Docker

This is a compilation of the notes from the sessions taken by Chap to teach the basics of Docker to domain team members. Recordings of the sessions can be accessed here

Session 1: Virtualization vs. Containerization, Basics 1

Containerization: 2

Session 2: Running processes & Dockerfile commands 3

Session 3: Docker-storage-mounts-and-networking-part-1 4

Session 4: Networking part 2 4

Session 5: 5

Some commands 5

Session1: 5

Session2: 6

Session3: 6

Session4: 7

Session5: 7

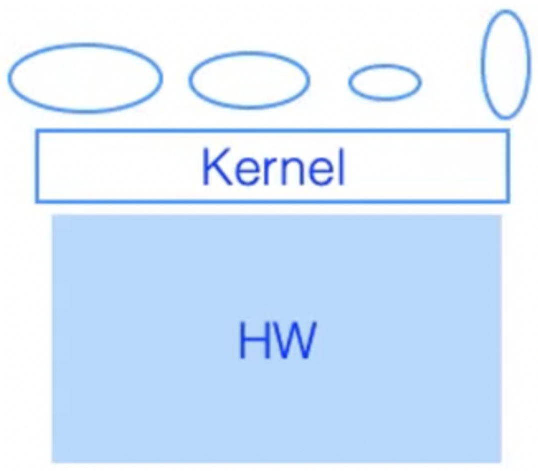
Introduction

This is notes authored by Chanukya Nanduru, from a series of training sessions conducted by Prasad Chodavarapu, originally for scientific team members at Aganitha who wished to develop their technical skills. In hindsight, this material is proving useful to new technical team members as well. You can either read through the content here and wherever you find yourself lost, switch over to the video recording for the corresponding session. Recordings for all sessions are available under training > Tech for Domain team members > Recordings > Docker folder in our google drive. To play a session's recording, click on the mp4 file in the folder for that session.

Each session's title is also hyperlinked to the corresponding video recording.

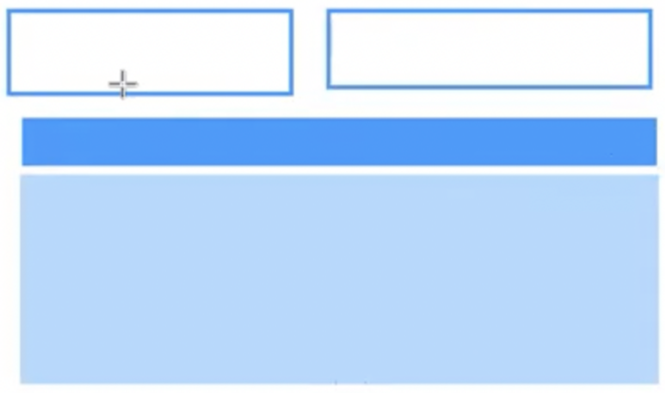
Session 1: Virtualization vs. Containerization, Basics

- Refer DevOps 102>Docker at Aganitha>Guide to Docker at Aganitha
- OS: Kernel + Utilities (e.g., Process Mgmt)
    - Code (Applications) that allows utilisation of hardware resources - Compute, Store, Network
    - Mediator between hardware and software applications
    - Kernel (not a physical entity),:
        - Core of the OS, Gatekeeper to hardware
        - Process management (e.g., Terminate process) & Hardware mediation



Oval shapes represent different sized applications

- Virtual Machine:
  - Driven by need for testing where ‘pure’ machines were required (Desktop-side). *Trivia: VMware company grew on this capability.*
  - Gradually, their value on the server-side was realised. However, different projects or users require different environments.
  - Instead of dedicating under-utilized machines for applications (e.g., Payroll, Tax filing) which have occasional peaks, software layers should be able to access hardware layers as needed (=>Help address global warming). This was the motivation behind virtualization.
  - Alternate to dual-boot environment for having 2 OS. Modern way is to use hypervisor
  - Hypervisor:
    - Enabler for OS to communicate with hardware
    - Previously, one hardware was connected to one kernel. If there are 2 kernels (same or different OS), conflict management is done by hypervisor.



Dark blue box represents Hypervisor

    - Need for 2 kernels can be driven by:
      - Dependency management
      - Standardisation of application packaging
    - Image: Kernel + Application package

        - Issue with VMs: Because Hypervisor is the middle man, it brings in some inefficiency.

Containerization:

- As long as only Linux is used, instead of creating another kernel for another instance of VM, the kernel itself is made capable of isolation (=> No need for hypervisor) i.e., A single kernel can serve multiple VMs/isolated instances.
- The functionality of hypervisor is collapsed into the kernel itself to address the issue with VMs
- Multiple containers can be started on the same kernel but each container has its dedicated space without being aware of the other containers=>Isolation is achieved.

Session 2: Running processes & Dockerfile commands

- Running server process inside a container forever i.e., should not stop when we exit e.g., Computations inside a Jupyter notebook
    - -d for running in the background
    - -it for running interactively (foreground)
    - Attaching to the container => Similar to running interactively
        - If the terminal exits, it will send a 'hang up' to the process running in the container
    - Entering container using *exec* or *run*
    - Ideally, one container should run one service
- Dockerfile commands
    - RUN: Running inside build script to change image. e.g., RUN apt-get…lead to installations during build time
    - CMD: after the container is built during run time/usage
    - LABEL: Attaching key-value pairs to image e.g., version, description
    - ENV: Setting environment variable inside container
    - ADD (&COPY): Adding a file into image (in addition to wget).
        - The file should be in the same folder as the docker file because the context is passed while building the image
        - Executable vs. source for software installation
            - Preferable to remove the source after installation
    - ENTRYPOINT:
        - Used for prepending commands
        - Useful when there is a constant part (shell script/environment setup) and variable part (command)
    - USER:
        - Creating user within the container/image RUN adduser <username>
        - Any command RUN after that will depend on privileges
    - WORKDIR:
        - Can be set during docker run
        - Better way is to specify in docker file
    - ARG:
        - Parameterizing docker file e.g., ubuntu version
        - Any argument used before FOR is applicable only for the first line.
        - If required to be used again, it has to be specified again
    - VOLUME:

- Expose: Related to networking
- Daemon process:
    - Runs forever
- Zombie process
    - Parent process died leaving the child process. Such child processes report ot Init
    - Init is a daemon process (i.e., running forever)
    - Tini
- IDs less than 1000=> System users

Session 3: Docker-storage-mounts-and-networking-part-1

- Though isolation w.r.t. runtime & dependencies is achieved through containers, it is important to enable interactions for a container (i.e., keep the traffic flowing in and out of a container w.r.t. files)
    - Storage
        - docker cp works only if container is alive
        - But, many applications which would have produced files in an image may be needed even after the container is down.
            - Mounting helps here. Similar to mounting a USB storage device to a laptop, a folder is mounted on the container
                - Named volumes: helpful when same directory is to be shared across users.
                - Absolute path
    - Network
        - Interacting with applications inside the container
            - Every container has an IP address
                - But that IP address can be a private IP address. So, we use Traefik (Ingress controller, ReverseProxy) to manage this(Described here).
            - Application installed within the container will have a port number
        - Refer Networking summary

Session 4: Networking part 2

- netcat (nc)
    - Can talk to any application using nc
        - Assuming we know what protocol that application accepts
    - can create server or create client application
    - can be used to connect to container for which the port should be exposed
    - curl : client specifically for http protocol
        - 302 is a redirect response
        - 401: Authorization required
- Exposing a port # in docker file
    - EXPOSE 80 in docker file
    - - p 80 (-p indicates publish) in docker run command
    - Only root users can start applications that use reserve ports e.g., 0-1024
- Finding IP address of container
    - Using inet utils & IP address show
    - docker inspect

- Issue: Some IP addresses can be private and also IP address changes every time a container is started.
    - So, we need help to connect to the container's IP addresses when we want to connect from outside. This is where Ingress controller or reverse proxy. e.g., Traefik (running on a specific port, 80 or 43, in own3) comes in.
    - Traefik basically listens to every container event such as start and stop and directs traffic.
    - Labels are used to indicate to Traefik the traffic to be diverted to it. Host names are designated in the labels based on which Traefik determines which container to route the traffic to
- Docker .cfg (config) file: atk has to be installed on server
        - Image name, container name and description can be filled.
        - Environmental variables can be specified in a separate file or added
        - Domains to be supported can be specified
        - Mount points can be specified
        - Detach or interactive mode can be specified
        - Autoremove setting can be specified
        - Restart policy can be specified
        - Labels can be added
        - Traefik can be enabled
        - Authentication enabling can be done
            - htpasswd
            - LDAP authentication
        - Container ports & host ports to be enabled
            - Note that 8001:8000 indicates that 8000 port on container it is bound to a host port 8001
            - 8001->8000/tcp indicates the same as above
            - You can directly connect to the port in the container or bind that port in the container to the host and go via the host route to the container port
- While connecting from outside, the port of the host should be given. Traefik will reroute to the container

Session 5:

- PID2Container : Utility written to know details of the PID run by a container. Custom written at Aganitha
- If required on VMs, it is to be copied from github

GPU access

- If application is inside docker container
    - Host (e.g., own3) should have : nvidia-driver-515
        - nvidia-container-runtime enables using all GPUs mounted on the machine
        - Including -- *gpus all* in the docker run command specifies the GPUs to be exposed
        - nvidia -smi shows the GPU details and usage. Can be run within the container also

    - cuda is a framework to make C programs access GPU. Cuda macros enable the routines to be portable to GPU. e.g., GROMACS if compiled with Cuda will be able to access GPU

Some commands

Session1:

- vi Dockerfile
    - Opens a docker file where we can mention
        - FROM
        - Run apt-get update etc.
- docker build -t <imagename> .
    - . => Context is the same directory in which the docker file is being created
    - To build an image
    - It will pick up from cache if it was built previously
- apt get install -y
    - 'Yes' as a response
- docker images | grep <keyword>
    - To view the docker images matching with the <keyword>
- docker run -it <imagename> bash
    - Shell prompt inside it
    - -it => interactive
        - exit in interactive mode leads to closure of container
- docker ps -a | grep <keyword>
    - Checking the status of the container
    - Checking the remnants
    - ps - a => to show the exited ones as well
    - traefik can be a keyword
- docker rm <imagename>
    - For troubleshooting docker keeps the logs of a dead container
    - This command is to fully remove the remnants of the container
- docker stop <imagename>
    - Stopping the container
- docker run -- name <containername> -- rm -it <imagename>
    - rm to remove without leaving remnants
    - -it : interact
- touch <filename>
    - if vi is installed in host, it is not available inside the container
- docker run -- name <containername> -- rm -v $PWD:/home/<foldername>-it <imagename>
    - Mounts present working directory (PWD) as /home/<foldername> inside the container
    - -v requires host path should start with /.
    - If not it is interpreted as a named volume(Details here).
- docker volume ls
    - List of named volumes

Session2:

- cat Dockerfile
    - Opens the file which shows base image name (FROM) & additional commands that modify the state of image
- docker run --name <name> -d <container> *//To be used to start a container & run*
    - to name the <container> as <name>
    - -d : runs in the background and gives an ID
    - Interact with container using name or ID given when -d is used
- docker exec -it <containername> <command> *//To be used with a running container*
    - Running <command> inside the container e.g., ps, ps aux,
- docker logs -f <containername>
    - -f : follows =>shows log messages as & when they appear
- python -m http server
    - Starts http server
- docker attach <imagename>
- docker run -w <working directory path>
- docker build -- build-arg <ARG>=

Session3:

- docker cp <image name: folder path/filename .
    - To copy file to a folder in an image
- docker cp filename folder path
    - To copy file from a folder in an image to a specified path
- -v $PWD:/
    - For mounting
    - First argument should always start with path i.e., /.
    - If not, it is not interpreted as a file path. It is interpreted as named volume
- docker ps | wc -l
    - # docker containers running
- docker exec -u root <container name> bash
    - Executing as root user in a container
- inetutils-ifconfig
    - Need to instal inetutils for this command to work
- route -n
    - routing table is displayed
- telnet <IP address> <Port #>
    -

Session4:

- which nc
    - shows the netcat installation path
- nc - l 7865
    - listens on the specified port number
- nc localhost 7865
    - without -l it serves a client

- nc www.google.com 80
  - Can talk to a google server using HTTP protocol e.g., HEAD / HTTP/1.0
- curl -v http://www.google.com
  - shows the http conversations
- docker inspect <container_name>
  - Reveals container of information along with IP address of container
- ping <IP address>
- nc <IP address> <port #>
  - To check if we can connect
- echo $?
  - exit code = 0 is success. Remaining values indicate failure
  - $? is an implicit variable populated by shell that reflects the status of the last command
- docker logs <container name>
- atk-gen-docker-config.sh
  - will generate docker.cfg
  - adding -a (additional) at the end enables passing parameters that are not part of the configuration file
  - -v will show the translation from .cfg file to docker file

Session5:

- pid2container.sh <PID>
  - If that process is running within a container, the output shows the details of the container