Chapter 3:

Sudo podman logs name → container logs

1. Sudo podman run image|container → creats a new container from an image. (registry; container image)
2. Sudo podman ps → displays the container ID & names → unique
3. Sudo podman run --name — image|container → defines the container name explicitly
4. Sudo podman run --name — -d image|container → runs in detached mode [in background]
5. Sudo podman run image|container /s|tmp (any command) → overrides the entry point
6. Sudo podman run -it image|container /bin/bash → starts an interactive shell
7. Sudo podman exec container[ID or name] → starts a process inside an already running container (-l)
8. Sudo podman ps → lists running containers
9. Sudo podman ps -a → lists all containers including stopped ones
10. Sudo podman inspect container → lists metadata about running or stopped container
11. Sudo podman inspect \
-f '{{.NetworkSettings.IPAddress}}' container (name) → retrives only the IP address
12. Sudo podman stop container → stops running container (-a)=all; kill (-s SIGkill)
13. Sudo podman restart container → restarts a stopped container by reusing old data
14. Sudo podman rm container → deletes a container (-f)=kill (-a)=all
15. Sudo podman ps --format="{{.ID}} ..." → formated list
16. Sudo podman pull container image → pulls container image

Stop → delete

*Set up host directory:

1. Sudo mkdir /var/dbfiles (-pv) container_file_t → creat directory
2. Sudo chown -Rv UID:UID /var/dbfiles → change ownership of file to UID
3. Sudo semanage fcontext -a -t file 'var/dbfiles(/.*)?' → apply the file context to the directory
4. Sudo restorecon -Rv /var/dbfiles (verify → ls -lbaz) → apply the container policy

* Mounting a volume:

Sudo podman run -v file:container Storage path container image

17. Sudo podman run -d --name=x -p IPaddress:host port container image → allow external access to container (Port; specific IP)
18. Sudo podman port container image → see the port assigned

*verify that DB is loaded successfully:

1. Sudo podman exec -it container image/ → non-interactive shell
2. mysql -uuser1 -h (IP) -pmypass 55\ -P port → port forwarding (IP → podman inspect)
3. Sudo podman exec -it container name, /bin/bash
mysql -uroot items -e "SELECT * FROM item" → interactive shell
19. mysql -uroot → connect to the MySql server
show databases;

Chapter 4:

1. Sudo podman search [option] <term> ~> finds images from /etc/containers/registries.conf
   - i) --limit <number> ~> number of images per registry
   - ii) --filter <filter=value>
     - stars = <number> ~> number of stars
     - is-automated = <true/false> ~> images automatically built
     - is-official = <true/false> ~> flagged as official
   - iii) --tls-verify <true/false> ~> Enables or disables HTTPS.
2. curl -Ls https://<name>/v2/-catalog?n=# | python -m json.tool ~> list all repositories available in registry
   (v2/ = endpoint)
3. curl -Ls https://<register>/v2/<name>/tags/list ~> list of tags available for a single image
4. Sudo podman login -u username -p password ~> requires access authorization
5. Sudo podman pull <registry>/<name>/<container> ~> pull image from registry
6. Sudo podman images ~> list all container images stored locally
7. Sudo podman pull (podman) rhscl/mysql-57-rhel7:5.7 ~> pull specific tag (5.7 = version (tag))
8. Sudo podman save [-o FILE_NAME] IMAGE_NAME [:TAG] ~> save image as .tar file
7. Sudo podman load [-i FILE_NAME] ~> restore the container image
10. Sudo podman rmi [options] IMAGE [IMAGE...] ~> delete an image
    - sudo podman rmi --force (-a = all) or stop -> rmi ~> force the removal
11. Sudo podman commit [OPTIONS] container [REPOSITORY[:PORT]/]IMAGE_NAME[:TAG]
    - i) --author = -a ~> who created the container image
    - ii) --message ~> commit message
    - iii) --format ~> format of the image
12. Sudo podman diff container image ~> identify what is changed
13. Sudo podman tag [OPTIONS] IMAGE[:TAG] [REGISTRY HOST/][USERNAME/] NAME[:TAG] ~> tag an image
    (NAME = dirname)
14. Sudo podman rmi [-]/[:TAG] ~> remove tag
15. Sudo podman push [OPTIONS] IMAGE ~> push image
16. curl http://localhost:8800 ~> attempt to access localhost
    (8800 = port)

[registry.search]

[registry.insecure]

pter6:

(1) oc login <clusterUrl> -u username -p password API
→ Most operations require a logged-in user.

(2) oc port-forward ~~pod~~ 3306:3306
→ port forwarding

(3) oc new-app -o json or -o yaml
→ to create a skeleton resource definition file

(4) oc create -f <filename>
→ create an application

(5) oc new-app α --as-deployment-config
→ create deployment config resources

(6) oc new-app --docker-image = α --name=α
→ private docker image registry

(7) oc new-app https://github.com/α/α
→ code stored in a git repository

(8) oc get Resource-type [-o json | -o yaml]
→ displays a summary of all resources of the specified type

(9) oc get all
→ retrieve a summary of the most components.

(10) oc describe Resource-type Resource-name
→ retrive additional information

(11) oc edit
→ edit resources of a resource definition

(12) oc delete Resource-type name
→ removes a resource from an Openshift cluster

(13) oc exec
→ executes commands inside a container

(14) oc get α -l app=α
→ -l: acts as a selector.

(15) oc new-project α
→ create a new project

(16) oc status
→ view status of the new application

(17) oc expose Resource-type Resource-name
→ expose for external access

(18) oc get pod -w
→ monitor the progress

(19) oc logs -f α
→ monitor logs of (bc/dc)

(20) oc get is -n openshift
→ available image streams

(21) oc new-app --as-deployment-config/
php~https://my.git --name = myapp
\-i php
→ Building an application

(22) oc get builds
→ list of application builds

(23) oc logs build/myapp-1
→ shows the last few lines of the build log

(24) oc get build config
oc start-build myapp
→ trigger new build

(25) oc new-project α
→ create new project

(26) oc get svc
→ list services

(27) oc logs -f bc/α
oc logs -f dc/α
] view logs of bc or dc

(28) oc expose SV/α \ --name = α
→ create a route with a name

(29) oc logs --all-containers
→ logs of containers

(30) oc get route -o jsonpath = '{..spec.host}{"/n"}'
oc get route/α
oc get route -o yaml
] URL of deployment
--context-dir ? → git path is available

pter 5:

Docker file instruction examples:

FROM α	~> Specify the base image
LABEL description = "α"	~> Add generic metadata
MANTAINER name <email>	~> indicates the author
RUN α	~> executes commands in a new layer
EXPOSE α	~> listens on the specified network port
ENV α	~> defines environment variables
ADD α	~> copies files or folders from a local or remote source (URL) and adds them to the container's file syste
COPY α	~> copies files from the working directory and adds them to the container's file system (no tar files)
USER α	~> specifies the username
ENTRYPOINT α	~> specifies the default command to execute
CMD α	~> provides the default arguments for ENTRYPOIN

chapter 7:

(1) oc get templates -n openshift [| grep persistent] → list preinstalled templates
(2) oc get template x -n openshift -o yaml → show a yaml template definition
(3) oc create -f x.yaml [-n openshift] → publish the application template (public) / create the application
(4) oc describe template x -n openshift
(5) oc process --parameters x -n openshift] → list available parameters from a template
(6) oc process -f <file name> → process a template
oc create -f → -o yaml -p to override a parameter]
(7) oc get pv → list the persistence volume objects
(8) oc get pv x -o yaml → see yaml definition
(9) oc create -f x.yaml → add more persistence volume
(10) oc create -f pvc.yaml → create pvc
(11) oc get pvc → list persistence volume claim

* oc get template > redirection [export] parameters & create
* oc new-app --template = x

chapter 8:

(1) oc log bc/<application name> → retrieve the logs from a build configuration
(2) oc start-build <application name> → request a new build
(3) oc log dc/<application name> → retrieve the logs from a deployment config.
(4) oc adm policy add-scc-to-user anyid -z default → execute container processes with non-root users.
(5) oc port-forward x [ ]:[ ] (local : pod) → port forward to specific port number
(6) podman logs <container name> → retrieve the output of running container
(7) oc get events → read openshift events
(8) oc delete pv <pv-name> → delete persistent volume
(9) oc create -f <pv-resource-file> → create persistent volume
(10) oc adm prune → automated way to remove obsolete images
(11) oc logs <podName> [-c <containerName>] → returns the output for a container within a pod
(12) oc exec -it x /bin/bash → execute a single interactive shell
(13) sudo podman cp (file) x <containerName>:path
sudo podman cp <containerName>:path .] → copy

git commit -am "__"
git push