicated* bridge network for this application  - All containers are attached to this network  2.2) Dedicate networks to use  - **Front-end** network: traffic for users  - **Back-end** network: traffic for applications  3) Add dependencies with **depends\_on** property  - Indicate which container container is dependent on | |
| **docker compose format (VER 3)** | |
| - Similar to Version 2  - Some options removed or added  - Comes with support for *docker swarm* | | | |
| **docker-compose.yml**  **version: ‘3’**  **services:**  redis:  image: redis  db:  image: postgres:9.4  **vote**:  **image**: voting-app  **ports**:  - 5000:80  **~~links~~**~~:~~  ~~-redis~~  **depends\_on:**  **-redis**  result:  image: result-app  ports:  - 5001:80  ~~links:~~  ~~-db~~  worker:  image: worker  ~~links:~~  ~~- redis~~  ~~- db~~ | |  | |
| **Private Registry** | | | |
| **docker login** <registry url> | Log in to private registry | |  |
| **docker run** **<registry url>/apps/<image name>** | Run an image from a private registry | |  |
| **docker run** -d **-p 5000:5000** --name <registry name> **registry:2** | Deploy private registry  - Docker registry is available as an image/application with image name registry  - It exposes the API on port 5000 | |  |
| **docker image tag <**image name> <image file path>  **docker push <image file path>** | Push your own custom image to the private registry | |  |
| **docker pull <image file path/IP>** | Pull image from the same network  - **Method 1**: on the same host  - **Method 2**: using IP/domain of docker host accessing from another environment | |  |
| **Example Command** | | **Explanation** | |
| docker login private-registry.io | | Log in to private registry | |
| docker run private-registry.io/apps/internal-app | |  | |
| docker run -d -p 5000:5000 --name registry registry:2 | | Run custom registry called registry | |
| docker image tag my-image localhost:5000/my-image  docker push localhost:5000/my-image | | Push custom image to private registry  - Image is running on docker host | |
| docker pull localhost:5000/my-image | | Pull image from local docker host | |
| docker pull **192.168.56.100**:500/my-image | | Pull image using **IP address** of docker host | |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |