CHAPTER 1

OVERVIEW OF CONTAINER TECHNOLOGY

Are there other competing container formats?

- lxd, used by lxc runtime   https://linuxcontainers.org/
- aci from appc
  https://github.com/appc/spec/blob/master/spec/aci.md#app-container-image
    - used by rkt from coreos
- docker (v1) deprecated 2/28/2017
  https://github.com/moby/moby/blob/master/image/spec/v1.md
- docker (2v1) overly complicated b/c of backwards compat with docker v1
  https://github.com/docker/distribution/blob/master/docs/spec/manifest-v2-1.md
- docker (2v2)
  https://github.com/docker/distribution/blob/master/docs/spec/manifest-v2-2.md
- oci (originally based on docker 2v2)
  https://github.com/opencontainers/image-spec/blob/master/spec.md

Consider the format used in one of the containers used in kubernetes:

```
$ skopeo inspect docker://k8s.gcr.io/pause
$ podman pull k8s.gcr.io/pause:latest
$ podman inspect k8s.gcr.io/pause:latest | grep Manifest
        "ManifestType": "application/vnd.docker.distribution.manifest.v2+json",
```

Consider what the registries used by cloud vendors support:

Google cloud: https://cloud.google.com/container-registry/docs/image-formats

Azure:
https://docs.microsoft.com/en-us/azure/container-registry/container-registry-image-formats

aws: https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-manifest-formats.html

OVERVIEW OF CONTAINER (RUNTIME) ARCHITECTURE

What are namespaces ?

- Innovations that followed include:
    - **Namespaces** - responsible for resource isolation
        - Think of the view-master !

- man 7 namespaces, nsenter
    - “”A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource””
- /proc/$$/ns/
    - Uts: hostname (uname -n)
    - Ipc: interprocess communications, shared memory (ipcs)
    - Net: ipv4/ipv6 stacks, routing, firewall, sockets
    - Pid: process id namespace (allows for migrating containers to new host, suspend/resume)
    - User: distinct uids and gids
    - Cgroup: cgroup root directory, prevents process *X* from escaping the limits imposed by ancestor cgroups

https://www.redhat.com/sysadmin/pid-namespace

[student@workstation ~]$ **unshare -Urpf --mount-proc**

```
[root@workstation ~]# sleep 9000 &
[1] 32
[root@workstation ~]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root         1      0  0 11:48 pts/1       00:00:00 -bash
root         32     1  0 11:49 pts/1       00:00:00 sleep 9000
root         33     1  0 11:49 pts/1       00:00:00 ps -ef
```

(from a different terminal)

```
[student@workstation ~]$ ps -ef | grep 9000
student      4497   4466  0 11:49 pts/1    00:00:00 sleep 9000
student      4583   4414  0 11:51 pts/2    00:00:00 grep --color=auto 9000
```

(notice the different pid for this same sleep process)

```
[student@workstation ~]$ sudo nsenter -t 4497 -a
-bash: /root/.bash_profile: Permission denied
[root@workstation /]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root         1      0  0 11:48 pts/1       00:00:00 -bash
root         32     1  0 11:49 pts/1       00:00:00 sleep 9000
root         34     0  3 11:51 pts/2       00:00:00 -bash
root         61     34  0 11:51 pts/2      00:00:00 ps -ef
```

What are control groups ?

- **Control groups** (cgroups) - limits what resources a process group can consume. Allows processes to be organized into hierarchical groups so that usage of particular resources can be limited and monitored.

Recently, distros like Fedora31 have made cgroups V2 the default. See https://www.redhat.com/sysadmin/fedora-31-control-group-v2

```
[root@workstation ~]# cd /sys/fs/cgroup/pids/system.slice/sssd.service
[root@workstation sssd.service]# ls
cgroup.clone_children  cgroup.procs  notify_on_release  pids.current  pids.events  pids.max
tasks
[root@workstation sssd.service]# cat pids.max
36445
```

[root@workstation sssd.service]# **cat pids.current**
3

- NOTE: memory accounting from /proc/meminfo is NOT namespaced. So, a container's view of memory from tools like free/top will show the system accounting vs /sys/fs/cgroup/memory/memory.usage_in_bytes
  - https://ops.tips/blog/why-top-inside-container-wrong-memory/

What is seccomp?

- **Seccomp** - limits what system calls a process can make… even if running as root ! "secure computing mode"
  - /proc/sys/kernel/seccomp/actions_avail
  - /proc/sys/kernel/seccomp/actions_logged
  - man 2 seccomp {fedora has this one}
    - Limits what system calls by either read(), write(), _exit(), and sigreturn() or by a list of allowed calls given as "filters"
    - The Seccomp field of the /proc/[pid]/status file provides a method of viewing the seccomp mode of a process
    - [student@workstation ~]$ **cat /proc/3421/status | grep -i seccomp**
    - Seccomp: 0
      - 0 means SECCOMP_MODE_DISABLED; 1 means SECCOMP_MODE_STRICT; 2 means SECCOMP_MODE_FILTER
        - In SECCOMP_MODE_STRICT, it cannot use any system calls except exit(), sigreturn(), read() and write().
        - In SECCOMP_MODE_FILTER, since linux 3.5, it is possible to define advanced custom filters based on the BPF (Berkley Packet Filters) to limit what system calls and their arguments can be used by the process.
    - podman run --security-opt=seccomp=unconfined
    - Will use **/usr/share/containers/seccomp.json** as the default profile

What are capabilities?

- **Capabilities** -A related and additionally important feature are Capabilities.
  - **man 7 capabilities** “Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. Capabilities are a per-thread attribute.”

```
[student@workstation ~]$ ps -ef | grep sshd
root      1049    1  0 08:11 ?     00:00:00 /usr/sbin/sshd -D
[student@workstation ~]$ grep Cap /proc/1049/status
CapInh:    0000000000000000
CapPrm:    000001ffffffffff
CapEff:    000001ffffffffff
CapBnd:    000001ffffffffff
CapAmb:    0000000000000000
[student@workstation ~]$ capsh --decode=000001ffffffffff
0x000001ffffffffff=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,c
ap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net
_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sy
s_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys
_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_wr
ite,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_al
arm,cap_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore
```

CapPrm = Permitted Capabilities
CapBnd = Bounding Capabilities
CapEff = Effective Capabilities

Capabilities can also be assigned to a file binary (getcap, setcap):

```
[student@workstation sbin]$ getcap /sbin/arping
/sbin/arping cap_net_raw=p
```

- e: Effective
  This means the capability is “activated”.

- p: Permitted
  This means the capability can be used/is allowed.

- i: Inherited
  The capability is kept by child/subprocesses upon execve() for example

What container specific selinux labels are used ?

- **SELinux** - protects processes from each other since they will all be running on the host system
  - ps -Z

```
[student@workstation ~]$ cat /usr/share/containers/selinux/contexts
process = "system_u:system_r:container_t:s0"
file = "system_u:object_r:container_file_t:s0"
ro_file="system_u:object_r:container_ro_file_t:s0"
kvm_process = "system_u:system_r:container_kvm_t:s0"
init_process = "system_u:system_r:container_init_t:s0"
engine_process = "system_u:system_r:container_engine_t:s0"
```

What happened to svirt_sandbox_file_t ?

```
[root@workstation ~]# mkdir /testdir
[root@workstation ~]# chcon -t svirt_sandbox_file_t /testdir
[root@workstation ~]# podman run -it -v /testdir:/data rhel7 /bin/bash
[root@a8b47cb39617 /]# touch /data/test1
```

```
# rpm -q selinux-policy
```

```
selinux-policy-3.13.1-229.el7.noarch
```

```
# yum install setools-console
```

```
# seinfo -tcontainer_file_t -x
   container_file_t
         device_node
         file_type
         filesystem_type
         mountpoint
         non_auth_file_type
         non_security_file_type
         noxattrfs
         ptynode
   Aliases
         svirt_sandbox_file_t
         svirt_lxc_file_t
```

""
Aliases are alternate names used to refer to a type. We can use an alias anywhere that we would use a type name, including TE rules, security contexts, and labeling statements. Aliases are typically used for compatibility when making policy changes. For example, an older policy might refer to the type netscape_t. An updated policy might switch to the type name to mozilla_t, but provide netscape_t as an alias to allow older modules to correctly compile.
""""

https://flylib.com/books/en/2.803.1.40/1/

https://danwalsh.livejournal.com/81756.html

What other selinux target contexts are allowed by selinux policy ?

```
[student@workstation ~]$ sudo yum install selinux-policy-doc
```

```
[student@workstation ~]$ man container_selinux

MANAGED FILES
        The SELinux process type container_t can manage files labeled with the following file
types:

        cephfs_t
        cifs_t
        container_file_t
        fusefs_t
        hugetlbfs_t
        nfs_t
        onload_fs_t

[student@workstation ~]$ podman run -d -p 8080:8080
registry.redhat.io/rhscl/httpd-24-rhel7
[student@workstation ~]$ podman exec -it 8b /bin/bash
bash-4.2$ cd /var/www/html/
bash-4.2$ echo "helloworld" > index.html
bash-4.2$ ls -lZ
-rw-r--r--. default root system_u:object_r:fusefs_t:s0          index.html

bash-4.2$ ps -efZ
LABEL                          UID              PID      PPID  C STIME TTY             TIME CMD
system_u:system_r:container_t:s0:c468,c532 default 1   0  0 16:38 ?        00:00:00 httpd -D
FOREGROUND
system_u:system_r:container_t:s0:c468,c532 default 40  1  0 16:38 ?        00:00:00 /usr/bin/cat
system_u:system_r:container_t:s0:c468,c532 default 41  1  0 16:38 ?        00:00:00 /usr/bin/cat
system_u:system_r:container_t:s0:c468,c532 default 42  1  0 16:38 ?        00:00:00 /usr/bin/cat
system_u:system_r:container_t:s0:c468,c532 default 43  1  0 16:38 ?        00:00:00 /usr/bin/cat
system_u:system_r:container_t:s0:c468,c532 default 44  1  0 16:38 ?        00:00:00 httpd -D
FOREGROUND
system_u:system_r:container_t:s0:c468,c532 default 45  1  0 16:38 ?        00:00:00 httpd -D
FOREGROUND
system_u:system_r:container_t:s0:c468,c532 default 52  1  0 16:38 ?        00:00:00 httpd -D
FOREGROUND
system_u:system_r:container_t:s0:c468,c532 default 67  1  0 16:38 ?        00:00:00 httpd -D
FOREGROUND
system_u:system_r:container_t:s0:c468,c532 default 69  1  0 16:38 ?        00:00:00 httpd -D
FOREGROUND
system_u:system_r:container_t:s0:c468,c532 default 102 0  0 16:48 pts/0 00:00:00 /bin/bash
system_u:system_r:container_t:s0:c468,c532 default 111 102  0 16:49 pts/0  00:00:00 ps -efZ
```

```
[student@workstation ~]$ curl localhost:8080
helloworld

[student@workstation ~]$ podman rm -a -f
```

MANAGING CONTAINERS WITH PODMAN

How to install podman and container-tools in RHEL ?

```
[root@rhel7 ~]# subscription-manager repos --enable rhel-7-server-extras-rpms
[root@rhel7 ~]# yum install podman
```

This training environment uses RHEL8.2 on workstation VM:

```
[root@workstation ~]# subscription-manager repos --list-enabled
This system has no repositories available through subscriptions.

[root@workstation ~]# yum repolist
Updating Subscription Management repositories.
Unable to read consumer identity
This system is not registered to Red Hat Subscription Management. You can use
subscription-manager to register.
repo id                                          repo name
rhel-8.2-for-x86_64-appstream-rpms                     Red Hat Enterprise Linux 8.2
AppStream (dvd)
rhel-8.2-for-x86_64-baseos-rpms                        Red Hat Enterprise Linux 8.2
BaseOS (dvd)
```

(DONT ACTUALLY RUN THE NEXT COMMAND)

```
[root@rhel8 ~]# yum module list container-tools
Updating Subscription Management repositories.
Last metadata expiration check: 0:00:48 ago on Mon 19 Aug 2019 10:51:59 AM EDT.
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
Name                Stream              Profiles            Summary
```

```
container-tools                 1.0                 common [d]                 Common tools
and dependencies for container runtimes
container-tools                 rhel8 [d][e]        common [d] [i]             Common tools
and dependencies for container runtimes

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

```
[root@workstation ~]# yum module info container-tools

              : buildah-0:1.11.6-7.module+el8.2.0+5856+b8046c6d.x86_64
              : conmon-2:2.0.6-1.module+el8.2.0+5182+3136e5d4.x86_64
              : podman-0:1.6.4-10.module+el8.2.0+6063+e761893a.src
              : runc-0:1.0.0-65.rc10.module+el8.2.0+5762+aaee29fb.src
              : skopeo-1:0.1.40-10.module+el8.2.0+5955+6cd70ceb.src
              : slirp4netns-0:0.4.2-3.git21fdece.module+el8.2.0+5658+9a15711d.src
```

Is there a module or group that can assist installation in Fedora ?

```
[root@badger ~]# cat /etc/redhat-release
Fedora release 34 (Thirty Four)

[root@badger ~]# dnf repolist
repo id                         repo name
fedora                          Fedora 34 - x86_64
fedora-cisco-openh264               Fedora 34 openh264 (From Cisco) - x86_64
fedora-modular                      Fedora Modular 34 - x86_64
google-chrome                       google-chrome
rpmfusion-free                  RPM Fusion for Fedora 34 - Free
rpmfusion-free-updates              RPM Fusion for Fedora 34 - Free - Updates
updates                         Fedora 34 - x86_64 - Updates
updates-modular                     Fedora Modular 34 - x86_64 - Updates

[root@badger ~]# dnf install -y @container-tools
Last metadata expiration check: 1:17:32 ago on Tue 21 Sep 2021 05:29:15 AM CDT.
Module or Group 'container-tools' is not available.
Error: Nothing to do.
```

Although there is no container-tools group or module in Fedora, there is a container-management group:

```
[root@badger ~]# dnf groupinfo "Container Management"
Last metadata expiration check: 1:16:42 ago on Tue 21 Sep 2021 05:29:15 AM CDT.
Group: Container Management
 Description: Tools for managing Linux containers
 Default Packages:
   podman
 Optional Packages:
   buildah
   flatpak
   flatpak-builder
   origin-clients

[root@badger ~]# dnf install -y @container-management
```

CHAPTER 2 CREATING CONTAINERIZED SERVICES

• Search for and fetch container images with Podman.
• Run and configure containers locally.
• Use the Red Hat Container Catalog.

FETCHING CONTAINER IMAGES WITH PODMAN

How can you determine the available tags within a repository ?

```
$ podman search registry.access.redhat.com/rhel7 --list-tags
```

(but this only includes a limited list of tags)

Use skopeo:

```
[student@workstation ~]$ skopeo inspect docker://registry.access.redhat.com/rhel7 | grep 7.5
        "7.5-424",
        "7.5-245.1527091554",
        "7.5-409.1533127727",
        "7.5",
        "7.5-231",
        "7.5-404",
        "7.5-433",
        "7.5-245",
        "7.5-409",
        "7.7-529",
```

Alternatively,

```
[student@workstation ~]$ curl  -L https://registry.access.redhat.com/v2/rhel7/tags/list
{"name": "rhel7", "tags": ["7.3-74", "7.4-120", "7.2-56", "7.3-89", "7.3-66", "7.5-424",
"7.5-245.1527091554", "7.4-129", "7.1-12", "7.6-122", "7.3-82", "7.7-384.1575996163",
"7.5-409.1533127727", "7.2-75", "7.2-38", "7.6", "7.7-348", "7.4", "7.5", "7.6-301.1561066494",
"7.4-164", "7.7", "7.8", "7.2-35", "7.7-269", "7.1-6", "7.6-122.1547747894", "7.5-231", "7.5-404",
"7.1-9", "7.1-24", "7.4-81", "7.6-362", "7.6-252", "7.6-202.1553789841", "7.2-104", "7.3-97",
"7.4-113", "7.7-384.1580117710", "7.6-119", "7.1-16", "7.1-4", "7.6-301", "7.5-433", "7.0-27",
"7.3-95", "7.6-115", "7.3-79", "7.0-21", "7.8-265", "7.4-152", "7.7-310", "7.5-245",
"7.6-252.1561619826", "7.3-53", "7.0-23", "7.7-481", "7.6-151.1550575774", "7.3", "7.2",
"7.4-105", "7.6-202.1554729462", "7.3-45", "7.5-409", "7.2-46", "7.6-151", "7.2-44", "7.2-43",
"7.1-11", "7.7-529", "7.2-61", "latest", "7.7-384", "7.2-84", "7.6-202"]}
```

We can make that prettier with json_reformat:

```
[[student@workstation ~]$ curl -L https://registry.access.redhat.com/v2/rhel7/tags/list | json_reformat
```

```
[student@workstation ~]$  curl -L https://registry.access.redhat.com/v2/rhel7/tags/list | json_reformat | grep 7.7
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   467  100   467    0     0   1040      0 --:--:-- --:--:-- --:--:--  1042
100   901  100   901    0     0   1852      0 --:--:-- --:--:-- --:--:--  1852
        "7.5-424",
        "7.5-245.1527091554",
        "7.5-409.1533127727",
        "7.5",
        "7.5-231",
        "7.5-404",
        "7.5-433",
        "7.5-245",
        "7.5-409",
        "7.7-529",
```

Now that we can identify an image within a repository that we want, let's pull a local copy using the "latest" tag:

```
[student@workstation ~]$  podman pull registry.access.redhat.com/rhel7:latest
9a3387c8f6bc9b63b119dc61ddbaed6bb20795a7b187908ca1b5ecabc5c19aac
```

How does that compare to:

```
[student@workstation ~]$  podman pull registry.access.redhat.com/rhel7
9a3387c8f6bc9b63b119dc61ddbaed6bb20795a7b187908ca1b5ecabc5c19aac
```

(same image id). What about a different version of rhel7?

```
[student@workstation ~]$  podman pull registry.access.redhat.com/rhel7:7.7
6682529ce3faf028687cef4fc6ffb30f51a1eb805b3709d31cb92a54caeb3daf
```

What about trying to pull using a short-name alias ?

Consider:

```
[student@workstation ~]$ head /etc/containers/registries.conf.d/001-rhel-shortnames.conf
[aliases]
"3scale-amp2/3scale-rhel7-operator-metadata" =
"registry.redhat.io/3scale-amp2/3scale-rhel7-operator-metadata"
"3scale-amp2/3scale-rhel7-operator" = "registry.redhat.io/3scale-amp2/3scale-rhel7-operator"
"3scale-amp24/wildcard-router" = "registry.redhat.io/3scale-amp24/wildcard-router
```

```
[student@workstation ~]$ grep rhel7
/etc/containers/registries.conf.d/001-rhel-shortnames.conf
"rhel7/open-vm-tools" = "registry.access.redhat.com/rhel7/open-vm-tools"
"rhel7" = "registry.access.redhat.com/rhel7"
"rhel7/rhel-atomic" = "registry.access.redhat.com/rhel7/rhel-atomic"
```

```
"rhel7/rhel" = "registry.access.redhat.com/rhel7/rhel"

[student@workstation ~]$ podman pull rhel7:latest
Trying to pull registry.access.redhat.com/rhel7:latest...
Getting image source signatures

[student@workstation ~]$  podman images
REPOSITORY                       TAG     IMAGE ID       CREATED        SIZE
registry.access.redhat.com/rhel7  latest        2664aa19856f  2 weeks ago  216 MB
registry.access.redhat.com/rhel7  7.7           6682529ce3fa  22 months ago  215 MB
```

RUNNING CONTAINERS

How to run my first container

```
[student@workstation ~]$ podman run 9a3387c8f6bc cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.9 (Maipo)

[student@workstation ~]$ podman run e64297b706b7 cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.5 (Maipo)
```

What about running a container we haven't "pulled" yet ?

```
[student@workstation ~]$ podman run rhel7:7.7 cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.7 (Maipo)
```

Podman run is also capable of pulling images that aren't available in the local image storage:

```
[student@workstation ~]$ podman run rhel7:7.8 cat /etc/redhat-release
Resolved "rhel7" as an alias (/etc/containers/registries.conf.d/001-rhel-shortnames.conf)
Trying to pull registry.access.redhat.com/rhel7:7.8...
Getting image source signatures
Checking if image destination supports signatures
Copying blob b13ffc206103 done
Copying blob 872582724f33 done
Copying config 9da37a6819 done
Writing manifest to image destination
Storing signatures
```

```
Red Hat Enterprise Linux Server release 7.8 (Maipo)
```

Notice we now have several images listed here:

```
[student@workstation ~]$ podman images
REPOSITORY                      TAG          IMAGE ID       CREATED       SIZE
registry.access.redhat.com/rhel7  latest     c7344c9fb18c  3 weeks ago  216 MB
registry.access.redhat.com/rhel7  7.8        9da37a681956  2 years ago  215 MB
registry.access.redhat.com/rhel   7.7        6682529ce3fa  2 years ago  215 MB
registry.access.redhat.com/rhel7  7.7        6682529ce3fa  2 years ago  215 MB
```

Let’s take a closer look at the cat commands that were executed using **podman run:**

```
[student@workstation ~]$  podman run --help
NAME:
  podman run - Run a command in a new container

USAGE:
  podman run [command options] IMAGE [COMMAND [ARG...]]
```

- Containers will exit after command execution. So let’s see what info we can find on those
    - Not in ps output

```
[student@workstation ~]$  ps -ef | grep cat
gdm      4068  3897  0 16:15 ?         00:00:00 /usr/libexec/gsd-print-notifications
root     7313  6645  0 18:30 pts/0     00:00:00 grep --color=auto ca
```

-
    - Use **podman ps** instead:

```
[student@workstation ~]$ podman ps

[student@workstation ~]$ podman ps -a
CONTAINER ID   IMAGE                                   COMMAND              CREATED
STATUS                        PORTS   NAMES                IS INFRA
27a3d872ff92   registry.access.redhat.com/rhel:7.4          cat /etc/redhat-rel...   About a minute
ago   Exited (0) About a minute ago            confident_dubinsky   false
```

```
6905b9e93f0d   registry.access.redhat.com/rhel7:7.5-404   cat /etc/redhat-rel...   6 minutes ago
          Exited (0) 6 minutes ago                 dreamy_panini           false

de3fa91850ee   registry.access.redhat.com/rhel7:latest       cat /etc/redhat-rel...   6 minutes ago
          Exited (0) 6 minutes ago                 competent_bell          false
```

- Some images have prebuilt commands that will be executed if the COMMAND is missing from the podman run:

```
[student@workstation ~]$  podman inspect registry.access.redhat.com/rhel7:latest | less
```

- (notice /bin/bash)
- So what happens when we run this without giving any command:

```
[student@workstation ~]$  podman run registry.access.redhat.com/rhel7:latest
[student@workstation ~]$
```

- Did it fail ? ps -a (no, just non-interactive)

```
[student@workstation ~]$  podman ps -a
CONTAINER ID   IMAGE                                   COMMAND              CREATED
STATUS                  PORTS   NAMES                   IS INFRA
04e62d0683ff   registry.access.redhat.com/rhel7:latest       /bin/bash                5 seconds ago
Exited (0) 4 seconds ago       youthful_benz           false
```

- So, to make it interactive run:

```
[student@workstation ~]$  podman run -i registry.access.redhat.com/rhel7:latest
asdljflkasdjf
/bin/bash: line 3: asdljflkasdjf: command not found

echo hello world
hello world
whoami
root
tty
not a tty
exit
```

- What about a terminal ? Let's add the -t flag

```
[student@workstation ~]$  podman run -it registry.access.redhat.com/rhel7:latest
[root@66f4d93191b7 /]# tty
/dev/pts/0
[root@66f4d93191b7 /]# exit
```

- Another useful option when running a container is the -d or detach, for example:

```
[student@workstation ~]$  podman run -d registry.access.redhat.com/rhel7:latest sleep
5000
829e8264c3f722e047002ebf9bf55b38fcc9b2be3b6f0a2afdfb4088d01a3a7f
[student@workstation ~]$
[student@workstation ~]$  podman ps
CONTAINER ID   IMAGE                          COMMAND     CREATED       STATUS
PORTS   NAMES
829e8264c3f7   registry.access.redhat.com/rhel7:7.5   sleep 5000   4 seconds ago   Up 3
seconds ago              loving_montalcini
```

- Sometimes we will want to pass environment variables for the processes in the container. Do this with -e:

```
[student@workstation ~]$  podman run registry.access.redhat.com/rhel7:latest env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
TERM=xterm
HOSTNAME=497c3c41f75f
container=oci
HOME=/root
[student@workstation ~]$ podman run -e FOO="hello world"
registry.access.redhat.com/rhel7:latest env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
TERM=xterm
HOSTNAME=c730b9772e9a
container=oci
FOO=hello world
HOME=/root
```

- If you'd like to name a container use --name like:

```
[student@workstation ~]$ podman run -d --name mycontainer
registry.access.redhat.com/rhel7:latest sleep 5000
09b5adcfbcc0f894ef2d1782ebe5a28ba78e2bc67901814726b8d3b63f9545b5

[student@workstation ~]$  podman ps
CONTAINER ID   IMAGE                                    COMMAND              CREATED
STATUS                   PORTS   NAMES
09b5adcfbcc0   registry.access.redhat.com/rhel7:7.5                sleep 5000               3
seconds ago   Up 2 seconds ago                mycontainer
```

- To run a new command inside a running container use podman-exec:

```
[student@workstation ~]$  podman exec -it mycontainer /bin/bash
[root@db380c01c168 /]# ps -ef
UID      PID  PPID  C STIME TTY       TIME CMD
```

```
root      1         0  0 18:58 ?     00:00:00 sleep 5000
root      5         0  3 19:00 pts/0         00:00:00 /bin/bash
root      17        5  0 19:00 pts/0         00:00:00 ps -ef

[root@db380c01c168 /]# cat /proc/1/cgroup

[root@07d1eca25e39 /]# grep Cap /proc/1/status
CapInh:    0000000000000000
CapPrm:    00000000800425fb
CapEff:    00000000800425fb
CapBnd:    00000000800425fb
CapAmb:    0000000000000000
[root@07d1eca25e39 /]# grep -i seccomp /proc/1/status
Seccomp:    2



[root@db380c01c168 /]# ps -efZ
LABEL                      UID      PID  PPID  C STIME TTY      TIME CMD
system_u:system_r:container_t:s0:c478,c651 root 1  0  0 18:58 ?            00:00:00 sleep 5000

[root@db380c01c168 /]# ipcs -a
[root@db380c01c168 /]# exit
[student@workstation ~]$
```

How to change the hostname or otherwise identify which container you are running in ?

The hostname will be set to the container id by default:

```
[student@workstation ~]$  podman run -it rhel:latest /bin/bash
[root@560cf16fe847 /]# uname -n
560cf16fe847

[student@workstation ~]$  podman ps -a
CONTAINER ID   IMAGE                                   COMMAND    CREATED      STATUS
PORTS   NAMES
560cf16fe847   registry.access.redhat.com/rhel:latest   /bin/bash   2 minutes ago   Up 2 minutes
ago              kind_elion
```

/run/.containerenv tells us that we are in a container:

```
[root@560cf16fe847 /]# ls -l /run/.containerenv
-rw-r--r--. 1 root root 0 Nov 12 13:44 /run/.containerenv
```

Env tells us what kind:

```
[root@560cf16fe847 /]# env
container=oci
```

Hostname could be set to something however:

```
[student@workstation ~]$  podman run --hostname foo --name foo -it rhel:latest /bin/bash
```

Actually running podman commands in a container wouldnt be an easy thing to allow. Why?

```
[student@workstation ~]$ podman start foo
[student@workstation ~]$  podman exec -it foo /bin/bash
[root@foo /]# podman ps
bash: podman: command not found
```

Consider Nested containers:
https://developers.redhat.com/blog/2019/08/14/best-practices-for-running-buildah-in-a-container#running_buildah_inside_a_container

https://github.com/containers/podman/issues/5188

- To redirect host traffic to a container on a specific port use -p host:container

```
[student@workstation ~]$  man podman-run

        -p, --publish=[]

        Publish a container's port, or range of ports, to the host

        Format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort |
containerPort
```

```
[student@workstation ~]$   podman run -d -p 8080:8080 --name httpd-basic
registry.access.redhat.com/rhscl/httpd-24-rhel7
```

```
...
77f803ed8546027f65f9c4422f9d81d0696ce1de708b1ca9cdebafbb552890e1

[student@workstation ~]$ podman run -d -p 8080:8080 --name httpd-basic
registry.redhat.io/rhscl/httpd-24-rhel7



[student@workstation ~]$  podman ps
CONTAINER ID   IMAGE                              COMMAND              CREATED
STATUS                 PORTS                  NAMES
77f803ed8546   docker.io/library/httpd:2.4         httpd-foreground   30 seconds ago   Up 29
seconds ago   0.0.0.0:8080->80/tcp   httpd-basic
```

Check netstat:

```
[student@workstation ~]$ netstat -tunap | grep 8080
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp6     0         0 :::8080                 :::*               LISTEN         4827/rootlessport



[student@workstation ~]$ curl localhost:8080
        <body>
         <h1>Red Hat Enterprise Linux <strong>Test Page</strong></h1>
```

What's that conmon process ?

Conmon is a monitoring program and communication tool between a container manager (like podman or CRI-O) and an OCI runtime (like runc or crun) for a single container.

```
[student@workstation ~]$ ps -ef | grep httpd
student          3761    3751  0 07:36 ?          00:00:00 httpd -DFOREGROUND
100000           3775    3761  0 07:36 ?          00:00:00 httpd -DFOREGROUND
100000           3776    3761  0 07:36 ?          00:00:00 httpd -DFOREGROUND
100000           3777    3761  0 07:36 ?          00:00:00 httpd -DFOREGROUND
```

```
[student@workstation ~]$ ps -ef | grep 3751
student          3751    1  0 07:36 ?     00:00:00 /usr/bin/conmon --api-version 1 -c
815d8f11fd3c3dc67bf2e6913fcdf6d2a517c3387eed801d89588ca2e7b1e2e1 -u
815d8f11fd3c3dc67bf2e6913fcdf6d2a517c3387eed801d89588ca2e7b1e2e1 -r /usr/bin/runc -b
/home/student/.local/share/containers/storage/overlay-containers/815d8f11fd3c3dc67bf2e6913f
cdf6d2a517c3387eed801d89588ca2e7b1e2e1/userdata
```

Another way to look at this is with pstree:

```
[student@workstation ~]$ pstree

        ├─conmon─┬─httpd───3*[httpd───26*[{httpd}]]
        │        └─{conmon}
```

```
[student@workstation ~]$ /usr/libexec/podman/conmon -h
```

https://github.com/containers/conmon

Can you run an image built from architectures different than the container host ?

No, but you can pull them. See:
https://www.redhat.com/sysadmin/specify-architecture-pulling-podman-images

```
[student@workstation ~]$ podman pull --arch=arm64 registry.access.redhat.com/ubi8
[student@workstation ~]$ podman pull registry.access.redhat.com/ubi8

[student@workstation ~]$ podman images
REPOSITORY                          TAG          IMAGE ID       CREATED        SIZE
registry.access.redhat.com/rhel7/rhel  latest    e2c37c467077  2 weeks ago  216 MB
registry.access.redhat.com/rhel7      7.9     e2c37c467077  2 weeks ago  216 MB
registry.access.redhat.com/rhel7      latest  e2c37c467077  2 weeks ago  216 MB
registry.access.redhat.com/ubi8       latest  d5c70d09f361  3 weeks ago  246 MB
<none>                           <none>       2fd9e1478809  3 weeks ago  225 MB

[student@workstation ~]$ podman run 2fd9e1478809 uname -a
```

```
Linux f1b40336fff3 4.18.0-348.2.1.el8_5.x86_64 #1 SMP Mon Nov 8 13:30:15 EST 2021
x86_64 x86_64 x86_64 GNU/Linux
[student@workstation ~]$ podman run d5c70d09f361 uname -a
standard_init_linux.go:228: exec user process caused: exec format error
```

It is possible to use a qemu process to emulate different architectures. See https://github.com/multiarch/qemu-user-static

USING ROOTLESS CONTAINERS

What are the disadvantages of running rootless containers ?

SEE https://github.com/containers/podman/blob/master/rootless.md

Some subcommands will not work, esp ones that depend on features like cgroups:

```
[student@workstation ~]$ podman pause 382
Error: pause is not supported for rootless containers
```

```
[student@workstation ~]$ podman stats 382
Error: stats is not supported in rootless mode without cgroups v2
```

https://www.redhat.com/sysadmin/behind-scenes-podman

https://opensource.com/article/19/2/how-does-rootless-podman-work

https://indico.cern.ch/event/757415/contributions/3421994/attachments/1855302/3047064/Podman_Rootless_Containers.pdf

Also, networking is handled differently for rootless as a non-root user has limitations on what it can do to the host's network:

```
[student@workstation ~]$ podman run -d -p 808:8080 --name myhttpd
registry.access.redhat.com/rhscl/httpd-24-rhel7
Error: error from slirp4netns while setting up port redirection: map[desc:bad request:
add_hostfwd: slirp_add_hostfwd failed]
```

How to better understand user namespaces?

Root = not different from the host
Rootless = maps user and group IDs to appear to be running under a different ID. Uses a "pause" process

```
[student@workstation ~]$ podman run -it rhel7
[root@5367563cc886 /]# whoami
root
[root@5367563cc886 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

man 7 user_namespaces:

In particular, a process can have a normal unprivileged user ID outside a user namespace while at the same time having a user ID of 0 inside the namespace; in other words, the process has full privileges for operations inside the user namespace, but is unprivileged for operations outside the namespace.

   User and group ID mappings: uid_map and gid_map
When a user namespace is created, it starts out without a mapping of user IDs (group IDs) to the parent user namespace. The /proc/[pid]/uid_map and /proc/[pid]/gid_map files (available since Linux 3.5) expose the mappings for user and group IDs inside the user namespace for the process pid.

Each line in the uid_map file specifies a 1-to-1 mapping of a range of contiguous user IDs between two user namespaces. The first two numbers specify the starting user ID in each of the two user namespaces. The third number specifies the length of the mapped range.

```
[root@5367563cc886 ~]$ cat /proc/self/uid_map
    (start of range)  (parent ns)      (range)
         0                  1000            1
         1                  100000          65536
```

So, the “root” user inside this namespace maps to the user with uid=1000 in the parent namespace (ie the “student” user).

A user with uid=1 in the child namespace would have a uid of 100000 in the parent namespace and increment up from there in the respective namespaces:

```
[root@c0ec71b5ed9e /]# cat /etc/passwd
[root@c0ec71b5ed9e /]# id 1
uid=1(bin) gid=1(bin) groups=1(bin)
[root@c0ec71b5ed9e /]# id 2
uid=2(daemon) gid=2(daemon) groups=2(daemon)
[root@c0ec71b5ed9e /]# id 3
uid=3(adm) gid=4(adm) groups=4(adm)
```

These users would map to 100000, 100001, and 100002 respectively:

uid=2 would be 100001
uid=3 -> 100002
uid=x -> 100000+(x-1)

What happens when we create a new user inside this container ?

```
[root@c0ec71b5ed9e /]# useradd foo
[root@c0ec71b5ed9e /]# id foo
uid=1000(foo) gid=1000(foo) groups=1000(foo)
```

Within this container user_namespace, the “foo” user has a uid=1000. What would be that user’s id outside the container ?

Use our mapping algorithm: uid=x -> 100000+(x-1)

The foo user with uid=1000 inside the container would thus have a uid of 100000+(1000-1) or 100999.

We can inspect the ownership of the files in the home directory for the upperdir (ephemeral storage) to prove it:

```
[student@workstation ~]$ podman inspect 99 | less
(look for UpperDir)
[student@workstation ~]$ cd
/home/student/.local/share/containers/storage/overlay/7088d79675cdfe1be4cb8be64428f3
0a510108688758b78e33ea9990e8c8edd5/diff
[student@workstation diff]$ ls
etc  home  root  run  var
[student@workstation diff]$ cd home/
[student@workstation home]$ ls -l
total 0
drwx------. 2 100999 100999 62 Sep 21 16:37 foo
```

Understanding rootless networking

Root = virtual ethernet device
Rootless = Slirp, tap device

Container networking normally uses CNI plugins to configure a bridge, but that would require root. For rootless, podman will execute /usr/bin/slirp4netns to setup networking. This command will create a tap device that is injected inside the new networking namespace.

Also, ping might not work depending on the RHEL version:

```
[student@workstation ~]$ podman run -it ubi8 /bin/bash
[root@840855c79201 /]# yum install iputils
[root@ff226094dfd3 /]# ping google.com
PING google.com (172.217.1.238) 56(84) bytes of data.
^C
--- google.com ping statistics ---
57 packets transmitted, 0 received, 100% packet loss, time 57365ms
```

Fixed per https://bugzilla.redhat.com/show_bug.cgi?id=2037807

```
[student@workstation ~]$ rpm -q systemd
systemd-239-58.el8.x86_64

[student@workstation ~]$ rpm -q --changelog systemd
* Mon Feb 07 2022 systemd maintenance team <systemd-maint@redhat.com> - 239-57
- hash-funcs: introduce macro to create typesafe hash_ops (#2037807)
- hash-func: add destructors for key and value (#2037807)
- util: define free_func_t (#2037807)
- hash-funcs: make basic hash_ops typesafe (#2037807)
- test: add tests for destructors of hashmap or set (#2037807)
- man: document the new sysctl.d/ - prefix (#2037807)
- sysctl: if options are prefixed with "-" ignore write errors (#2037807)
- sysctl: fix segfault (#2037807)
```

https://github.com/containers/podman/blob/main/troubleshooting.md#5-rootless-containers-cannot-ping-hosts

```
[student@workstation ~]$ sysctl -a | grep ping
net.ipv4.ping_group_range = 0    2147483647
```

```
[student@workstation ~]$ podman run -it ubi8 /bin/bash
[root@34cb445d6819 /]# yum install iputils -y

[root@34cb445d6819 /]# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=4.21 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=1.70 ms

[root@34cb445d6819 /]# exit
```

Understanding rootless storage

Root = overlay2
Rootless = FUSE-overlayFS is legacy, now (RHEL8.5+) native overlay just inside a mount namespace

https://www.redhat.com/sysadmin/podman-rootless-overlay

“”
The fuse-overlay has been great. However, it is a user-space file system, which means it needs to do almost twice as much work as the kernel. Every read/write has to be interpreted by the fuse-overlay before being passed onto the host kernel. For heavy workloads that hammer the file system, the performance of fuse-overlay suffers. You could see the fuse-overlayfs pegging out the CPU. Bottom line, we should see better performance with native overlayfs, especially for heavy read/write containers in rootless mode. For example, podman build . performance should improve significantly. Note that when writing to volumes, the fuse-overlayfs is seldom used, so performance will not be affected
“”

https://www.redhat.com/en/blog/whats-new-red-hat-enterprise-linux-85-container-tools See “Better Performance with Native OverlayFS“

To see the mount you’ll have to look inside the mount namespace for a running container:

```
[student@workstation ~]$ lsns -t mnt
        NS TYPE NPROCS   PID USER      COMMAND
4026531840 mnt      69  2315 student /usr/lib/systemd/systemd --user
4026532257 mnt       2  2464 student podman
4026532464 mnt       1 24819 student sleep 5000
4026532599 mnt       1 24805 student /usr/bin/slirp4netns --disable-host-loopback --mtu=65520
--enable-sandbox --enable-seccomp -c -e 3 -r 4 --net
[student@workstation ~]$ cat /proc/24819/mounts | grep overlay
overlay / overlay
rw,context="system_u:object_r:container_file_t:s0:c111,c779",relatime,lowerdir=/home/student/.l
ocal/share/containers/storage/overlay/l/QP4SN4QNWT5H3ILUUG6SKBUFNI:/home/student/.lo
cal/share/containers/storage/overlay/l/GS7XTMTUHQDG36F4YKAI6PQYDZ,upperdir=/home/st
udent/.local/share/containers/storage/overlay/d460fa813b863b8d70195872e9abe50811cc87dfb
10663df41bbe362566892d4/diff,workdir=/home/student/.local/share/containers/storage/overlay/
d460fa813b863b8d70195872e9abe50811cc87dfb10663df41bbe362566892d4/work 0 0
```

CHAPTER 3: MANAGING CONTAINERS

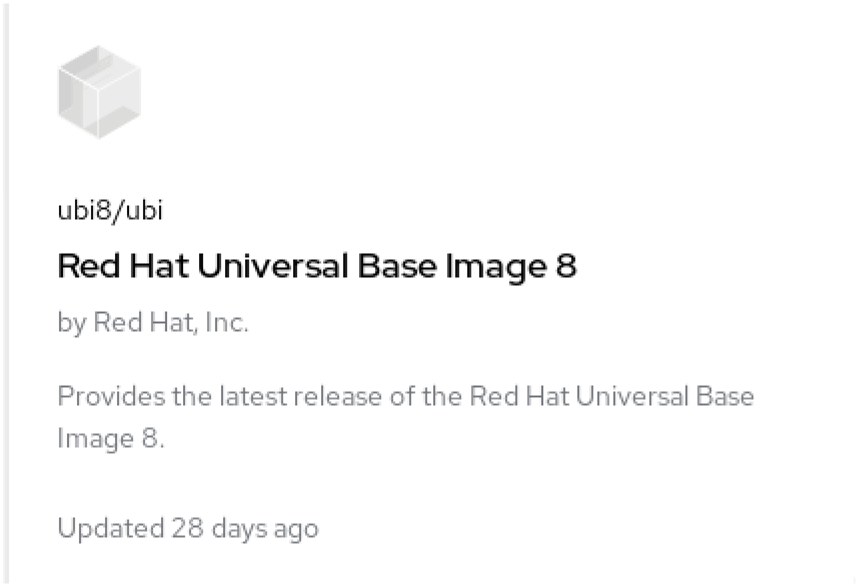
CONTAINER LIFE CYCLE MANAGEMENT WITH PODMAN

Objective: manage the life cycle of a container from creation to deletion

- Highlight Figure 3.1: Podman managing subcommands
- Getting syntax help on any of those commands - man pages !
- Podman has subcommands to: create a new container (run), delete a container (rm), list containers (ps), stop a container (stop), and start a process in a container (exec).

There is a rhel7 container image but not a rhel8 image. Where is the rhel8 container image?

Check out ubi on https://catalog.redhat.com/software/containers/explore/



ubi8/ubi

Red Hat Universal Base Image 8

by Red Hat, Inc.

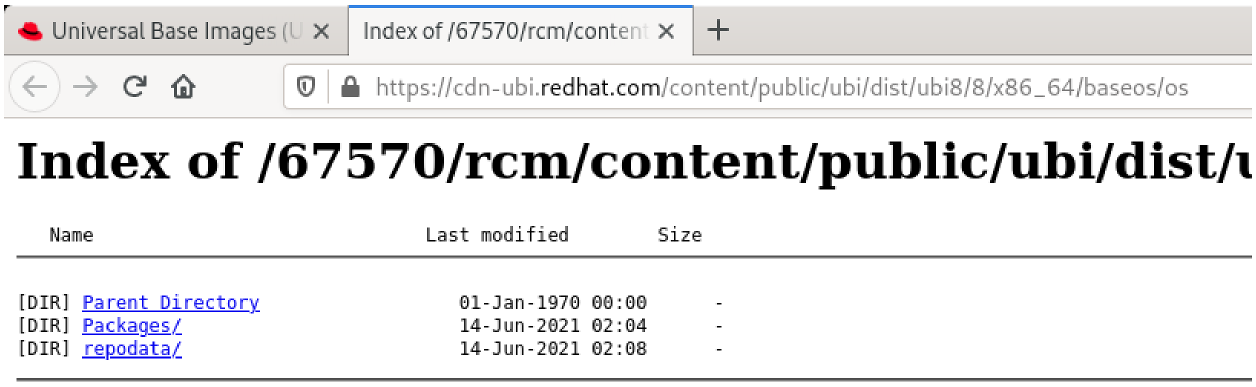
Provides the latest release of the Red Hat Universal Base Image 8.

Updated 28 days ago

```
[student@workstation ~]$ podman pull registry.access.redhat.com/ubi8
Trying to pull registry.access.redhat.com/ubi8...
```

```
[student@workstation ~]$ podman run 2722 cat /etc/redhat-release
Red Hat Enterprise Linux release 8.4 (Ootpa)

[student@workstation ~]$ podman run 2722 cat /etc/yum.repos.d/ubi.repo
[ubi-8-baseos]
name = Red Hat Universal Base Image 8 (RPMs) - BaseOS
baseurl = https://cdn-ubi.redhat.com/content/public/ubi/dist/ubi8/8/$basearch/baseos/os
enabled = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
gpgcheck = 1
```

https://access.redhat.com/articles/4238681
https://cdn-ubi.redhat.com/content/public/ubi/dist/ubi8/8/x86_64/baseos/os


Universal Base Images (U
Index of /67570/rcm/content
https://cdn-ubi.redhat.com/content/public/ubi/dist/ubi8/8/x86_64/baseos/os

Index of /67570/rcm/content/public/ubi/dist/u

```
     Name                          Last modified       Size

[DIR] Parent Directory              01-Jan-1970 00:00     -
[DIR] Packages/                     14-Jun-2021 02:04     -
[DIR] repodata/                     14-Jun-2021 02:08     -
```

Universal base images are great containers to use as building blocks for any application. They make great "parent" images.

How are the names autogenerated by podman determined ?

Names will be autogenerated for containers with the form ***adjective_famousperson***

SEE the sourcecode:
https://github.com/containers/podman/blob/main/vendor/github.com/docker/docker/pkg/namesgenerator/names-generator.go

```
[ablum@badger ~]$ cd /home/ablum/go/src/namesgenerator
```

```
[ablum@badger namesgenerator]$ go mod init
[ablum@badger namesgenerator]$ go build
[ablum@badger namesgenerator]$ ./namesgenerator
suspicious_colden
[ablum@badger namesgenerator]$ ./namesgenerator
priceless_davinci
[ablum@badger namesgenerator]$ ./namesgenerator
nice_varahamihira
```

```
	left = [...]string{
		"admiring",
		"adoring",
		"affectionate",
		"agitated",
		"amazing",
		"angry",

	right = [...]string{
		// Muhammad ibn Jābir al-Ḥarrānī al-Battānī was a founding father of astronomy.
https://en.wikipedia.org/wiki/Mu%E1%B8%A5ammad_ibn_J%C4%81bir_al-%E1%B8%A4arr%C4%81n%C4%AB_al-Batt%C4%81n%C4%AB
		"albattani",

		// Frances E. Allen, became the first female IBM Fellow in 1989. In 2006, she
became the first female recipient of the ACM's Turing Award.
https://en.wikipedia.org/wiki/Frances_E._Allen
		"allen",


func main() {
begin:
	rand.Seed(time.Now().UnixNano())
	name := fmt.Sprintf("%s_%s", left[rand.Intn(len(left))], right[rand.Intn(len(right))])
	if name == "boring_wozniak" /* Steve Wozniak is not boring */ {
		goto begin
	}
	fmt.Println(name)
}
```

What is actually running inside the httpd image from rhscl ?

```
[student@workstation ~]$ podman ps --no-trunc
CONTAINER ID                                                      IMAGE
COMMAND              CREATED       STATUS         PORTS  NAMES
a4b6429a3108095e1fdf1c509e105132f96a2a1da2eb2ce38614283f1151fb59
registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-httpd  12 minutes ago  Up
12 minutes ago          myhttpd
1d4db13a4995d493eb4176a3e88b05255e15cf9899dc336927b77882944734dc
registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-httpd  13 minutes ago  Up
13 minutes ago          objective_khorana
```

We could run another program inside the same namespaces of our myhttpd container using `podman exec`:

```
[student@workstation ~]$ podman exec -it myhttpd /bin/bash
bash-4.2$
bash-4.2$ ps -ef
UID          PID     PPID  C STIME TTY          TIME CMD
default      1       0  0 14:55 ?    00:00:00 httpd -D FOREGROUND
default      40      1  0 14:55 ?    00:00:00 /usr/bin/cat
default      41      1  0 14:55 ?    00:00:00 /usr/bin/cat
default      42      1  0 14:55 ?    00:00:00 /usr/bin/cat
default      43      1  0 14:55 ?    00:00:00 /usr/bin/cat
default      44      1  0 14:55 ?    00:00:00 httpd -D FOREGROUND
default      45      1  0 14:55 ?    00:00:00 httpd -D FOREGROUND
default      54      1  0 14:55 ?    00:00:00 httpd -D FOREGROUND
default      66      1  0 14:55 ?    00:00:00 httpd -D FOREGROUND
default      69      1  0 14:55 ?    00:00:00 httpd -D FOREGROUND
default      90      0  0 15:09 pts/0        00:00:00 /bin/bash
default      99      90  0 15:10 pts/0       00:00:00 ps -ef
```

What about the run-httpd ?

```
bash-4.2$ cat /usr/bin/run-httpd
...SNIP…
process_extending_files ${HTTPD_APP_ROOT}/src/httpd-pre-init/
${HTTPD_CONTAINER_SCRIPTS_PATH}/pre-init/

exec httpd -D FOREGROUND $@
```

So, this script (a wrapper) executed the httpd -D FOREGROUND we see running within this namespace.

What about creating a systemd.unit file so that this container is started on system boot ?

```
[student@workstation ~]$ **podman generate systemd -n myhttpd**
# container-myhttpd.service
# autogenerated by Podman 1.6.4
# Mon Sep 13 15:42:02 EDT 2021

[Unit]
Description=Podman container-myhttpd.service
Documentation=man:podman-generate-systemd(1)

[Service]
Restart=on-failure
ExecStart=/usr/bin/podman start myhttpd
ExecStop=/usr/bin/podman stop -t 10 myhttpd
KillMode=none
Type=forking
PIDFile=/run/user/1000/overlay-containers/a4b6429a3108095e1fdf1c509e105132f96a2a1da2e
b2ce38614283f1151fb59/userdata/conmon.pid

[Install]
WantedBy=multi-user.target

[student@workstation ~]$ podman stop myhttpd
[student@workstation ~]$ mkdir -p ~/.config/systemd/user
[student@workstation ~]$  podman generate systemd -n myhttpd >
~/.config/systemd/user/myhttpd.service
[student@workstation ~]$ systemctl --user daemon-reload
[student@workstation ~]$ systemctl --user enable myhttpd.service
[student@workstation ~]$ systemctl --user start myhttpd
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                                            COMMAND               CREATED
STATUS          PORTS  NAMES
a4b6429a3108  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 8 seconds ago           myhttpd
```

```
1d4db13a4995  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 5 hours ago              objective_khorana
```

```
[student@workstation ~]$ systemctl --user status myhttpd
● myhttpd.service - Podman container-myhttpd.service
  Loaded: loaded (/home/student/.config/systemd/user/myhttpd.service; enabled; vendor preset:
enabled)
  Active: active (running) since Mon 2021-09-13 16:03:49 EDT; 37s ago
```

```
[student@workstation ~]$ systemctl --user stop myhttpd
```

SEE also
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index#proc_enabling-systemd-services_assembly_porting-containers-to-systemd-using-podman

Now, it's time for us to remove the container...we are completely done with it. Consider, we have some running, but then some exited containers:

```
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                                                   COMMAND               CREATED
STATUS          PORTS  NAMES
a4b6429a3108  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 3 minutes ago            myhttpd
1d4db13a4995  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 5 hours ago              objective_khorana
```

```
[student@workstation ~]$ podman ps -a
CONTAINER ID  IMAGE                                                   COMMAND               CREATED
STATUS                  PORTS  NAMES
a4b6429a3108  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 3 minutes ago            myhttpd
1d4db13a4995  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 5 hours ago              objective_khorana
0a395937282a  registry.access.redhat.com/ubi8:latest                  cat /etc/yum.repo...  5 hours
ago  Exited (0) 5 hours ago            awesome_maxwell
e4d157d70fad  registry.access.redhat.com/ubi8:latest                  cat /etc/redhat-r...  5 hours
ago  Exited (0) 5 hours ago            quirky_sinoussi
```

To remove, we should use `podman rm`:

```
[student@workstation ~]$ podman rm myhttpd
```

```
Error: cannot remove container
a4b6429a3108095e1fdf1c509e105132f96a2a1da2eb2ce38614283f1151fb59 as it is running -
running or paused containers cannot be removed without force: container state improper

[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                                             COMMAND               CREATED
STATUS          PORTS  NAMES
a4b6429a3108  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 6 seconds ago            myhttpd
1d4db13a4995  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 5 hours ago              objective_khorana
[student@workstation ~]$ podman stop myhttpd
A4b6429a3108095e1fdf1c509e105132f96a2a1da2eb2ce38614283f1151fb59
```

Ok, good, now we check to make sure its stopped:

```
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                                             COMMAND               CREATED
STATUS                  PORTS  NAMES
a4b6429a3108  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up Less than a second ago         myhttpd
1d4db13a4995  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 5 hours ago              objective_khorana
```

Grr, systemd you are too good...its still running:

```
[student@workstation ~]$ systemctl --user stop myhttpd
[student@workstation ~]$ systemctl --user disable myhttpd
Removed /home/student/.config/systemd/user/multi-user.target.wants/myhttpd.service.
[student@workstation ~]$ podman rm myhttpd
A4b6429a3108095e1fdf1c509e105132f96a2a1da2eb2ce38614283f1151fb59

[student@workstation ~]$ podman ps -a
CONTAINER ID  IMAGE                                             COMMAND               CREATED
STATUS                  PORTS  NAMES
1d4db13a4995  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  5
hours ago  Up 5 hours ago              objective_khorana
0a395937282a  registry.access.redhat.com/ubi8:latest                cat /etc/yum.repo...  5 hours
ago  Exited (0) 5 hours ago            awesome_maxwell
e4d157d70fad  registry.access.redhat.com/ubi8:latest                cat /etc/redhat-r...  5 hours
ago  Exited (0) 5 hours ago            quirky_sinoussi
[student@workstation ~]$
```

Let's remove all these:

```
[student@workstation ~]$
[student@workstation ~]$ podman rm -a
0a395937282a3a57e0f5fddcd19563f68d29ada3e0bd895f95a520164cd3edfb
e4d157d70fad0e2252b7fa516ca0442847123c3a6e5109dbf6cdbff9f30dd74a
Error: cannot remove container
1d4db13a4995d493eb4176a3e88b05255e15cf9899dc336927b77882944734dc as it is running -
running or paused containers cannot be removed without force: container state improper
```

```
[student@workstation ~]$ podman rm -a -f
1d4db13a4995d493eb4176a3e88b05255e15cf9899dc336927b77882944734dc
```

```
[student@workstation ~]$ podman ps -a
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
```

How to extract metadata from `podman inspect` ?

```
[student@workstation ~]$ podman inspect distracted_grothendieck
```

```
[student@workstation ~]# man podman-inspect
```

This displays the low-level information on containers and images identified by name or ID. By default, this will render all results in a JSON array.

First, Let's understand JSON a bit:

(From: https://developers.squarespace.com/what-is-json)
JSON, or JavaScript Object Notation, is a minimal, readable format for structuring data.

Primarily this is done using “key” and “value” pairs...or

“Key” : “value”

Values can be any of the following:

- **String:** Several plain text characters which usually form a word
- **Boolean:** True or false.

- **Number:** An integer.
- **Object:** An associative array of key/value pairs….a “dictionary”   {   }
- **Array:** An associative array of value...a “list”   [   ]

Let’s go back to the podman-inspect man page:

    --format, -f="FORMAT"

    Format the output using the given Go template. The keys of the returned JSON can be used as the values for the --format flag (see
    examples below).

- Uses Go syntax: “A template variable can be a boolean, string, character, integer, floating-point, imaginary, or complex constant in Go syntax. Data passed to the template can be accessed using dot {{ . }}.
- “Actions” — data evaluations or control structures — is delimited by{{and}}. The data evaluated inside it is called a *Pipeline*. Anything outside them is sent to the output unchanged.

- If the data is a complex type then it’s fields can be accessed using the dot {{ .FieldName }}. Dots can be chained together if the data contains multiple complex structures. {{ .Struct.StructTwo.Field }}
  - https://curtisvermeeren.github.io/2017/09/14/Golang-Templates-Cheatsheet
  - https://www.openshift.com/blog/customizing-oc-output-with-go-templates

```
[student@workstation ~]# podman inspect 166196236b59 --format '{{.Created}}'
2019-07-29 19:48:31.858078856 +0000 UTC
```

```
[student@workstation ~]# podman inspect 5b6 --format '{{.State.Pid}}'
3404
```

What about processing a complex object that contains a list of other objects. Consider the Ulimits list here:

```
[student@workstation ~]# podman inspect 5b6

 "HostConfig": {
...SNIP...
        "Ulimits": [
                {
                "Name": "RLIMIT_NOFILE",
                "Soft": 1048576,
                "Hard": 1048576
                },
                {
                "Name": "RLIMIT_NPROC",
                "Soft": 1048576,
                "Hard": 1048576
                }
        ],
```

```
[student@workstation ~]# podman inspect 5b6 --format '{{.HostConfig.Ulimits}}'
[{RLIMIT_NOFILE 1048576 1048576} {RLIMIT_NPROC 1048576 1048576}]
```

Lets loop through the Ulimits list printing only the names for each item:

```
[student@workstation ~]# podman inspect 5b6 --format '{{range
.HostConfig.Ulimits}}{{.Name}}{{end}}'
RLIMIT_NOFILERLIMIT_NPROC
```

```
[student@workstation ~]# podman inspect 5b6 --format '{{range
.HostConfig.Ulimits}}name: {{.Name}} hard: {{.Hard}}{{end}}'
name:   RLIMIT_NOFILE   hard:   1048576name:   RLIMIT_NPROC   hard:   1048576
```

```
[student@workstation ~]# podman inspect 5b6 --format '{{range
.HostConfig.Ulimits}}name: {{.Name}} hard: {{.Hard}}{{"\n"}}{{end}}'
name:   RLIMIT_NOFILE   hard:   1048576
name:   RLIMIT_NPROC       hard:   1048576
```

```
[student@workstation ~]# podman inspect a21 --format '{{index .HostConfig.Ulimits 0}}'
{RLIMIT_NOFILE 1048576 1048576}
[student@workstation ~]# podman inspect a21 --format '{{index .HostConfig.Ulimits 1}}'
{RLIMIT_NPROC 1048576 1048576}
```

If ulimits are not available, try the BoundingCaps:

```
[student@workstation ~]$ podman inspect myhttpd --format '{{range
.BoundingCaps}}capablility:{{.}}{{"\n"}}{{end}}'
capablility:CAP_CHOWN
capablility:CAP_DAC_OVERRIDE
capablility:CAP_FOWNER
capablility:CAP_FSETID
capablility:CAP_KILL
```

```
capablility:CAP_NET_BIND_SERVICE
capablility:CAP_NET_RAW
capablility:CAP_SETFCAP
capablility:CAP_SETGID
capablility:CAP_SETPCAP
capablility:CAP_SETUID
capablility:CAP_SYS_CHROOT
```

```
[student@workstation sbin]$ podman inspect 07d1eca25e39 --format '{{range .BoundingCaps}}{{if eq . "CAP_KILL"}}eek this can kill{{end}}{{end}}'
eek this can kill
```

```
[student@workstation sbin]$ podman inspect 07d1eca25e39 --format '{{range .BoundingCaps}}{{if eq . "CAP_FOO"}}eek this can kill{{end}}{{end}}'
```

```
[student@workstation ~]# podman inspect 5b6 --format '{{.State.Pid}}'
3404
```

```
[student@workstation ~]# ps -fp 3404
UID        PID   PPID  C STIME TTY          TIME CMD
root     3404   3392  0 06:56 ?        00:00:00 sleep 5000
```

Ok - but where does this fit in with openshift ? Consider
https://www.openshift.com/blog/customizing-oc-output-with-go-templates

```
[student@workstation ~]# podman exec -it 166196236b59 /bin/bash
bash-4.2$ exit
exit
```

You can also use this with podman ps to help create tables that are useful to inspect information.

For example,

```
[student@workstation ~]$ podman ps -a --format json
[student@workstation ~]$ podman ps -a --format='{{.Names}} {{.State}} {{.Image}}'
```

What is the use case for `podman pause` ?

`podman pause` uses the cgroup "freezer" to freeze (halt) a task without stopping it or without the task knowing.

NOTE: THis is NOT supported with rootless due to a limit in the freezer cgroup.

The container and its processes are paused while the image is committed. This minimizes the likelihood of data corruption when creating the new image.
(man podman-commit)

https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt

The cgroup freezer will also be useful for checkpointing running groups
of tasks. The cgroup freezer is hierarchical. Freezing a cgroup freezes all
tasks belonging to the cgroup and all its descendant cgroups

```
[root@badger ~]# podman run -d docker.io/library/httpd

[root@badger ~]# podman ps
CONTAINER ID  IMAGE                          COMMAND           CREATED        STATUS
PORTS  NAMES
b0f5ce994715  docker.io/library/httpd:latest  httpd-foreground  4 seconds ago  Up 3 seconds
ago       elated_montalcini

[root@badger ~]# ps -ef | grep httpd
root      27369 27357  2 15:31 ?        00:00:00 httpd -DFOREGROUND
bin       27389 27369  0 15:31 ?        00:00:00 httpd -DFOREGROUND
bin       27390 27369  0 15:31 ?        00:00:00 httpd -DFOREGROUND
bin       27391 27369  0 15:31 ?        00:00:00 httpd -DFOREGROUND
root      27509 19190  0 15:31 pts/1    00:00:00 grep --color=auto httpd

[root@badger ~]# cat /sys/fs/cgroup/freezer/machine.slice/libpod-b0f5ce*/freezer.state
THAWED

[root@badger ~]# podman pause b0f5ce994715

[root@badger ~]# cat /sys/fs/cgroup/freezer/machine.slice/libpod-b0f5ce*/freezer.state
```

```
FROZEN

[root@badger ~]# curl 10.88.0.30
(hangs)

[root@badger ~]# podman commit b0f5ce994715 docker.io/library/httpd:mypause
[root@badger ~]# podman images
REPOSITORY                    TAG      IMAGE ID       CREATED         SIZE
docker.io/library/httpd       mypause  1ab4200ebb2a   9 seconds ago   170 MB

[root@badger ~]# podman unpause b0f5ce994
b0f5ce9947155595b0d2d2c2c31dded774e8107c77e9658df0b2e23fc5c7306c

[root@badger ~]# curl 10.88.0.30
<html><body><h1>It works!</h1></body></html>

[root@badger ~]# cat /sys/fs/cgroup/freezer/machine.slice/libpod-b0f5ce*/freezer.state
THAWED
```

ATTACHING PERSISTENT STORAGE TO CONTAINERS

How does overlay work?

```
[root@workstation overlay-images]# cat /etc/containers/storage.conf  | grep -v ^#
[storage]
driver = "overlay"
runroot = "/var/run/containers/storage"
graphroot = "/var/lib/containers/storage"
```

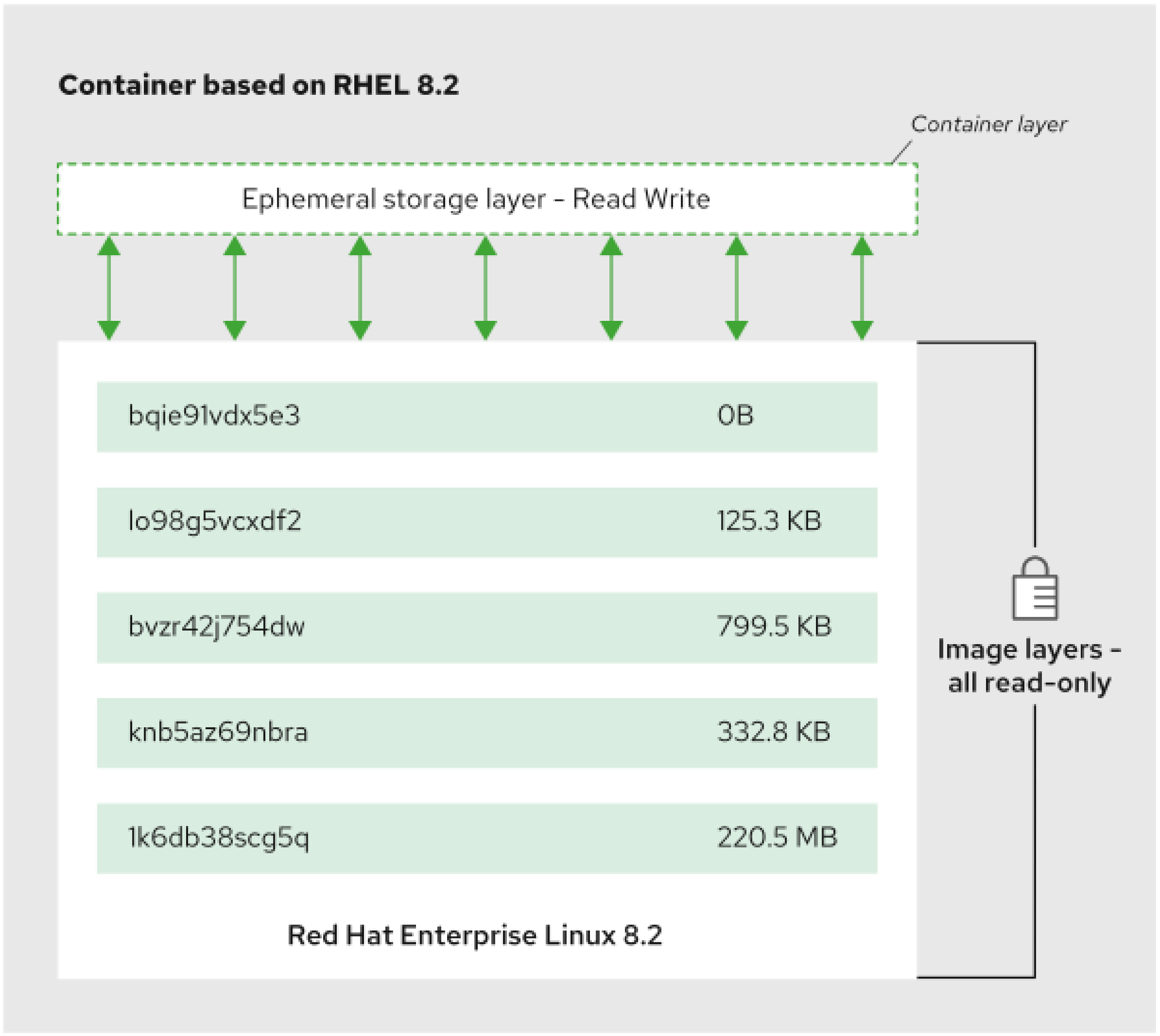

Container based on RHEL 8.2
Container layer
Ephemeral storage layer - Read Write
bqie91vdx5e3 0B
lo98g5vcxdf2 125.3 KB
bvzr42j754dw 799.5 KB
knb5az69nbra 332.8 KB
1k6db38scg5q 220.5 MB
Red Hat Enterprise Linux 8.2
Image layers - all read-only

Figure 3.3: Container layers

https://docs.docker.com/storage/storagedriver/overlayfs-driver/#how-the-overlay2-driver-works

The overlay2 driver natively supports up to 128 lower OverlayFS layers - the original overlay driver only worked with 2 layers, extra layers in overlay relied on hard-linked directories. This created excessive use of inodes (known limitation)

NOTE1: rootless uses /home/student/.local/share/containers/ for it's ephemeral storage

~~**NOTE2: rootless uses fuse-overlayfs: https://github.com/containers/fuse-overlayfs**~~

```
[root@workstation storage]# mkdir -p /data/lower{1..3} /data/upper /data/work /data/merged

[root@workstation storage]# touch /data/lower1/file{1..3}  /data/lower2/file{4..6}
/data/lower3/file{7..9}  /data/upper/file10

[root@workstation storage]# tree /data

[root@workstation storage]# mount -t overlay overlay -o
lowerdir=/data/lower1:/data/lower2:/data/lower3,upperdir=/data/upper,workdir=/data/work
/data/merged

[root@workstation storage]# tree /data

[root@workstation storage]# ls -l /data/merged
total 0
-rw-r--r--. 1 root root 0 Jul 29 20:29 file1
-rw-r--r--. 1 root root 0 Jul 29 20:29 file2
-rw-r--r--. 1 root root 0 Jul 29 20:29 file3
-rw-r--r--. 1 root root 0 Jul 29 20:30 file4
-rw-r--r--. 1 root root 0 Jul 29 20:30 file5
-rw-r--r--. 1 root root 0 Jul 29 20:30 file6
-rw-r--r--. 1 root root 0 Jul 29 20:30 file7

 [root@workstation storage]#   echo hello > /data/merged/file10
```

Which directory(ies) should show a change ?

```
[root@workstation storage]#  ls -lR /data/
```

Notice its 6 bytes in both the /data/merged AND /data/upper layer. Ok now let’s modify a different file, one in the “lower” layers.

```
[root@workstation storage]# echo hello > /data/merged/file9
```

Which directory(ies) should show a change ?

```
[root@workstation storage]#  ls -lR /data/
```

Notice “lower3” is where the original file9 was created in, but it didn’t change:

```
/data/lower3:
```

```
total 0
-rw-r--r--. 1 root root 0 Aug 13 11:37 file7
-rw-r--r--. 1 root root 0 Aug 13 11:37 file8
-rw-r--r--. 1 root root 0 Aug 13 11:37 file9
```

The change shows up in the “upper” layer (and merged):

```
/data/upper:
total 8
-rw-r--r--. 1 root root 6 Aug 13 11:39 file10
-rw-r--r--. 1 root root 6 Aug 13 11:40 file9
```

Containers use overlay to share underlying images in “lower” layers. The upper is used to track changes only.

- Notice the layers for this container:

```
[root@workstation storage]# podman run -d rhscl/httpd-24-rhel7
[root@workstation storage]# podman inspect 37 | less

        "GraphDriver": {
        "Name": "overlay",
        "Data": {
                "LowerDir":
"/var/lib/containers/storage/overlay/77b158205d2cc791c9dc6f1ca04db6dca5e0737fc8a44f6c60
df42626d4838be/diff:/var/lib/containers/storage/overlay/4d3aa8111d6a2805f45d11997708bb5a1
79add74d61da9543b10818caa72e1fd/diff:/var/lib/containers/storage/overlay/5201771aee3980a
1208cf5111c23763492e29cdacbae70d85f0dfbbeb3fa069c/diff:/var/lib/containers/storage/overlay
/da289ed398e809e9e58320f71e3f32a0cfc881fc7db55e1a3a7bb1125e5b8c1e/diff",
                "MergedDir":
"/var/lib/containers/storage/overlay/49701e617da01a61fa60a9ced94efaaa1697828c621a37f98d
6ab3650de3bc1d/merged",
                "UpperDir":
"/var/lib/containers/storage/overlay/49701e617da01a61fa60a9ced94efaaa1697828c621a37f98d
6ab3650de3bc1d/diff",
                "WorkDir":
"/var/lib/containers/storage/overlay/49701e617da01a61fa60a9ced94efaaa1697828c621a37f98d
6ab3650de3bc1d/work"
        }
```

- To reclaim storage on the host system run **podman rm**
    - Consider both **podman ps** and **podman ps -a**

Has overlay2 always been the storage driver used on container hosts ?

No. See https://bugzilla.redhat.com/show_bug.cgi?id=1475625

- Do we support devicemapper ?

[1/30/2020]

We currently support both devicemapper and overlayfs (overlay2) with docker. Before RHEL 7.5, the default docker storage configuration is devicemapper in direct-lvm[1] mode. Starting with 7.5, the default docker storage configuration is overlay2[2]. I do not know how the various versions/releases of OCP 3.x change/overwrite the default docker
storage configuration.

Podman, buildah, and CRI-O only support overlayfs[3][4], and this is configured by default.

CRI-O and docker can be installed and running on the same RHEL 7 host,
but they use completely different, incompatible storage configurations.

So, the original question is do we currently support the docker devicemapper storage driver, correct? If so, yes, we still support devicemapper in direct-lvm mode. Hope that helps!

-Derrick

[1]
https://docs.docker.com/storage/storagedriver/device-mapper-driver/#configure-direct-lvm-mode-for-production
[2] https://docs.docker.com/storage/storagedriver/overlayfs-driver/
[3]
https://www.redhat.com/en/blog/working-container-storage-library-and-tools-red-hat-enterprise-linux
[4] https://bugzilla.redhat.com/show_bug.cgi?id=1774789#c3

What if we want persistent, performant storage ?

- Add “volumes” which our directories on the host system exposed to the container.
- Permissions and selinux matter !
  - Rootless containers will need to set permissions using `podman unshare`

```
[student@workstation ~]$ mkdir mydata

[student@workstation ~]$ podman run -d -v /home/student/mydata:/var/www/html
registry.redhat.io/rhscl/httpd-24-rhel7
Trying to pull registry.redhat.io/rhscl/httpd-24-rhel7...
8971a0ce72fb6fb35615f6d4dcbb28b77747c16dec095befb08a2d4d558e8a3e
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                                    COMMAND            CREATED
STATUS         PORTS  NAMES
8971a0ce72fb  registry.redhat.io/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  10 seconds ago
Up 9 seconds ago       musing_shirley
[student@workstation ~]$
[student@workstation ~]$
[student@workstation ~]$ podman inspect 89 --format='{{.Mounts}}'
[{bind  /home/student/mydata /var/www/html   [rbind] true rprivate}]
[student@workstation ~]$ podman exec -it 89 /bin/bash
bash-4.2$ df -h
Filesystem      Size  Used Avail Use% Mounted on
fuse-overlayfs  9.9G  6.6G  3.3G  67% /
tmpfs           64M    0   64M   0% /dev
tmpfs           580M  128K  580M   1% /etc/hosts
shm             63M    0   63M   0% /dev/shm
/dev/vda3       9.9G  6.6G  3.3G  67% /var/www/html

bash-4.2$ touch /var/www/html/index.html
touch: cannot touch '/var/www/html/index.html': Permission denied
bash-4.2$ id
uid=1001(default) gid=0(root) groups=0(root)
bash-4.2$ ls -ld /var/www/html
drwxrwxr-x. 2 root root 6 Sep 14 14:20 /var/www/html
bash-4.2$ chown 1001 /var/www/html
chown: changing ownership of '/var/www/html': Permission denied
bash-4.2$
bash-4.2$ exit
```

```
exit
Error: non zero exit code: 1: OCI runtime error
```

Ok, let's try to fix these permissions outside the container

```
[student@workstation ~]$ ls -ldn mydata
drwxrwxr-x. 2 1000 1000 6 Sep 14 10:20 mydata
[student@workstation ~]$ chown 1001 mydata
chown: changing ownership of 'mydata': Operation not permitted
```

Failed. You'll need to use `podman unshare` to run the chown as if student's UID were 0 (ie root). SEE man podman-unshare

```
[student@workstation ~]$ podman unshare chown -R 1001 mydata
[student@workstation ~]$ ls -ldn mydata
drwxrwxr-x. 2 101000 1000 6 Sep 14 10:20 mydata
[student@workstation ~]$ cat /etc/subuid
student:100000:65536
devops:165536:65536
[student@workstation ~]$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
101000-100000
1000
```

NOTE: 1000 is 1 less than the mapped uid (1001) used within the user namspace of the containerized process. Consider making this a different uid (one typically used with mysql):

```
[student@workstation ~]$ podman unshare chown 27:27 mydata
[student@workstation ~]$ ls -ldn mydata
drwxrwxr-x. 2 100026 100026 6 Sep 14 10:20 mydata
```

Think it will be "27" inside the user namespace, but it maps to "100026" outside. This uid falls within the subordinate range (aka subuid) allocated to the student user:

```
[student@workstation ~]$ man subuid
```

Each line in /etc/subuid contains a user name and a range of subordinate user ids that user is allowed to use. This is specified with three fields
delimited by colons (":"). These fields are:

- login name or UID
- numerical subordinate user ID
- numerical subordinate user ID count

Ok, let's return the ownership back to a subordinate ID that will map to permissions of the working user within the httpd container's namespace (ie the "default" user):

```
[student@workstation ~]$ podman unshare chown 1001:0 mydata
[student@workstation ~]$ ls -ldn mydata
drwxrwxr-x. 2 101000 1000 6 Sep 14 10:20 mydata
```

Verify with:

```
[student@workstation ~]$ podman exec -it 89 /bin/bash
bash-4.2$ ls -ldn /var/www/html
drwxrwxr-x. 2 1001 0 6 Sep 14 14:20 /var/www/html
bash-4.2$ id
uid=1001(default) gid=0(root) groups=0(root)
bash-4.2$ touch /var/www/html/index.html
touch: cannot touch '/var/www/html/index.html': Permission denied
```

Grrr… still not working, any thoughts ? SELinux ?

```
[student@workstation ~]$ sudo grep mydata /var/log/audit/audit.log
type=AVC msg=audit(1631631220.011:137): avc:  denied  { write } for  pid=3436 comm="touch"
name="mydata" dev="vda3" ino=26100164
scontext=system_u:system_r:container_t:s0:c408,c911
tcontext=unconfined_u:object_r:user_home_t:s0 tclass=dir permissive=0
```

```
[student@workstation ~]$ ls -ldZ mydata
drwxrwxr-x. 2 101000 student unconfined_u:object_r:user_home_t:s0 6 Sep 14 10:20 mydata
```

```
[student@workstation ~]$ ps -eZ | grep httpd
system_u:system_r:container_t:s0:c408,c911 2896 ? 00:00:00 httpd
```

If we try to fix the permissions as the student user:

```
[student@workstation ~]$ chcon -t container_file_t mydata
chcon: failed to change context of 'mydata' to 'unconfined_u:object_r:container_file_t:s0':
Operation not permitted
```

We must do this with `podman unshare` like before:

```
[student@workstation ~]$ podman unshare chcon -t container_file_t mydata
[student@workstation ~]$ ls -ldZ mydata
drwxrwxr-x. 2 101000 student unconfined_u:object_r:container_file_t:s0 6 Sep 14 10:20 mydata

[student@workstation ~]$ podman exec -it 89 /bin/bash
bash-4.2$ echo hello > /var/www/html/index.html
bash-4.2$
bash-4.2$
bash-4.2$ exit
exit
[student@workstation ~]$ podman exec 89 curl -s "http://localhost:8080"
hello
```

works!

How to apply selinux labels ?

The selinux label on a volume can be made persistent on a host with:

```
[student@workstation ~]$ sudo semanage fcontext -a -t container_file_t
'/home/student/mydata(/.*)?'
[student@workstation ~]$ sudo restorecon -Rv /home/student/mydata/
/home/student/mydata not reset as customized by admin to
unconfined_u:object_r:container_file_t:s0
```

Alternatively use the :Z option like --

```
[student@workstation ~]$ mkdir mydata1
[student@workstation ~]$  podman unshare chown 1001 mydata1
[student@workstation ~]$  podman run -d -v /home/student/mydata1:/var/www/html:Z
registry.redhat.io/rhscl/httpd-24-rhel7
9c9d8752085e22ad32407b1f655a1a49e2427ceeae46ca6455665c6b0412db96
[student@workstation ~]$ podman exec 9c ls -ldZ /var/www/html
drwxrwxr-x. default root system_u:object_r:container_file_t:s0:c1002,c1014 /var/www/html
[student@workstation ~]$ podman exec 9c ls -ldZ /var/www/html
drwxrwxr-x. default root system_u:object_r:container_file_t:s0:c1002,c1014 /var/www/html
```

```
[student@workstation ~]$ podman exec 9c touch /var/www/html/index.html
[student@workstation ~]$ ls -lZ mydata1
total 0
-rw-r--r--. 1 101000 student system_u:object_r:container_file_t:s0:c1002,c1014 0 Sep 14 11:03
index.html
```

ACCESSING CONTAINERS

Objective: Describe the basics of networking with containers.
Objective:Remotely connect to services within a container.

***NOTE*: Rootless container networking is different and uses slirp4netns which provides user-mode networking ("slirp") for network namespaces.**

What is used to configure networking for containers run as the root user?

- the Container Networking Interface (CNI) open source project. https://github.com/containernetworking/cni
- The CNI project aims to standardize the network interface for containers in cloud native environments, such as Kubernetes and Red Hat OpenShift Container Platform
- consists of a specification and libraries for writing plugins to configure network interfaces in Linux containers

What other plugins are available ?
https://github.com/containernetworking/plugins

Some CNI network plugins, maintained by the containernetworking team. There are others maintained by different teams. Consider

There is an ovs cni plugin among others:
Ovs-cni plugin: https://github.com/kubevirt/ovs-cni/blob/master/docs/cni-plugin.md

```
[student@workstation ~]$ cat /etc/cni/net.d/87-podman-bridge.conflist
```

```
[root@workstation ~]# bridge link show
4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 master virbr0 state disabled priority 32
cost 100

[root@workstation ~]# ip addr show
(note missing interface for cni-podman0)
```

Now, let's run a container and check the networking on the host again:

```
[root@workstation ~]# podman run -d rhscl/httpd-24-rhel7
44d328fb7ca9f951e5e165a4eff4860789c446fc35c249844259b9f3320e67fa

[root@workstation ~]# bridge link show
4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 master virbr0 state disabled priority 32
cost 100
6: vethf643eab8@virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master
cni-podman0 state forwarding priority 32 cost 2


[root@workstation ~]# ip addr show cni-podman0
5: cni-podman0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
        link/ether 32:f4:04:21:cd:64 brd ff:ff:ff:ff:ff:ff
        inet 10.88.0.1/16 brd 10.88.255.255 scope global cni-podman0
        valid_lft forever preferred_lft forever
        inet6 fe80::30f4:4ff:fe21:cd64/64 scope link
        valid_lft forever preferred_lft forever
```

How does ip allocation work with cni ipam "host-local" ?

IPAM=IP Address Management

Check in **/var/lib/cni/networks/podman/**

The IP address assigned to a particular container is listed as a flat text file while its contents matches its networking namespace (or containerID).

The **last_reserved_ip.0** is a "helper" file indicating the last assigned IP address.

```
[root@workstation ~]# cd /var/lib/cni/networks/podman/
[root@workstation podman]# ls
10.88.0.31  10.88.0.33  last_reserved_ip.0  lock
[root@workstation podman]# cat 10.88.0.33
e186fa7ddb77eb96c1017e5b34193c432e10d7227b0d7b060f63bdc9379e7dcb
```

```
[root@workstation ~]# podman ps
CONTAINER ID  IMAGE                                          COMMAND               CREATED
STATUS         PORTS  NAMES
e186fa7ddb77  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  2
minutes ago  Up 2 minutes ago         pedantic_stonebraker
```

```
[root@workstation ~]# curl 10.88.0.33:8080 | grep title
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3985  100  3985    0     0   486k      0 --:--:-- --:--:-- --:--:--  555k
        <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
```

But, does this work from a different machine in our network?

```
[root@workstation ~]# ssh student@bastion

[student@bastion ~]$
[student@bastion ~]$
[student@bastion ~]$ curl 10.88.0.33:8080 | grep title
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:14 --:--:--     0
^C
[student@bastion ~]$
[student@bastion ~]$ nc -v 10.88.0.33 8080
Ncat: Version 7.70 ( https://nmap.org/ncat )
^C
[student@bastion ~]$ nc -v workstation 22
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connected to 172.25.250.9:22.
SSH-2.0-OpenSSH_8.0
```

So, the apache process running in the container on workstation is not available outside of workstation.

- Containers running on other hosts aren't connected by podman either.
- Mapping a networking port on the workstation (host) to the container's networking namespace (apache) would be required even for a container run as the root user

SEE Figure 3.4: Basic Linux container networking

MAPPING NETWORK PORTS

How can we connect to a containerized application from the host's network ?

Ports !

```
[student@workstation ~]# podman run -d -p 8888:8080 ubi8 sleep 5000
845072a99bbdca4d0581e0a75326fa72d44c75d1e4f52f63420c1dfc810d1110
[student@workstation ~]# netstat -tunap | grep 8888
tcp        0        0 0.0.0.0:8888           0.0.0.0:*                LISTEN        3439/slirp4netns


[student@workstation ~]$ ps -ef | grep slirp
student          3439    1  0 06:54 pts/1          00:00:00 /usr/bin/slirp4netns --api-socket
/run/user/1000/libpod/tmp/930f971c0e8c14bdba1605f09f7fd3ea14c9aa0d1384f3a7a348e4181f
a126f1.net --disable-host-loopback --mtu 65520 --enable-sandbox -c -e 3 -r 4 --netns-type=path
/run/user/1000/netns/cni-200e5dfd-dabc-4c02-b2a5-392a7dc4ce87 tap0

[student@workstation ~]$ podman exec -it 93  /bin/bash
[root@930f971c0e8c /]# yum install iproute -y
[root@930f971c0e8c /]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: tap0: <BROADCAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state UNKNOWN group
default qlen 1000
        link/ether 36:c3:4a:58:15:04 brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.100/24 brd 10.0.2.255 scope global tap0
        valid_lft forever preferred_lft forever
        inet6 fe80::34c3:4aff:fe58:1504/64 scope link
        valid_lft forever preferred_lft forever
```

This network exists within a separate networking namespace from the global one. Consider findmnt. Look for nsfs (namespace filesystem):

```
[student@workstation ~]$ podman unshare findmnt | grep nsfs
|        └─/run/user/1000/netns/cni-e74fcbb7-b359-3084-eab8-599b903d2647
        nsfs[net:[4026532260]]                      nsfs          rw,seclabel
[student@workstation ~]$ findmnt | grep nsfs
[student@workstation ~]$
```

Nsfs is a pseudo-filesystem. See the commit in the kernel here:

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=e149ed2b805fefdccf7ccdfc19eca22fdd4514ac

NOTE: if run as root, conmon is the listening process:

```
[root@workstation ~]# podman run -d -p 8081:8080 rhscl/httpd-24-rhel7
eef01bf84d01da35e289ec27b8dd4ea4aec188114c18daee567fef3d5f22a51d
[root@workstation ~]# netstat -tunap | grep 8081
tcp        0        0 0.0.0.0:8081          0.0.0.0:*               LISTEN        8956/conmon


 [student@workstation ~]$ curl localhost:8081 | grep title
  % Total         % Received % Xferd  Average Speed   Time Time    Time  Current
                                Dload  Upload   Total   Spent Left  Speed
100  3985  100  3985 0         0  1945k        0 --:--:-- --:--:-- --:--:-- 1945k
         <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
```

Since we are using host's network, the host's firewall will need to be modified:

```
[student@workstation ~]# sudo firewall-cmd --add-port=8081/tcp
[student@workstation ~]$ ssh bastion
Activate the web console with: systemctl enable --now cockpit.socket

This system is not registered to Red Hat Insights. See https://cloud.redhat.com/
To register this system, run: insights-client --register

Last login: Mon Jun 14 12:22:35 2021 from 172.25.250.9
[student@bastion ~]$ curl workstation:8081 | grep title
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3985  100  3985    0     0   353k      0 --:--:-- --:--:-- --:--:--  353k
          <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
```

Is Podman 4.0 is moving away from CNI ?

https://www.redhat.com/sysadmin/podman-new-network-stack

Of the new features in Podman v4.0, one of the most important is a new network stack, written from scratch in Rust to support Podman. The new stack is composed of two tools, the Netavark network setup tool and the Aardvark DNS server.

Existing containers in nondefault networks cannot be converted to Netavark, and Netavark doesn't support advanced CNI plugins (for example, connecting to Kubernetes networks created using Flannel). To ensure a smooth transition, we will continue to support CNI with Podman, and existing Podman installations will continue to use CNI for networking.

New installations can opt to use CNI by explicitly specifying it via the containers.conf configuration file, using the network_backend field. CNI and Netavark cannot be used simultaneously in order to avoid conflicts in the configurations the two create.

What is used currently in DO180 classroom ? CNI see:

```
[root@workstation ~]# cat /usr/share/containers/containers.conf
```

```
[network]

# Network backend determines what network driver will be used to set up and tear down
container networks.
# Valid values are "cni" and "netavark".
# The default value is empty which means that it will automatically choose CNI or netavark. If
there are
# already containers/images or CNI networks preset it will choose CNI.
#
# Before changing this value all containers must be stopped otherwise it is likely that
# iptables rules and network interfaces might leak on the host. A reboot will fix this.
#
#network_backend = ""
network_backend = "cni"
```

Are there python modules that can work directly with libpod ?

No. Not directly, but there is a RESTful API available with podman started by running `podman system service tcp:localhost:8080 --log-level=debug --time=0` like:

This could be adapted to python code using the requests module ie
https://www.redhat.com/sysadmin/podman-python-bash

There are some efforts to use this api in python here: https://github.com/containers/podman-py

A new API is coming with version 2.0 of podman (we are using 1.6.4 in the training env):
https://www.redhat.com/sysadmin/podmans-new-rest-api

https://docs.podman.io/en/latest/_static/api.html

CHAPTER 4: MANAGING CONTAINER IMAGES

Where can I find the actual OCI specification ?

OCI spec: https://opencontainers.org/

Runtime-spec https://github.com/opencontainers/runtime-spec
Image-spec https://github.com/opencontainers/image-spec

CONFIGURING REGISTRIES IN PODMAN

How can you block access to certain registries system-wide ?

This should now be possible because of https://bugzilla.redhat.com/show_bug.cgi?id=1787667

..but you can't really - https://bugzilla.redhat.com/show_bug.cgi?id=1811098

(older podman configuration format)

```
[student@workstation ~]$ sudo vi /etc/containers/registries.conf
[registries.block]
registries = ['docker.io']
```

(newer podman v2 configuration format)

```
[student@workstation ~]$ sudo vi /etc/containers/registries.conf
unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io", "quay.io"]

[[registry]]
prefix = "quay.io"
location = "quay.io"
insecure = false
```

```
blocked = true
```

(end new format)

```
[student@workstation ~]$ podman pull quay.io/redhattraining/httpd-parent:latest
Trying to pull quay.io/redhattraining/httpd-parent:latest...
Error: initializing source docker://quay.io/redhattraining/httpd-parent:latest: registry quay.io is
blocked in /etc/containers/registries.conf or /home/student/.config/containers/registries.conf.d
```

```
[student@workstation ~]$ podman pull quay.io/ajblum/mytest:latest
Trying to pull quay.io/ajblum/mytest:latest...
Error: initializing source docker://quay.io/ajblum/mytest:latest: registry quay.io is blocked in
/etc/containers/registries.conf or /home/student/.config/containers/registries.conf.d
```

It's also possible to block registries from a particular namespace. For this, use the location for matching instead of the prefix:

```
unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io", "quay.io"]

[[registry]]
location = "quay.io/ajblum"
insecure = false
blocked = true
```

```
[student@workstation ~]$ podman pull quay.io/ajblum/mytest:latest
Trying to pull quay.io/ajblum/mytest:latest...
Error: initializing source docker://quay.io/ajblum/mytest:latest: registry quay.io is blocked in
/etc/containers/registries.conf or /home/student/.config/containers/registries.conf.d
```

Fails as expected, let's try a different

```
[student@workstation ~]$ podman pull quay.io/redhattraining/httpd-parent:latest
Trying to pull quay.io/redhattraining/httpd-parent:latest...
Getting image source signatures
```

Works. So, only a specific namespace can be blocked.

Ok - Looks good, right ? Now, we create a local registries.conf as a non-root user (one that allows access to docker.io):

```
[student@workstation ~]$ mkdir -p ~/.config/containers/
[student@workstation ~]$  touch ~/.config/containers/registries.conf

[student@workstation ~]$ podman pull quay.io/ajblum/mytest:latest
Trying to pull quay.io/ajblum/mytest:latest...
Getting image source signatures
```

Worked … eek ! But, really nothing can stop a motivated user to work around this global config. From BZ 1811098 "a local user could still pull an image via curl or by pointing the tools to another path." https://bugzilla.redhat.com/show_bug.cgi?id=1811098

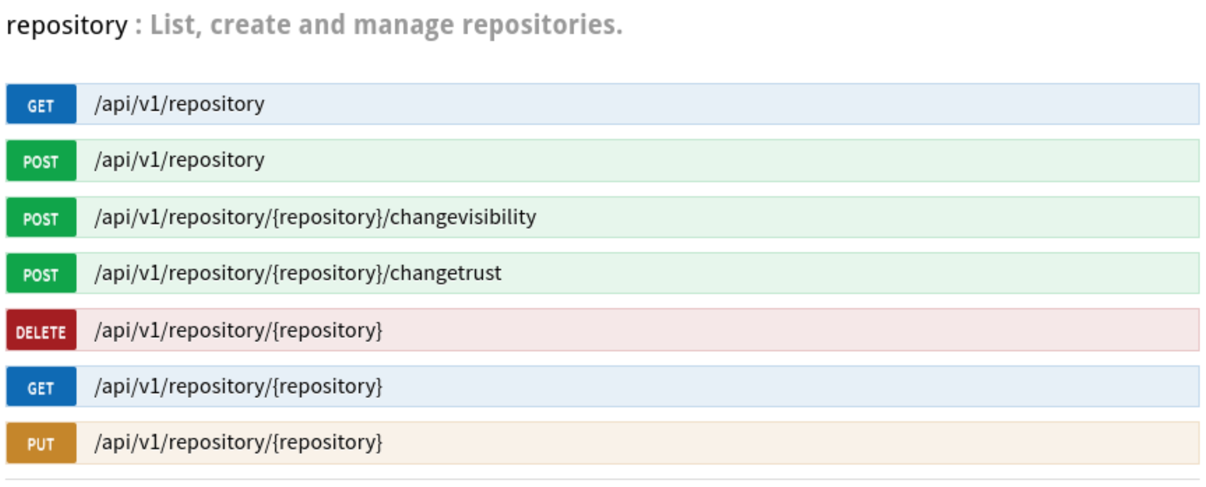
How to use the registry http api directly ?

```
[root@workstation ~]# podman search quay.io/mytest
INDEX      NAME                   DESCRIPTION                    STARS   OFFICIAL
AUTOMATED
quay.io   quay.io/ihoukai/mytest                                  0
quay.io   quay.io/little_arhat/mytest   test of homu/quay integration   0
quay.io   quay.io/guenael/mytest                                  0
quay.io   quay.io/ajblum/mytest                                   0
```

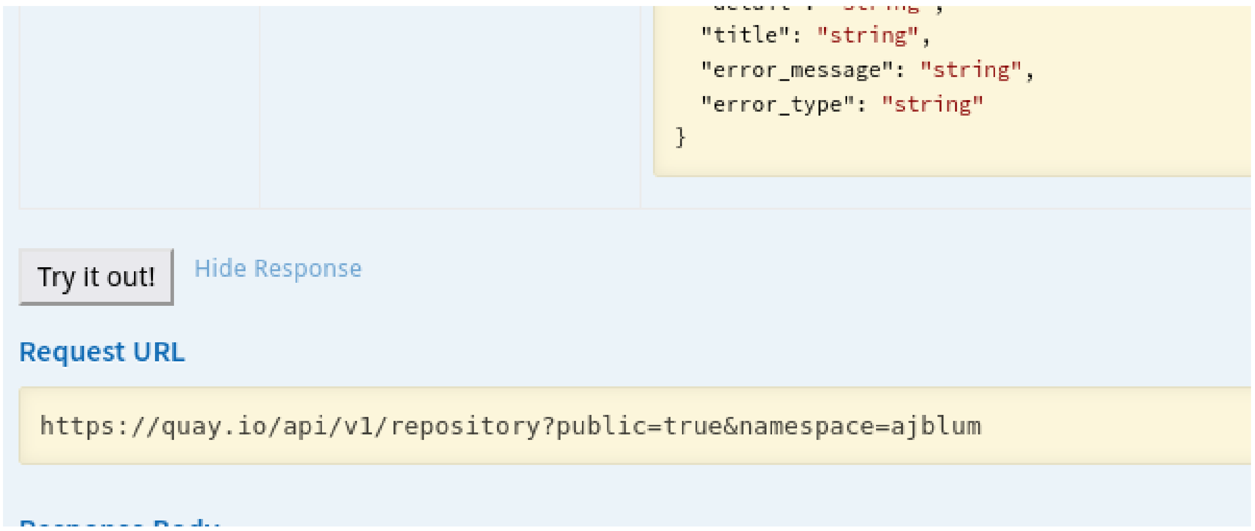
```
[root@workstation ~]# curl https://quay.io/v2/ajblum/mytest/tags/list
{"name":"ajblum/mytest","tags":["1.0","latest","2.0","3.0","4.0","5.0"]}
```

Additional curl troubleshooting: https://access.redhat.com/articles/3560571

Docker Registry API docs: https://docs.docker.com/registry/spec/api/
For Quay: https://docs.quay.io/api/swagger/


repository : List, create and manage repositories.

```
GET     /api/v1/repository
POST    /api/v1/repository
POST    /api/v1/repository/{repository}/changevisibility
POST    /api/v1/repository/{repository}/changetrust
DELETE  /api/v1/repository/{repository}
GET     /api/v1/repository/{repository}
PUT     /api/v1/repository/{repository}
```



```
  "title": "string",
  "error_message": "string",
  "error_type": "string"
}
```

Request URL

```
https://quay.io/api/v1/repository?public=true&namespace=ajblum
```

```
[student@workstation ~]$ curl -L
"https://quay.io/api/v1/repository?public=true&namespace=ajblum"
{"repositories": [{"namespace": "ajblum", "name": "mytest", "description": "", "is_public": true,
"kind": "image", "state": "NORMAL", "quota": null}, {"namespace": "ajblum", "name": "myapp",
"description": null, "is_public": true, "kind": "image", "state": "NORMAL", "quota": null},
{"namespace": "ajblum", "name": "httpd-systemd", "description": null, "is_public": true, "kind":
….SNIP….

[student@workstation ~]$ curl -Ls
"https://quay.io/api/v1/repository?public=true&namespace=ajblum" | jq
'.repositories[].name'
"mytest"
"myapp"
```

```
"httpd-systemd"
"versioned-hello"
"myubi"
"foo"
"helloworld"
"debezium-connector-postgres"
"rhel7-attr"
"hello-openshift"
"myubitest"
"mysigtest"
"do180"
```

A better tool, skopeo:

```
[root@workstation ~]# skopeo inspect docker://quay.io/ajblum/mytest
{
        "Name": "quay.io/ajblum/mytest",
        "Tag": "latest",
        "Digest":
"sha256:6cd0217844a2d778786dcc8c9c948aecc6ca1a36f8f16e5e4bbd4151f7ba5a61",
        "RepoTags": [
        "1.0",
        "latest",
        "2.0",
        "3.0",
        "4.0",
        "5.0"
```

There is also an RFE against podman for searching:

https://bugzilla.redhat.com/show_bug.cgi?id=1757531

Other ways to copy images locally other than podman pull ?

```
[student@workstation ~]$ mkdir /tmp/mytest
[student@workstation ~]$ skopeo copy docker://quay.io/ajblum/mytest:1.0 dir:/tmp/mytest
[student@workstation mytest]$ cd /tmp/mytest/
[student@workstation mytest]$  ls
```

```
[student@workstation mytest]$ cat manifest.json
[student@workstation mytest]$ cat manifest.json | json_reformat
[student@workstation mytest]$ file
a38d7adc1eb9f56b95435dfb6a51d26e225ef0181c0c71f9f8434c79e98aa59f
[student@workstation mytest]$  tar xvzf
a38d7adc1eb9f56b95435dfb6a51d26e225ef0181c0c71f9f8434c79e98aa59f
```

```
[student@workstation ~]$ skopeo copy docker://quay.io/ajblum/mytest:1.0
containers-storage:quay.io/ajblum/mytest:1.0

[student@workstation ~]$ skopeo copy docker://quay.io/ajblum/mytest:1.0
oci-archive:/tmp/mytest/mytest.tar

[student@workstation ~]$  podman load -i /tmp/mytest/mytest.tar
Getting image source signatures

[student@workstation ~]$ podman images
REPOSITORY   TAG          IMAGE ID      CREATED      SIZE
<none>          <none>   a6a3e178a6bc   4 months ago   215MB
```

What about registry.redhat.io ?

This works:

```
[student@workstation ~]# skopeo inspect docker://registry.access.redhat.com/rhel
{
        "Name": "registry.access.redhat.com/rhel",
        "Digest":
"sha256:2d215868e282e68998adece762d374ea49d66266d9dee67776eddc80a3d8e168",
        "RepoTags": [
        "7.3-74",
```

But, not this:

```
[student@workstation ~]# skopeo inspect docker://registry.redhat.io/rhel
FATA[0000] unable to retrieve auth token: invalid username/password
```

Skopeo will use the same authentication used by podman.

How can you pull an image using its digest ?

Suppose you are interested in specific images from registry.access.redhat.com/rhscl/httpd-24-rhel7

```
[student@workstation storage]$ skopeo inspect
docker://registry.access.redhat.com/rhscl/httpd-24-rhel7:latest | head -10
{
        "Name": "registry.access.redhat.com/rhscl/httpd-24-rhel7",
        "Digest":
"sha256:02152fd99c0bcfae06af21301ad92ffa122a46e537465d2b6f064f56e5c0685f",
        "RepoTags": [
        "2.4-170.1638430400-source",
        "2.4-170",
        "2.4-172",
        "2.4-146-source",
        "2.4-136.1614612498",
        "2.4-170.1638430400",
```

Compare to

```
[student@workstation ~]$ skopeo inspect
docker://registry.access.redhat.com/rhscl/httpd-24-rhel7:2.4-172 | head -10
{
        "Name": "registry.access.redhat.com/rhscl/httpd-24-rhel7",
        "Digest":
"sha256:ed835f1a45efb7dfd62894274692f494ddbf83d1072019ecafc040574cce5886",
        "RepoTags": [
        "2.4-170.1638430400-source",
        "2.4-170",
        "2.4-172",
        "2.4-146-source",
        "2.4-136.1614612498",
        "2.4-170.1638430400",
```

We could make a local copy using the tag “2.4-172” but lets try using this digest:

```
[student@workstation storage]$ podman pull
registry.access.redhat.com/rhscl/httpd-24-rhel7@sha256:ed835f1a45efb7dfd62894274692
f494ddbf83d1072019ecafc040574cce5886
```

```
…SNIP…
fcea1b0658e6a351aec4119d8c9ee2adb725e151536b98aa8c13d4c6b8e8647b
```

```
[student@workstation storage]$ podman images
registry.access.redhat.com/rhscl/httpd-24-rhel7  <none>     fcea1b0658e6  2 months ago  329
MB
```

We see later how we can assign a local tag to this image if we want.

REGISTRY AUTHENTICATION

See https://access.redhat.com/RegistryAuthentication

- Some container registries require authentication.

```
[student@workstation ~]$ podman pull registry.redhat.io/rhel7
Trying to pull registry.redhat.io/rhel7...Failed
error pulling image "registry.redhat.io/rhel7": unable to pull registry.redhat.io/rhel7: unable to pull
image: Error determining manifest MIME type for docker://registry.redhat.io/rhel7:latest: unable
to retrieve auth token: invalid username/password
```

```
[student@workstation ~]$ podman login -u rhn-support-ablum registry.redhat.io
Password:
Login Succeeded!
[student@workstation ~]$ podman pull registry.redhat.io/rhel7
Trying to pull registry.redhat.io/rhel7...Getting image source signatures
Copying blob
sha256:c9281c141a1bfec06e291d2ad29bfdedfd10a99d583fc0f48d3c26723ebe0761
…
```

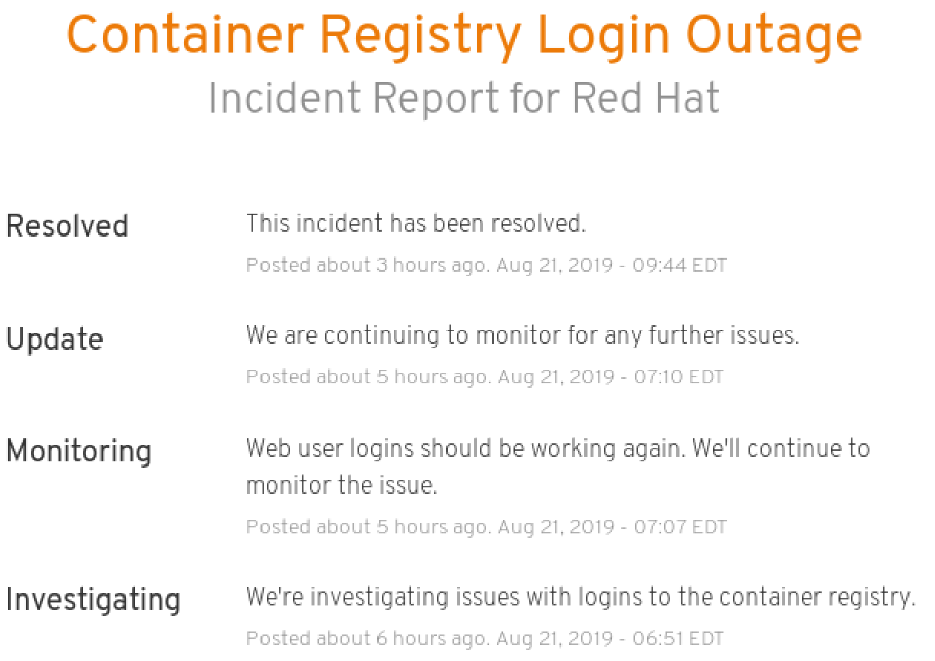
Although this may be deprecated at some point, this doesn’t require authentication:

```
[student@workstation ~]$ podman pull registry.access.redhat.com/rhel
Trying to pull registry.access.redhat.com/rhel...Getting image source signatures
Skipping fetch of repeat blob
sha256:c9281c141a1bfec06e291d2ad29bfdedfd10a99d583fc0f48d3c26723ebe0761
```

- Mention service accounts and why they would be a good idea
    - https://access.redhat.com/terms-based-registry/
    -

https://access.redhat.com/terms-based-registry/#/token/ablum-rhel8-training

https://status.redhat.com/incidents/bjqjyxcknf86


Container Registry Login Outage

Incident Report for Red Hat

Resolved	This incident has been resolved. Posted about 3 hours ago. Aug 21, 2019 - 09:44 EDT
Update	We are continuing to monitor for any further issues. Posted about 5 hours ago. Aug 21, 2019 - 07:10 EDT
Monitoring	Web user logins should be working again. We'll continue to monitor the issue. Posted about 5 hours ago. Aug 21, 2019 - 07:07 EDT
Investigating	We're investigating issues with logins to the container registry. Posted about 6 hours ago. Aug 21, 2019 - 06:51 EDT

https://redhat.service-now.com/surl.do?n=INC0930151

“”

When performing user credential auth to the registry there is a dependence on a restricted party screening service external to Red Hat. In this case that service looks like it had some issues and returned errors and then false export blocks. We need to follow up with the vendor to know what happened on their end. The important point here is that access to the registry through web credentials is not considered a C1 function because of this extra dependency. Resilient, production integration with the registry should always use an auth token that can be generated from the customer portal. The relevant documentation is here:

https://access.redhat.com/RegistryAuthentication

Any customer that is using user credentials for access to the registry in a production set up should be referred to this section https://access.redhat.com/RegistryAuthentication#registry-service-accounts-for-shared-environments-4

where it reads:

Registry Service Accounts for Shared Environments
To consume container images from registry.redhat.io in shared environments such as OpenShift, it is recommended for an administrator to use a Registry Service Account, also referred to as authentication tokens, in place of an individual's Customer Portal credentials.

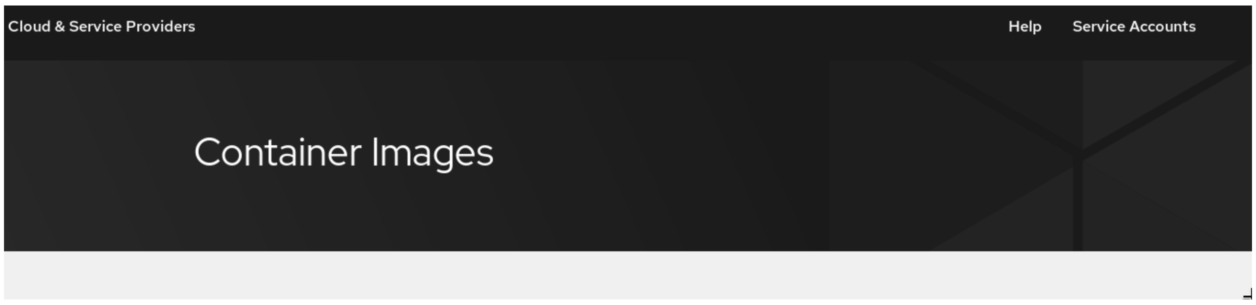
Service Accounts are a mechanism provided to a Customer Portal organization, used exclusively for authenticating to and retrieving content from registry.redhat.io. The use of Service Accounts is encouraged to prevent the need to use Customer Portal credentials on shared systems, in contrast to Customer Portal accounts, Registry Service Accounts are resilient to some security controls applied to Customer Portal accounts, such as mandated password resets.

This outage is a good example of why customers should be using service accounts for any resilient registry integration. Any cases that have been open by customers against this outage are an opportunity to better **educate** the customers as to the preferred auth method for our terms based registry.
“”

To login using a service account:

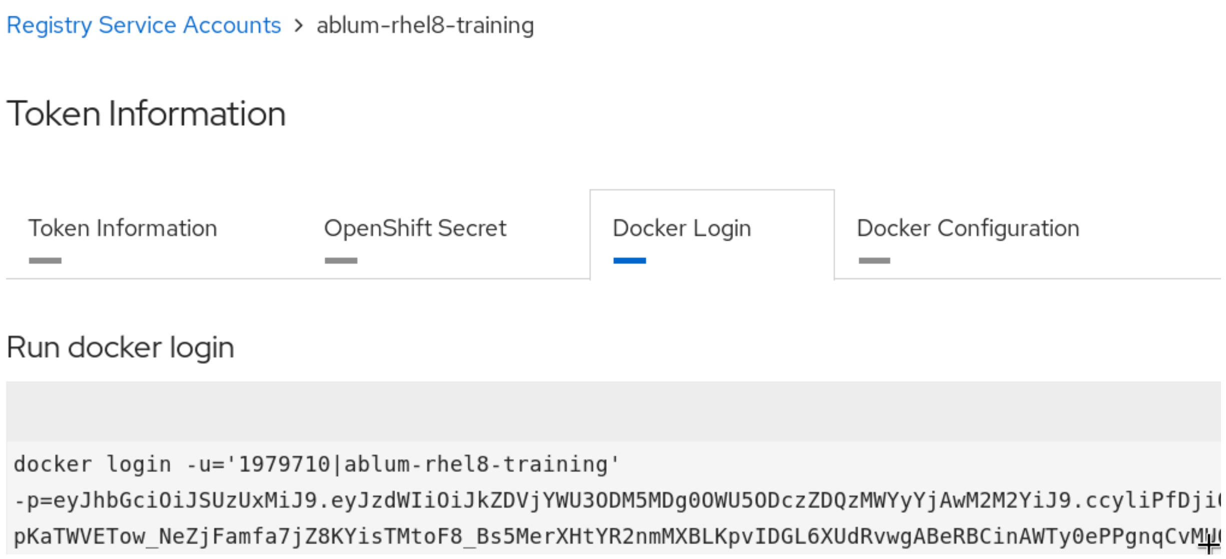
Navigate to https://catalog.redhat.com/software/containers/explore

Click on Service Accounts in upper right:



Find your service account (or create one).

Click on Docker Login:



Run docker login

```
docker login -u='1979710|ablum-rhel8-training'
-p=eyJhbGciOiJSUzUxMiJ9.eyJzdWIiOiJkZDVjYWU3ODM5MDg0OWU5ODczZDQzMWYyYjAwM2M2YiJ9.ccyliPfDji
pKaTWVETow_NeZjFamfa7jZ8KYisTMtoF8_Bs5MerXHtYR2nmMXBLKpvIDGL6XUdRvwgABeRBCinAWTy0ePPgnqCvM
```

[root@workstation ~]# **podman login -u='1979710|ablum-rhel8-training' -p=eyJhbGciOiJSUzUxMiJ9.eyJzdWIiOiJkZDVjYWU3ODM5MDg0OWU5ODczZDQzMWYyYjAwM2M2YiJ9.ccyliPfDjiOFKseSYF5e0LiC-pKaTWVETow_NeZjFamfa7jZ8KYisTMtoF8_Bs5MerXHtYR2nmMXBLKp… registry.redhat.io**

[root@workstation ~]# **podman pull registry.redhat.io/rhel7**

What is the expiration for authentication access tokens?

Normal username and password authentication with registry.redhat.io will result in the user's info being cached in **/run/user/1000/containers/auth.json**

```
{
   "auths": {
          "registry.redhat.io": {
                   "auth": "asdkfjkdasjfjjsadklfj"
          }
   }
}
```

When podman runs commands (like pull) podman will use these credentials to obtain a token that is short-lived (300s). For example,

```
[ablum@badger ~]$ curl -Lv -u 'rhn-support-ablum:password'
"https://sso.redhat.com/auth/realms/rhcc/protocol/redhat-docker-v2/auth?service=docker
-registry&client_id=curl&scope=repository:rhel:pull"
```

```
bkYzs3tRYjVETakY","expires_in":300,"issued_at":"2020-05-07T11:28:46Z"}
```

When using service accounts for authentication, you can decode parts.

How to use images from a registry for disconnected customers?

http://post-office.corp.redhat.com/archives/sbr-containers/2019-August/msg00012.html
1.) run on a low-side machine (or one that is allowed to connect to the
internet):

```
# yum module install container-tools

# podman login -u rhn-support-ablum registry.redhat.io

# skopeo copy docker://registry.redhat.io/rhel7:latest
dir:/root/redhat_io/rhel7

# cd /root/redhat_io/rhel7 && tar -cvf /root/redhat_io_images/rhel7.tar
* && cd ..
```

2.) loop for other images needed (or find a different way to sync the
registry?)

3.) Create an ISO from tar'd images:

```
# yum install genisoimage

# mkisofs -o /root/redhat_io.iso /root/redhat_io_images

# dd if=/root/redhat_io.iso of=/dev/cdrom
```

4.) Carry ISO to disconnected network/system and copy tar'd images from
ISO

5.) Follow https://access.redhat.com/solutions/4175551 to upload image tar files to satellite using hammer

MANIPULATING CONTAINER IMAGES

Working around container image signatures.

```
[root@workstation ~]# podman images
REPOSITORY                           TAG     IMAGE ID       CREATED        SIZE
registry.redhat.io/rhscl/httpd-24-rhel7   latest a1fdc13b5792  8 days ago    324MB
registry.access.redhat.com/rhel      latest  31cd91012c57   10 days ago         214MB
registry.redhat.io/rhel7             latest  31cd91012c57   10 days ago         214MB
registry.lab.example.com/rhel7       7.5-404  e64297b706b7  13 months ago  211MB
```

```
[student@workstation ~]$ podman save -o httpd.tar
registry.access.redhat.com/rhscl/httpd-24-rhel7:latest
Getting image source signatures
Checking if image destination supports signatures
Error: Can not copy signatures to docker-archive:httpd.tar: Storing signatures for docker tar files
is not supported
```

Trying to save the ubi8:latest gives the same

```
[student@workstation ~]$ podman save 1264065f6ae8 -o ubi8.tar
Getting image source signatures
Checking if image destination supports signatures
Error: Can not copy signatures to docker-archive:ubi8.tar: Storing signatures for docker
tar files is not supported
```

Is this true for all images ?

```
[student@workstation ~]$ podman pull quay.io/ajblum/hello-openshift:latest
Trying to pull quay.io/ajblum/hello-openshift:latest...
Getting image source signatures
Copying blob a3ed95caeb02 done
Copying blob b30065c58b6f done
Copying config 7af3297a3f done
Writing manifest to image destination
Storing signatures
```

```
7af3297a3fb4487b740ed6798163f618e6eddea1ee5fa0ba340329fcae31c8f6
[student@workstation ~]$ podman save -o hello-openshift.tar
quay.io/ajblum/hello-openshift:latest
Getting image source signatures
Copying blob 5f70bf18a086 done
Copying blob da0e4d9121c7 done
Copying config 7af3297a3f done
Writing manifest to image destination
Storing signatures
```

Nope, works. Images from RedHat are signed using the same gpg key rpm packages are signed with. Podman uses the same rpm-gpg key to verify images from redhat.

```
[student@workstation ~]$ cat /etc/containers/policy.json
[student@workstation ~]$ podman image trust show
default                 accept
registry.access.redhat.com  signedBy          security@redhat.com, security@redhat.com
https://access.redhat.com/webassets/docker/content/sigstore
registry.redhat.io      signedBy          security@redhat.com, security@redhat.com
https://registry.redhat.io/containers/sigstore
                        insecureAcceptAnything
```

Blog on gpg signatures used with RH images:
https://developers.redhat.com/blog/2019/10/29/verifying-signatures-of-red-hat-container-images

Instead, we could copy the image without the signature:

```
[student@workstation ~]$ skopeo copy docker://registry.access.redhat.com/ubi8:latest
containers-storage:localhost/ubi8:latest --remove-signatures
Copying blob 028bdc977650 skipped: already exists
Copying blob 0c673eb68f88 [--------------------------------------] 0.0b / 0.0b
Copying config 2fd9e14788 done
Writing manifest to image destination
Storing signatures
[student@workstation ~]$ podman images
REPOSITORY                              TAG          IMAGE ID       CREATED       SIZE
registry.access.redhat.com/rhscl/httpd-24-rhel7  latest    7b8d40facfb4  2 weeks ago  330 MB
registry.access.redhat.com/rhel7             latest   e2c37c467077  3 weeks ago  216 MB
localhost/ubi8                          latest   2fd9e1478809  3 weeks ago  225 MB
registry.access.redhat.com/ubi8              latest   2fd9e1478809  3 weeks ago  225 MB
[student@workstation ~]$ podman save -o ubi8.tar ubi8:latest
Getting image source signatures
```

```
Copying blob 77bf63677b0c done
Copying blob 3edb0d97db5c done
Copying config 2fd9e14788 done
Writing manifest to image destination
Storing signatures
```

This should be avoided in future versions of podman
https://github.com/containers/podman/pull/7956

How can you remove images using `podman system prune` ?

From man 1 podman-system-prune:

podman system prune removes all unused containers (both dangling and unreferenced), pods and optionally, volumes from local storage.

```
[student@workstation ~]$ podman run -d -v myvol:/var/www/html rhscl/httpd-24-rhel7
6e5f9dc5474c4853e0bf01e508ba2471bbc190830a1c51b06204dd436846f07e

[student@workstation ~]$ podman volume list
DRIVER   VOLUME NAME
local    myvol

[student@workstation ~]$ podman volume inspect myvol
[
        {
        "Name": "myvol",
        "Driver": "local",
        "Mountpoint": "/home/student/.local/share/containers/storage/volumes/myvol/_data",
        "CreatedAt": "2021-09-22T16:03:27.230242031-04:00",
        "Labels": {

        },
        "Scope": "local",
        "Options": {

        }
        }
]

[student@workstation ~]$ podman inspect 6e --format '{{.Mounts}}'
[{volume myvol /home/student/.local/share/containers/storage/volumes/myvol/_data
/var/www/html local  [noexec nosuid nodev rbind] true rprivate}]

[student@workstation ~]$ podman exec -it 6e /bin/bash
bash-4.2$ df -h
Filesystem      Size  Used Avail Use% Mounted on
fuse-overlayfs  9.9G  6.3G  3.7G  64% /
tmpfs            64M     0   64M   0% /dev
tmpfs           580M  100K  580M   1% /etc/hosts
shm              63M     0   63M   0% /dev/shm
/dev/vda3       9.9G  6.3G  3.7G  64% /var/www/html
tmpfs           2.9G     0  2.9G   0% /sys/fs/cgroup
devtmpfs        2.8G     0  2.8G   0% /dev/tty
tmpfs           2.9G     0  2.9G   0% /proc/acpi
tmpfs           2.9G     0  2.9G   0% /proc/scsi
tmpfs           2.9G     0  2.9G   0% /sys/firmware
tmpfs           2.9G     0  2.9G   0% /sys/fs/selinux
bash-4.2$ touch /var/www/html/index.html
bash-4.2$ exit
exit

[student@workstation ~]$ ls
/home/student/.local/share/containers/storage/volumes/myvol/_data
index.html


[student@workstation ~]$ podman system prune --volumes

WARNING! This will remove:
        - all stopped containers
        - all volumes not used by at least one container
        - all stopped pods
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Pods
Deleted Containers
Deleted Volumes
[student@workstation ~]$
[student@workstation ~]$
```

```
[student@workstation ~]$ podman ps
CONTAINER ID  IMAGE                                          COMMAND               CREATED
STATUS         PORTS  NAMES
6e5f9dc5474c  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  2
minutes ago  Up 2 minutes ago         youthful_mahavira
[student@workstation ~]$ podman stop 6e
6e5f9dc5474c4853e0bf01e508ba2471bbc190830a1c51b06204dd436846f07e
[student@workstation ~]$ podman system prune --volumes

WARNING! This will remove:
        - all stopped containers
        - all volumes not used by at least one container
        - all stopped pods
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Pods
Deleted Containers
6e5f9dc5474c4853e0bf01e508ba2471bbc190830a1c51b06204dd436846f07e
Deleted Volumes
myvol
[student@workstation ~]$ podman volume list
[student@workstation ~]$

[student@workstation ~]$ mkdir mydir
[student@workstation ~]$ podman unshare chown 1001:1001 mydir
[student@workstation ~]$ podman unshare chcon -t container_file_t mydir
[student@workstation ~]$ podman run -d -v /home/student/mydir:/var/www/html
rhscl/httpd-24-rhel7
5e0175ff9761696c20887cd450a4500552e3ed877cfd62fc4f43fb4c0d39c03f

[student@workstation ~]$ podman volume list
[student@workstation ~]$
[student@workstation ~]$ podman inspect 5e --format '{{.Mounts}}'
[{bind  /home/student/mydir /var/www/html   [rbind] true rprivate}]

[student@workstation ~]$ ls -ldZ /home/student/mydir
drwxrwxr-x. 2 101000 101000 unconfined_u:object_r:container_file_t:s0 6 Sep 22 16:07
/home/student/mydir
[student@workstation ~]$ podman stop 5e
5e0175ff9761696c20887cd450a4500552e3ed877cfd62fc4f43fb4c0d39c03f
[student@workstation ~]$ podman system prune --volumes
```

```
WARNING! This will remove:
        - all stopped containers
        - all volumes not used by at least one container
        - all stopped pods
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Pods
Deleted Containers
5e0175ff9761696c20887cd450a4500552e3ed877cfd62fc4f43fb4c0d39c03f
Deleted Volumes
```

See also https://bugzilla.redhat.com/show_bug.cgi?id=1811570#c15 :

“”
podman system prune should NOT be removing buildah containers/images.
“”

Modifying images using podman commit

- One strategy would be to run a container from an image, modify it, and then commit it to a new image
- Uses `podman commit`

```
[student@workstation ~]$ podman run -it ubi8 /bin/bash

[root@160bfbd24ae2 /]# echo "ablum was here" > testfile
[root@160bfbd24ae2 /]# vi /etc/motd
This is the message for today
[root@160bfbd24ae2 ~]# rm /root/.cshrc
rm: remove regular file '/root/.cshrc'? yes
[root@160bfbd24ae2 /]# exit
exit

[student@workstation ~]$ podman ps -a
CONTAINER ID   IMAGE                                  COMMAND     CREATED      STATUS
        PORTS   NAMES                IS INFRA
160bfbd24ae2  registry.access.redhat.com/ubi8:latest            /bin/bash              3
minutes ago  Exited (0) 18 seconds ago                        nice_faraday

[student@workstation ~]$ podman diff 160
```

```
A /ablum_was_here
C /etc
C /etc/motd
C /root
A /root/.bash_history
D /root/.cshrc

[student@workstation ~]$ podman inspect 160 --format '{{.GraphDriver.Data.UpperDir}}'
[student@workstation ~]$ ls -lR
/home/student/.local/share/containers/storage/overlay/13ff5bb892ebf396a0a7402b93d61f1
d0c0df1dccf81a3b5d29de36283723f52/diff
[student@workstation ~]$ ls -l
/home/student/.local/share/containers/storage/overlay/13ff5bb892ebf396a0a7402b93d61f1
d0c0df1dccf81a3b5d29de36283723f52/diff/root/.cshrc
c---------. 2 student student 0, 0 Jun 29 08:27
/home/student/.local/share/containers/storage/overlay/13ff5bb892ebf396a0a7402b93d61f1d0c0
df1dccf81a3b5d29de36283723f52/diff/root/.cshrc

[student@workstation ~]$ podman commit 584e167bd18c ubi8:modified

[student@workstation ~]$ podman images
REPOSITORY                    TAG      IMAGE ID       CREATED        SIZE
localhost/rhel7               modified  d48953c14c05   2 minutes ago   211MB
registry.lab.example.com/rhel7   latest        e64297b706b7   13 months ago   211MB

[student@workstation ~]$ podman run d6 ls -l /root/.cshrc
ls: cannot access '/root/.cshrc': No such file or directory
[student@workstation ~]$ podman run d6 cat /etc/motd
this is the message for today
[student@workstation ~]$ podman run d6 cat /ablum_was_here
[student@workstation ~]$ podman run d6 ls -l /ablum_was_here
-rw-r--r--. 1 root root 0 Jun 29 12:27 /ablum_was_here


[student@workstation ~]$ mkdir ~/mydata2

[student@workstation ~]$ chcon -Rv -t container_file_t ~/mydata2
changing security context of '/mydata2'

[root@workstation /]# podman run -it -v /home/student/mydata2:/opt rhel7 /bin/bash
[root@ee9d566b0357 /]# df /opt
Filesystem      1K-blocks       Used Available Use% Mounted on
/dev/vda1       104845184 5596740  99248444   6% /opt
```

```
[root@ee9d566b0357 /]# touch /opt/important
[root@ee9d566b0357 /]# exit
exit

[root@workstation /]# podman ps -a
CONTAINER ID   IMAGE                                  COMMAND              CREATED
STATUS                 PORTS   NAMES                 IS INFRA
ee9d566b0357   registry.lab.example.com/rhel7:latest   /bin/bash            41 seconds ago
Exited (0) 6 seconds ago       elastic_johnson   false
6b25e5020b56   localhost/rhel7:modified               tail /var/log/yum.l...   7 minutes ago   Exited
(0) 7 minutes ago       modest_gates            false
584e167bd18c   registry.lab.example.com/rhel7:latest   /bin/bash            9 minutes ago Exited
(0) 9 minutes ago       modest_mestorf          false
[root@workstation /]# podman diff ee9d566b0357
C /root
A /root/.bash_history
```

- NOTE /opt/important is NOT listed in diff
- Use `**podman inspect**` to see the mounts (volumes)

```
        "Mounts": [
{
        "destination": "/opt",
        "type": "bind",
        "source": "/home/student/mydata1",
        "options": [
        "rbind",
        "rw",
        "rprivate"
```

PUBLISHING IMAGES TO A REGISTRY

How to resolve error “image is signed or the destination specifies a digest” when pushing to quay.io ?

```
[ablum@badger ~]$ podman push 52de04277b39 quay.io/ajblum/mytest:latest
Getting image source signatures
Checking if image destination supports signatures
Error: Copying this image requires changing layer representation, which is not possible
(image is signed or the destination specifies a digest)
```

This fails with new versions of podman

```
[ablum@badger ~]$ podman version
Version:          3.4.4
API Version:  3.4.4
Go Version:   go1.16.8
Built:              Wed Dec  8 15:45:07 2021
OS/Arch:          linux/amd64
```

```
[ablum@badger ~]$ podman push 52de04277b39 quay.io/ajblum/mytest:latest
--remove-signatures
Copying blob c8013a2772b6 done
Copying blob 7699752e6ed6 done
Copying config 52de04277b done
Writing manifest to image destination
Storing signatures
```

Blog on gpg signatures used with RH images:
https://developers.redhat.com/blog/2019/10/29/verifying-signatures-of-red-hat-container-images

See also `podman image trust show`

Also, this might cause a different issue on quay which is enforcing strict schema standard related to signatures. It will cause a problem for older clients who try to pull: https://issues.redhat.com/browse/PROJQUAY-3285 The workaround is to add **--format v2s1** when pulling or copying using those older clients

How would a local image be shared back to a private registry ?

`podman push`

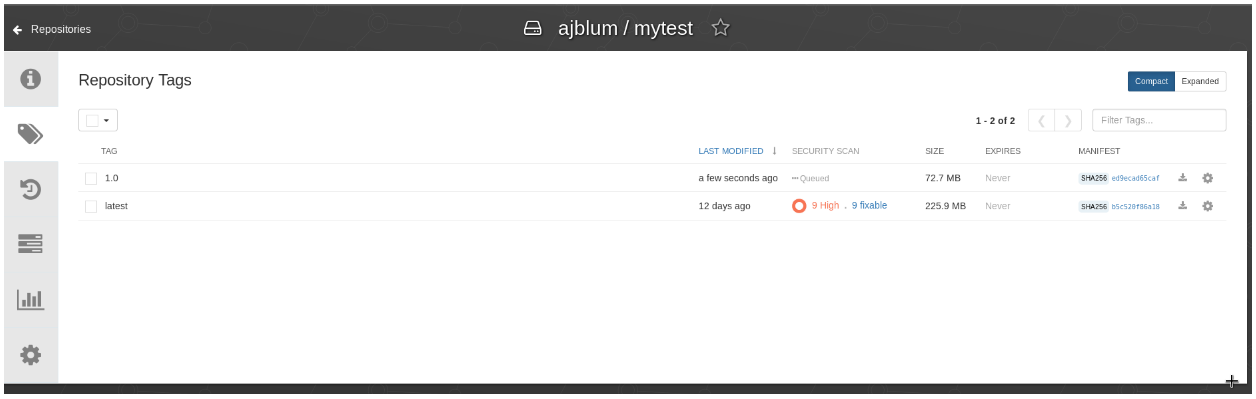
```
[root@workstation /]# **podman push d48953c14c05 
registry.lab.example.com/rhel7:modified**
Getting image source signatures
Copying blob 
sha256:24a5c6254cd9693d64581b6f3df5e4ee551cfd5429cf25301d12afa82ac91037
 200.88 MB / 200.88 MB [======================================================] 
41s

[root@workstation /]# **podman push d48953c14c05 registry.redhat.io/rhel7**
Getting image source signatures
Copying blob 
sha256:24a5c6254cd9693d64581b6f3df5e4ee551cfd5429cf25301d12afa82ac91037
 8 B / 200.88 MB [>-----------------------------------------------------] 0s
Error copying image to the remote destination: Error writing blob: Error initiating layer upload to 
/v2/rhel7/blobs/uploads/ in registry.redhat.io: error parsing HTTP 403 response body: invalid 
character '<' looking for beginning of value: "<HTML><HEAD>\n<TITLE>Access 
Denied</TITLE>\n</HEAD><BODY>\n<H1>Access Denied</H1>\n \nYou don't have 
permission to access 
\"http&#58;&#47;&#47;registry&#46;redhat&#46;io&#47;v2&#47;rhel7&#47;blobs&#47;uploads&
#47;\" on this 
server.<P>\nReference&#32;&#35;18&#46;ae8d4017&#46;1564688844&#46;35231e6\n</BOD
Y>\n</HTML>\n"
```

What about pushing to public registries ?

https://cloud.docker.com/repository/list

```
[student@workstation mytmp]$ **podman run -it 272209ff0ae5 /bin/bash**

[root@11001ec73d8d /]# **echo test > ablum_was_here**
[root@11001ec73d8d /]# **rm /etc/motd**
rm: remove regular empty file '/etc/motd'? **yes**
[root@11001ec73d8d /]# **echo '#ablum_was_here' >> /etc/hosts**
[root@11001ec73d8d /]# **cat /etc/hosts**

[root@11001ec73d8d /]# **exit**
Exit
```

```
[student@workstation mytmp]$ **podman ps -a | head -2**
CONTAINER ID  IMAGE                                        COMMAND            CREATED
          STATUS                    PORTS                    NAMES
11001ec73d8d  registry.access.redhat.com/ubi8:latest                 /bin/bash                3
minutes ago    Exited (0) 22 seconds ago                                   nervous_banach
[student@workstation mytmp]$ **podman diff 1100**
A /ablum_was_here
C /root
A /root/.bash_history
C /etc
D /etc/motd

[student@workstation mytmp]$ **podman commit 1100 mytest:1.0**
Getting image source signatures
Copying blob 1a6543399d61 skipped: already exists
Copying blob f0a77c369efd skipped: already exists
Copying blob 96f1a9906488 done
Copying config 085e96ca8c done
Writing manifest to image destination
Storing signatures
085e96ca8c163fdd011c2d246984b3cb3f781c90e04ff2fa381c646d664eb417
[student@workstation mytmp]$
[student@workstation mytmp]$
[student@workstation mytmp]$ **podman login quay.io**
Username: ajblum
Password:
Login Succeeded!
[student@workstation mytmp]$
[student@workstation mytmp]$ **podman push mytest:1.0 quay.io/ajblum/mytest:13.0**
Getting image source signatures
Copying blob f0a77c369efd done
Copying blob 96f1a9906488 done
Copying blob 1a6543399d61 done
Copying config 085e96ca8c done
Writing manifest to image destination
Copying config 085e96ca8c done
Writing manifest to image destination
Storing signatures
```



Repository Tags

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
1.0	a few seconds ago	Queued	72.7 MB	Never	SHA256 ed9ecad65caf
latest	12 days ago	9 High . 9 fixable	225.9 MB	Never	SHA256 b5c520f86a18

```
[student@workstation mytmp]$ podman rmi mytest:1.0
Untagged: localhost/mytest:1.0
Deleted: 085e96ca8c163fdd011c2d246984b3cb3f781c90e04ff2fa381c646d664eb417
[student@workstation mytmp]$
[student@workstation mytmp]$
[student@workstation mytmp]$ podman images
REPOSITORY                              TAG    IMAGE ID      CREATED      SIZE
(missing mytest:1.0)

[student@workstation mytmp]$ podman run quay.io/ajblum/mytest:13.0 cat
ablum_was_here
Trying to pull quay.io/ajblum/mytest:13.0...
Getting image source signatures
Copying blob ebd715ce2661 skipped: already exists
Copying blob b2af222bc479 skipped: already exists
Copying blob e1bb39f7b07a done
Copying config 085e96ca8c done
Writing manifest to image destination
Storing signatures
test
```

What services are running on a registry ?

```
[root@workstation /]# ssh root@registry.lab.example.com

[root@services ~]# netstat -tunap | grep -i 443
tcp      0        0 172.25.250.13:443         0.0.0.0:*                LISTEN        3247/registry
```

```
[root@services ~]# ps -ef | grep 3247
root      3247     1  0 14:27 ?      00:00:00 /usr/bin/registry serve
/etc/docker-distribution/registry/config.yml
root      4041  3993  0 15:16 pts/0     00:00:00 grep --color=auto 3247

[root@services ~]# systemctl | grep docker
  docker-distribution.service                                              loaded active running   v2
Registry server for Docker

[root@services ~]# rpm -qf /usr/lib/systemd/system/docker-distribution.service
docker-distribution-2.6.2-2.git48294d9.el7.x86_64

[root@services ~]# rpm -qc docker-distribution
/etc/docker-distribution/registry/config.yml
version: 0.1
log:
  fields:
        service: registry
storage:
        cache:
        layerinfo: inmemory
        filesystem:
        rootdirectory: /var/lib/registry
        delete:
        enabled: true
http:
        addr: registry.lab.example.com:443
        host: https://registry.lab.example.com
        tls:
        certificate: /etc/pki/tls/certs/example.com.crt
        key: /etc/pki/tls/private/example.com.key


[root@services ~]# rpm -qd docker-distribution
/usr/share/doc/docker-distribution-2.6.2/AUTHORS
/usr/share/doc/docker-distribution-2.6.2/CONTRIBUTING.md
/usr/share/doc/docker-distribution-2.6.2/LICENSE
/usr/share/doc/docker-distribution-2.6.2/MAINTAINERS
/usr/share/doc/docker-distribution-2.6.2/README.md
```

https://github.com/docker/distribution/blob/master/docs/spec/api.md

NOTE1: Red Hat Quay is the future !

NOTE2: skopeo is a better tool (vs podman pull/push/save/load) for sync'ing content around

CHAPTER 5 CREATING CUSTOM IMAGES

BUILDING CUSTOM CONTAINER IMAGES WITH DOCKERFILES

Why do I get permission denied building images after running su - (or sudo su -) ?

```
[root@workstation echo1]# podman build -t mytest:1.0 .
STEP 1: FROM registry.redhat.io/rhel7:latest
Error: error creating build container: Error initializing source
docker://registry.redhat.io/rhel7:latest: unable to retrieve auth token: invalid username/password:
unauthorized: Please login to the Red Hat Registry using your Customer Portal credentials.
Further instructions can be found here: https://access.redhat.com/RegistryAuthentication
```

Check:

```
[root@workstation echo1]# echo ${XDG_RUNTIME_DIR}
```

```
[root@workstation echo1]# **man pam_systemd**
```

“”	On login, this module — in conjunction with systemd-logind.service — ensures the following:

	1. If it does not exist yet, the user runtime directory /run/user/$UID is either created or mounted as new "tmpfs" file system with quota
	applied, and its ownership changed to the user that is logging in.

	2. The $XDG_SESSION_ID environment variable is initialized. If auditing is available and pam_loginuid.so was run before this module (which
	is highly recommended), the variable is initialized from the auditing session id (/proc/self/sessionid). Otherwise, an independent
	session counter is used.””

Workaround is just pass the auth file path in the build command:

```
[root@workstation echo1]# podman build -t mytest:1.0 --authfile /run/user/0/containers/auth.json .
STEP 1: FROM registry.redhat.io/rhel7:latest
Getting image source signatures
```

ENTRYPOINT VS CMD

Understanding this with example:

```
[root@workstation ~]# mkdir echo1
[root@workstation ~]# cd echo1
[root@workstation echo1]# vim Containerfile
FROM ubi8
ENTRYPOINT ["/usr/bin/echo", "Hello world!"]
[root@workstation date1]# podman build -t ubi8:echo1 .
[root@workstation echo1]# podman run ubi8:echo1
Hello world!
[root@workstation echo1]# podman inspect 3a97b163f35a --format '{{.ContainerConfig.Entrypoint}}'
[/usr/bin/echo Hello world!]
```

Now, lets add in a CMD instruction:

```
[root@workstation echo1]# vim Containerfile


FROM ubi8
ENTRYPOINT ["/usr/bin/echo"]
CMD ["Hello world!"]

[root@workstation echo1]# podman build -t ubi8:echo2 .
[root@workstation echo1]# podman run ubi8:echo2
Hello world!
```

Let's override the CMD built in the image when running these two, just to understand the impact they have:

```
[root@workstation echo1]# podman run rhel7:echo2 hello foo
hello foo

[root@workstation echo1]# podman run rhel7:echo1 hello foo
Hello world! hello foo

[root@workstation echo1]# podman run rhel7:echo2 -ne "hello foo"
hello foo[root@workstation echo1]#

[root@workstation echo1]# podman run rhel7:echo1 -ne "hello foo"
Hello world! -ne hello foo


[root@workstation echo1]# podman ps -a --no-trunc --format 'table {{.Image}}
{{.Command}}'
IMAGE                    COMMAND
localhost/rhel7:echo1.1   /usr/bin/echo -ne hello foo
localhost/rhel7:echo1  /usr/bin/echo Hello world! -ne hello foo
```

Notice how the commands built vary due to the differences in the entrypoint.

EXEC vs SHELL

Now, let's understand "exec" form vs "shell" form.

```
[root@workstation echo1]# vim Dockerfile
FROM rhel7
```

```
ENV FOO "Hello World!"
ENTRYPOINT ["/usr/bin/echo","$FOO"]
```

```
[root@workstation echo1]# podman build -t rhel7:echo1.2 .
[root@workstation echo1]# podman run rhel7:echo1.2
$FOO
```

Did it do what we were expecting ? Why not ?
Let’s override the entrypoint and check out the environment:

```
[root@workstation echo1]# podman run --entrypoint "env" rhel7:echo1.2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
TERM=xterm
HOSTNAME=8102a385729a
container=oci
FOO=Hello World!
HOME=/root
```

The reason ? “Exec” form vs “shell” form. With exec form, the entrypoint executable isn’t run inside a shell. Environment variables are available for use inside a shell. So, let’s change to use the shell form and observe the behavior:

```
 [root@workstation echo1]# vim Dockerfile
FROM rhel7
ENV FOO "Hello World!"
ENTRYPOINT /usr/bin/echo $FOO
[root@workstation echo1]# podman build -t rhel7:echo1.3 .
[root@workstation echo1]# podman run rhel7:echo1.3
Hello World!
```

```
[root@workstation echo1]# podman ps -a --no-trunc
CONTAINER ID                                                      IMAGE                    COMMAND
        CREATED                 STATUS                              PORTS   NAMES                  IS
INFRA
173d53f94eb8b48149407456512800f27355c925302e9ce9bba789e5d501ea9a
localhost/rhel7:echo1.4   /bin/sh -c /usr/bin/echo $FOO                About a minute ago   Exited
(0) About a minute ago          nifty_mccarthy            false
```

Notice how the entrypoint is wrapped with /bin/sh -c “   “. This allows the shell to expand the variable with the Hello World! string.

Also, now lets see what happens when we pass cmd arguments to the podman run:

```
[root@workstation echo1]# podman run rhel7:echo1.3 whoot
Hello World!
[root@workstation echo1]# podman run rhel7:echo1.3 -ne
Hello World!
```

Notice how the CMD is ignored here when using the shell form for entrypoint

So, what’s preferred exec or shell ? exec preferred in most cases.

Exec PRO: natively allows for the executable to handle signals sent to it
Shell PRO: prevents any CMD from affecting the way the entrypoint runs

Stopping containers:: If you are using the shell format, start the ENTRYPOINT with the exec command or the process wont be able to handle the SIGTERM and then a SIGKILL will be sent.

RUN AND LAYERS

```
[root@workstation ~]# mkdir myls
[root@workstation myls]# vim Dockerfile
FROM rhel7
ENTRYPOINT ["/usr/bin/ls"]
RUN touch /var/tmp/data
[root@workstation myls]# podman build -t rhel7:myls1 .
[root@workstation myls]# podman run rhel7:myls1 /var/tmp/
data
[root@workstation myls]# podman run rhel7:myls1 -l /var/tmp/
total 0
-rw-r--r--. 1 root root 0 Aug 15 17:02 data
```

```
[root@workstation myls]# vim Dockerfile

FROM rhel7
ENTRYPOINT ["/usr/bin/ls"]
RUN touch /var/tmp/data
RUN touch /var/tmp/data1
RUN touch /var/tmp/data2
RUN touch /var/tmp/data3
RUN touch /var/tmp/data4
RUN touch /var/tmp/data5
[root@workstation myls]# podman build -t rhel7:myls1.1 .
```

```
[root@workstation myls]# podman run rhel7:myls1.1 -l /var/tmp/
total 0
-rw-r--r--. 1 root root 0 Aug 15 17:02 data
-rw-r--r--. 1 root root 0 Aug 15 17:03 data1
-rw-r--r--. 1 root root 0 Aug 15 17:03 data2
-rw-r--r--. 1 root root 0 Aug 15 17:03 data3
-rw-r--r--. 1 root root 0 Aug 15 17:03 data4
-rw-r--r--. 1 root root 0 Aug 15 17:03 data5

[root@workstation myls]# podman inspect rhel7:myls1.1 --format
'{{.GraphDriver.Data.LowerDir}}'

"/var/lib/containers/storage/overlay/bac6c053b9c1da5358c139ebdf4c16560e6db6a64f09592fcc
8b1054b195b059/diff:/var/lib/containers/storage/overlay/e6064a9d0531666c471742660e06913
105e1de507ba72b2692ca86bd4ea145dc/diff:/var/lib/containers/storage/overlay/7730fed410205
03148ad5f44d8c0b9e324ae6f8c3ad47712986f6f6edb5c3e52/diff:/var/lib/containers/storage/ove
rlay/b98054dd86a3c833e6d55689e5849fb09590484a6e3cddcbbe87c39a37a86515/diff:/var/lib/c
ontainers/storage/overlay/66e7744fbe7bd7ef81ae7ede769dbb35e5390923755c323380ded1e7
bdf0cf19/diff:/var/lib/containers/storage/overlay/f9c4bab32ab9a1e25673c7fc3f9f88386767f69e0
db0157b775ffb9477e4b267/diff:/var/lib/containers/storage/overlay/3a5127d99f58d4abfcd3dc42c
6cdf3bbe7dd3a6d9140a92eb8cd54b94997ff82/diff:/var/lib/containers/storage/overlay/3451cffb7
8092cbf2877e44b1fa2774cae891125b5752c5e1c02303bc4ab61a4/diff",

[root@workstation myls]# podman inspect rhel7:myls1 --format
'{{.GraphDriver.Data.LowerDir}}'
"/var/lib/containers/storage/overlay/f9c4bab32ab9a1e25673c7fc3f9f88386767f69e0db0157b775
ffb9477e4b267/diff:/var/lib/containers/storage/overlay/3a5127d99f58d4abfcd3dc42c6cdf3bbe7d
d3a6d9140a92eb8cd54b94997ff82/diff:/var/lib/containers/storage/overlay/3451cffb78092cbf287
7e44b1fa2774cae891125b5752c5e1c02303bc4ab61a4/diff",
```

To avoid these extra layers, combine RUN instructions into one:

```
[root@workstation myls]# vim Dockerfile
FROM rhel7
ENTRYPOINT ["/usr/bin/ls"]
RUN touch /var/tmp/data && \
      touch /var/tmp/data1 && \
      touch /var/tmp/data2 && \
      touch /var/tmp/data3 && \
      touch /var/tmp/data4 && \
      touch /var/tmp/data5
```

```
[root@workstation myls]# podman build -t rhel7:myls1.2 .
[root@workstation myls]# podman inspect rhel7:myls1.2 --format
'{{.GraphDriver.Data.LowerDir}}'
/var/lib/containers/storage/overlay/f9c4bab32ab9a1e25673c7fc3f9f88386767f69e0db0157b775ff
b9477e4b267/diff:/var/lib/containers/storage/overlay/3a5127d99f58d4abfcd3dc42c6cdf3bbe7dd
3a6d9140a92eb8cd54b94997ff82/diff:/var/lib/containers/storage/overlay/3451cffb78092cbf2877
e44b1fa2774cae891125b5752c5e1c02303bc4ab61a4/diff
```

When building we are also keeping around these intermediate layers:

```
[root@workstation myls]# podman images -a
```

To build without saving the intermediate (build) layers:

```
$ podman build --layers=false -t <name:tag> <dir>
```

```
[root@workstation myls]# man podman-build
--layers

        Cache intermediate images during the build process (Default is true).
```

Can you squash the excessive layers when building ?

Yes. Layers can be squashed using **--squash** or **--squash-all**

```
[student@workstation myls]$ podman build --help | grep squash
      --squash                           squash newly built layers into a single new layer
      --squash-all                       Squash all layers into a single layer
```

```
[student@workstation myls]$ cat Containerfile
FROM registry.redhat.io/ubi8:latest
ENTRYPOINT ["/usr/bin/ls"]
RUN touch /var/tmp/data0
RUN touch /var/tmp/data1
RUN touch /var/tmp/data2
RUN touch /var/tmp/data3
RUN touch /var/tmp/data4
RUN touch /var/tmp/data5
RUN touch /var/tmp/data6
```

```
[student@workstation myls]$ podman build -t ubi8:squash --squash .
STEP 1/9: FROM registry.redhat.io/ubi8:latest
STEP 2/9: ENTRYPOINT ["/usr/bin/ls"]
STEP 3/9: RUN touch /var/tmp/data0
STEP 4/9: RUN touch /var/tmp/data1
STEP 5/9: RUN touch /var/tmp/data2
STEP 6/9: RUN touch /var/tmp/data3
STEP 7/9: RUN touch /var/tmp/data4
STEP 8/9: RUN touch /var/tmp/data5
STEP 9/9: RUN touch /var/tmp/data6
COMMIT ubi8:squash
Getting image source signatures
Copying blob b38cb9259677 skipped: already exists
Copying blob 23e15b9ab3f0 skipped: already exists
Copying blob f4cb19500042 done
Copying config 1a958042d3 done
Writing manifest to image destination
Storing signatures
--> 1a958042d30
Successfully tagged localhost/ubi8:squash
1a958042d30d08789a566e09578d503d300b0dcb0e0b1b03ed39aaff885b12e4

[student@workstation myls]$ podman inspect ubi8:squash --format
'{{.GraphDriver.Data.LowerDir}}'
/home/student/.local/share/containers/storage/overlay/f4a999c201294a0a171618e413f1ea7628
562e96d8ac148e00708e421aee56a8/diff:/home/student/.local/share/containers/storage/overlay/
b38cb92596778e2c18c2bde15f229772fe794af39345dd456c3bf6702cc11eef/diff
```

NOTE: There is some storage savings you will get from keeping some of the common layers shared across your container runtime. If you use --squash-all then you will be left with no shared layers missing out on page cache and potentially increasing the overall storage use on your runtime host.

```
[student@workstation myls]$ podman build -t ubi8:squashall --squash-all .
[student@workstation myls]$ podman inspect ubi8:squashall --format
'{{.GraphDriver.Data.LowerDir}}'
<no value>
```

All the data in this case is in the image’s UpperDir which will become a unique LowerDir when running the squashall image.

How can you use “OR” logic instead of “AND” logic like the &&

```
[root@workstation ~]#
[root@workstation ~]# mkdir test1
[root@workstation ~]# cd test1/
[root@workstation test1]# vim Dockerfile
FROM rhel7
ENTRYPOINT ["/usr/bin/ls"]
RUN touch /var/tmp/test1 && touch /var/tmp/test2
RUN touch /var/tmp/foo/bar/test3 || touch /var/tmp/test3
[root@workstation test1]# podman build -t myls:1.0 .
STEP 1: FROM rhel7
Getting image source signatures
Skipping fetch of repeat blob
sha256:00f17e0b37b0515380a4aece3cb72086c0356fc780ef4526f75476bea36a2c8b
Skipping fetch of repeat blob
sha256:305d73a95c8fece2b53a34e040df1c97eb6b7f7cc4e0a7933465f0b7325e3d72
Copying config
sha256:55a1f4beaf8e2d27982b38e3ecfd458c66753cbfd3a09bcf562877fe60255157
 6.44 KB / 6.44 KB [========================================================]
0s
Writing manifest to image destination
Storing signatures
STEP 2: ENTRYPOINT ["/usr/bin/ls"]
ERRO[0002] HOSTNAME is not supported for OCI image format, hostname 9d3d66a8bfcc will
be ignored. Must use `docker` format
--> 9daeef6d672b0d61990b3ce73a025ae19437d52bee563bbeb45be9527bd6eeb7
STEP 3: FROM 9daeef6d672b0d61990b3ce73a025ae19437d52bee563bbeb45be9527bd6eeb7
STEP 4: RUN touch /var/tmp/test1 && touch /var/tmp/test2
--> 71900ead142213818c31b3a2c5f8e0b65c3d118cc36a93f5ef73ce977e0baac7
STEP 5: FROM 71900ead142213818c31b3a2c5f8e0b65c3d118cc36a93f5ef73ce977e0baac7
STEP 6: RUN touch /var/tmp/foo/bar/test3 || touch /var/tmp/test3
touch: cannot touch '/var/tmp/foo/bar/test3': No such file or directory
--> b8d4bad38f669d1031fff9719daede8998fd0404b3ec03d686ea671b2caf3400
STEP 7: COMMIT myls:1.0

[root@workstation ~]# podman run myls:1.0 /var/tmp
test1
test2
test3
```

Now, try with &&:

```
[root@workstation test2]# vim Dockerfile
FROM rhel7
ENTRYPOINT ["/usr/bin/ls"]
RUN touch /var/tmp/test1 && touch /var/tmp/test2
RUN touch /var/tmp/foo/bar/test3 && touch /var/tmp/test3
```

```
[root@workstation test2]# podman build -t myls:2.0 .
STEP 1: FROM rhel7
STEP 2: ENTRYPOINT ["/usr/bin/ls"]
--> Using cache 9daeef6d672b0d61990b3ce73a025ae19437d52bee563bbeb45be9527bd6eeb7
STEP 3: FROM 9daeef6d672b0d61990b3ce73a025ae19437d52bee563bbeb45be9527bd6eeb7
STEP 4: RUN touch /var/tmp/test1 && touch /var/tmp/test2
--> Using cache 71900ead142213818c31b3a2c5f8e0b65c3d118cc36a93f5ef73ce977e0baac7
STEP 5: FROM 71900ead142213818c31b3a2c5f8e0b65c3d118cc36a93f5ef73ce977e0baac7
STEP 6: RUN touch /var/tmp/foo/bar/test3 && touch /var/tmp/test3
touch: cannot touch '/var/tmp/foo/bar/test3': No such file or directory
error building at step
{Env:[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin container=oci]
Command:run Args:[touch /var/tmp/foo/bar/test3 && touch /var/tmp/test3] Flags:[]
Attrs:map[] Message:RUN touch /var/tmp/foo/bar/test3 && touch /var/tmp/test3
Original:RUN touch /var/tmp/foo/bar/test3 && touch /var/tmp/test3}: error while running
runtime: exit status 1
```

COPY VS ADD

- They both add files to the container filesystem
- COPY only copies local files, ADD can do that plus decompress .tar or files from a URL
- Use COPY when you don't need ADD
- Both the ADD and COPY instructions copy the files, retaining permissions, with root as the owner, even if the USER instruction is specified. Red Hat recommends using a RUN instruction after the copy to change the owner and avoid "permission denied" errors.

```
[root@workstation ~]# mkdir mycat
[root@workstation ~]# cd mycat
[root@workstation mycat]# echo "helloworld" > important
```

```
[root@workstation mycat]# vim Containerfile
FROM ubi8
ENTRYPOINT ["/bin/cat"]
COPY ./important /tmp/
```

```
[root@workstation mycat]# podman build -t ubi8:mycat1 .
[root@workstation myecho]# podman run ubi8:mycat1 /tmp/important
helloworld
```

Now, lets try and use ADD. First we create a tarball

```
[root@workstation mycat]# tar cvf important.tar important
important
[root@workstation mycat]# vim Containerfile
FROM ubi8
ENTRYPOINT ["/bin/cat"]
ADD ./important.tar /tmp/
```

```
[root@workstation mycat]# podman build -t ubi8:mycat1.1 .
[root@workstation mycat]# podman run ubi8:mycat1.1 /tmp/important
helloworld
[root@workstation mycat]# podman run -it --entrypoint /bin/bash ubi8:mycat1.1
[root@d3f45bdf01ea /]# ls -l /tmp
total 8
-rw-r--r--. 1 root root  11 Aug 15 17:21 important
```

Watch out for permissions on these files using ADD and COPY. ADD will assign the owner to be the first user created (ie uid=1000). This might result in an inability to modify files.

```
[root@workstation mycat]# vim Containerfile
FROM ubi8
ENTRYPOINT ["/bin/cat"]
RUN useradd test1
RUN useradd test2
RUN useradd foo
ADD ./important.tar /tmp/
USER foo
```

```
[student@workstation mycat]$ podman build -t ubi8:mycat1.2 .
[student@workstation mycat]$ podman run -it --entrypoint /bin/bash ubi8:mycat1.2
[foo@93b3b0afa711 /]$ whoami
```

```
foo
[foo@93b3b0afa711 /]$ **ls -l /tmp/important**
-rw-rw-r--. 1 test1 test1 10240 Sep 23 10:40 /tmp/important
```

With COPY, the ownership of the copied file(s) will be root:

```
[root@workstation mycat]# vim Containerfile
FROM ubi8
ENTRYPOINT ["/bin/cat"]
RUN useradd test1
RUN useradd test2
RUN useradd foo
COPY important /tmp
USER foo
```

```
[student@workstation mycat]$ podman build -t ubi8:mycat1.3 .
[student@workstation mycat]$ podman run -it --entrypoint /bin/bash ubi8:mycat1.3
[foo@6e6bd10818a0 /]$ ls -l /tmp
total 20
-rw-rw-r--. 1 root root 10240 Sep 23 10:40 important
-rwx------. 1 root root   701 Sep 14 16:20 ks-script-6x37t6sn
-rwx------. 1 root root   291 Sep 14 16:20 ks-script-hlgzt1pz
```

Let’s fix this with another RUN instruction to change owner. Only, be careful where the USER instruction is in comparison to the chown. B/c we won’t be able to chown a file owned by root as the megatron user:

```
[student@workstation mycat]$ vim Containerfile
FROM ubi8
ENTRYPOINT ["/bin/cat"]
RUN useradd test1
RUN useradd test2
RUN useradd foo
ADD important.tar /tmp
RUN chown foo:foo /tmp/important*
USER foo
```

ADDING METADATA: EXPOSE, LABEL, MAINTAINER

```
[student@workstation mycat]$ vim Containerfile
FROM ubi8
LABEL myenv=dev \
        site=dc \
        org=gss
MAINTAINER Andrew Blum <ablum@redhat.com>
EXPOSE 8080
ENTRYPOINT ["/bin/cat"]
RUN useradd test1
RUN useradd test2
RUN useradd foo
ADD important.tar /tmp
RUN chown foo:foo /tmp/important*
USER foo
WORKDIR /var

[student@workstation mycat]$ podman build -t mycat:latest .
[student@workstation mycat]$ podman inspect mycat:latest | less

        "Config": {
        "User": "foo",
        "ExposedPorts": {
                "8080/tcp": {}
        }


        "Labels": {
                "myenv": "dev",
                "name": "ubi8",
                "org": "gss",

        "Author": "Andrew Blum \u003cablum@redhat.com\u003e",
```

EXTRA PRACTICE (if needed)

Let’s build a simple Hello World webserver:

```
[student@workstation ~]$ **mkdir webhello**
[student@workstation ~]$ **cd webhello/**
[student@workstation webhello]$ **vim index.html**
<!DOCTYPE html>
<html>
   <title>Welcome to the web Hello World</title>
   <body>
          <h1>Hello World!</h1>
   </body>
</html>
```

```
[student@workstation webhello]$ **vi Dockerfile**

FROM registry.access.redhat.com/ubi8:latest
MAINTAINER Andrew Blum <ablum@redhat.com>
LABEL stage=dev \
         version=1.0
ENV myport=8080

RUN yum install httpd -y
RUN sed -i "s/Listen 80/Listen ${myport}/" /etc/httpd/conf/httpd.conf
COPY index.html /var/www/html/index.html

EXPOSE ${myport}
ENTRYPOINT ["httpd","-D","FOREGROUND"]
```

```
[student@workstation webhello]$ **podman run -d -p 9090:8080 webhello:1.0**
3d057c001849eaa83be7497b945f04018f268188ccf95df159aabd6ebb3891fa
[student@workstation webhello]$ **curl localhost:9090**
<!DOCTYPE html>
<html>
   <title>Welcome to the web Hello World</title>
   <body>
          <h1>Hello World!</h1>
   </body>
```

```
</html>

[student@workstation webhello]$ podman inspect webhello:1.0 | less


        "ExposedPorts": {
                "8080/tcp": {}

                "stage": "dev",
                "summary": "Provides the latest release of Red Hat Universal Base Image 8.",
                "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/ubi8/images/8.4-206",
                "vcs-ref": "ed5adf70c28eb951940c72f4173fa32c4bca2165",
                "vcs-type": "git",
                "vendor": "Red Hat, Inc.",
                "version": "1.0"


        "Author": "Andrew Blum \u003cablum@redhat.com\u003e",
```

(USE ONLY IF EXTRA EXAMPLE IS NEEDED)
Let's get a simple webserver created:

```
[root@workstation ~]# mkdir myhttpd
[root@workstation ~]# cd myhttpd/
[root@workstation myhttpd]# cp /etc/yum.repos.d/rhel_dvd.repo .
[root@workstation myhttpd]# vim Dockerfile
FROM rhel7
ENTRYPOINT ["httpd","-D","FOREGROUND"]
COPY ./rhel_dvd.repo /etc/yum.repos.d/
RUN yum install httpd -y

[root@workstation myhttpd]# podman build -t myhttpd:1.0 .
[root@workstation myhttpd]# podman run -d myhttpd:1.0
c5bd745f3bb3b30a9a098d1bfd27fe840c678496c565139baf897e951f2889e3
[root@workstation myhttpd]# podman inspect c5 | grep -i ipaddress
        "SecondaryIPAddresses": null,
        "IPAddress": "10.88.0.37",
[root@workstation myhttpd]# curl 10.88.0.37 | head
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3985  100  3985    0     0   961k      0 --:--:-- --:--:-- --:--:-- 1297k
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <head>
            <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
```

Ok, lets modify this container such that a user `megatron` serves up content from his home directory (ie public_html):

```
[root@workstation myhttpd]# echo helloworld > index.html

[root@workstation myhttpd]# vim Dockerfile
FROM rhel7
ENTRYPOINT ["httpd","-D","FOREGROUND"]
COPY ./rhel_dvd.repo /etc/yum.repos.d/
RUN yum install httpd -y
RUN sed -i 's/UserDir disabled/\#UserDir disabled/' /etc/httpd/conf.d/userdir.conf
RUN sed -i 's/\#UserDir public_html/Userdir public_html/' /etc/httpd/conf.d/userdir.conf
RUN useradd megatron
RUN mkdir /home/megatron/public_html
COPY ./index.html /home/megatron/public_html
RUN chown -R megatron /home/megatron
RUN chmod 755 /home/megatron/

[root@workstation myhttpd]# podman build -t myhttpd:2.0 .
[root@workstation myhttpd]# podman run -d myhttpd:2.0
0331755fd6480fc81b6f97b9504f0bde342bb056cdad058773cb582597cd4349
[root@workstation myhttpd]# podman inspect 033 | grep -i ipaddress
            "SecondaryIPAddresses": null,
            "IPAddress": "10.88.0.38",
[root@workstation myhttpd]# curl 10.88.0.38/~megatron/
helloworld
```

Now EXPOSE:

```
[root@workstation myhttpd]# vim Dockerfile
```

```
FROM rhel7
ENTRYPOINT ["httpd","-D","FOREGROUND"]
ENV myport 8080
COPY ./rhel_dvd.repo /etc/yum.repos.d/
RUN yum install httpd -y
RUN sed -i 's/UserDir disabled/\#UserDir disabled/' /etc/httpd/conf.d/userdir.conf
RUN sed -i 's/\#UserDir public_html/Userdir public_html/' /etc/httpd/conf.d/userdir.conf
RUN sed -i "s/Listen 80/Listen ${myport}/" /etc/httpd/conf/httpd.conf
RUN useradd megatron
RUN mkdir /home/megatron/public_html
COPY ./index.html /home/megatron/public_html
RUN chown -R megatron /home/megatron
RUN chmod 755 /home/megatron/
EXPOSE ${myport}
```

```
[root@workstation myhttpd]# podman build -t myhttpd:3.0 .
```

```
[root@workstation myhttpd]# podman inspect myhttpd:3.0 | less
```

(show the exposed port is now in the metadata)

```
[root@workstation myhttpd]# podman run -d myhttpd:3.0
6ea9a0759467f3d7618fb420ebdca8dda24ecf4882822e5f92361c1473aa4a7e
[root@workstation myhttpd]# podman inspect 6ea | grep -i ipaddr
            "SecondaryIPAddresses": null,
            "IPAddress": "10.88.0.39",
[root@workstation myhttpd]# curl 10.88.0.39:8080
```

Now add in author and a label:

```
[root@workstation myhttpd]# vim Dockerfile
```

```
FROM rhel7
MAINTAINER Andrew Blum <ablum@redhat.com>
LABEL myttpd dev
ENTRYPOINT ["httpd","-D","FOREGROUND"]
```

```
ENV myport 8080
COPY ./rhel_dvd.repo /etc/yum.repos.d/
RUN yum install httpd -y
RUN sed -i 's/UserDir disabled/\#UserDir disabled/' /etc/httpd/conf.d/userdir.conf
RUN sed -i 's/\#UserDir public_html/Userdir public_html/' /etc/httpd/conf.d/userdir.conf
RUN sed -i "s/Listen 80/Listen ${myport}/" /etc/httpd/conf/httpd.conf
RUN useradd megatron
RUN mkdir /home/megatron/public_html
COPY ./index.html /home/megatron/public_html
RUN chown -R megatron /home/megatron
RUN chmod 755 /home/megatron/
EXPOSE ${myport}
```

```
[root@workstation myhttpd]# podman build -t myhttpd:4.0 .
[root@workstation myhttpd]# podman inspect myhttpd:4.0 | less
```

Finally, lets clean it up and add in comments:

```
# This file provides a webserver that serves static content from a non-root user's public_html
directory
FROM rhel7
MAINTAINER Andrew Blum <ablum@redhat.com>
LABEL myttpd dev
ENTRYPOINT ["httpd","-D","FOREGROUND"]
ENV myport 8080
EXPOSE ${myport}
# use classroom yum repo and install httpd
COPY ./rhel_dvd.repo /etc/yum.repos.d/
RUN yum install httpd -y
# modify default httpd configuration to allow for local user's public_html and to listen on a
different port
RUN sed -i 's/UserDir disabled/\#UserDir disabled/' /etc/httpd/conf.d/userdir.conf && \
        sed -i 's/\#UserDir public_html/Userdir public_html/' /etc/httpd/conf.d/userdir.conf && \
        sed -i "s/Listen 80/Listen ${myport}/" /etc/httpd/conf/httpd.conf
# add a user and set permissions to allow apache user to serve up public_html
RUN useradd megatron && \
        mkdir /home/megatron/public_html && \
        chown -R megatron /home/megatron && \
        chmod 755 /home/megatron/
# copy webcontent to public_html directory
COPY ./index.html /home/megatron/public_html
```

```
[root@workstation myhttpd]# **podman build -t myhttpd:latest .**
```

```
[student@workstation ~]$ mkdir myserver
student@workstation ~]$ cd myserver/
[student@workstation myserver]$ echo helloworld > index.html
[student@workstation myserver]$
[student@workstation myserver]$ vim Dockerfile
# This file provides a webserver that serves static content from a non-root user's public_html
directory
FROM ubi8
MAINTAINER Andrew Blum <ablum@redhat.com>
LABEL myttpd dev
ENTRYPOINT ["httpd","-D","FOREGROUND"]
ENV myport 8080
EXPOSE ${myport}
RUN yum install httpd -y
# modify default httpd configuration to allow for local user's public_html and to listen on a
different port
RUN sed -i 's/UserDir disabled/\#UserDir disabled/' /etc/httpd/conf.d/userdir.conf && \
        sed -i 's/\#UserDir public_html/Userdir public_html/' /etc/httpd/conf.d/userdir.conf && \
        sed -i "s/Listen 80/Listen ${myport}/" /etc/httpd/conf/httpd.conf
# add a user and set permissions to allow apache user to serve up public_html
RUN useradd megatron && \
        mkdir /home/megatron/public_html && \
        chown -R megatron /home/megatron && \
        chmod 755 /home/megatron/
# copy webcontent to public_html directory
COPY ./index.html /home/megatron/public_html

[student@workstation myserver]$ podman build -t myhttpd:latest .
[student@workstation myserver]$ podman run -d -p 8083:8080 myhttpd
[student@workstation myserver]$ curl localhost:8083/~megatron/
helloworld
```

```
[student@workstation public_html]$ mkdir ~/public_html
[student@workstation public_html]$ sudo chcon -R -t container_file_t ~/public_html/
[student@workstation public_html]$ podman run -d -p 8084:8080 -v
~/public_html:/home/megatron/public_html myhttpd
```

```
[student@workstation public_html]$  curl localhost:8084/~megatron/
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
 <head>
  <title>Index of /~megatron</title>
 </head>
 <body>
<h1>Index of /~megatron</h1>
  <table>
   <tr><th valign="top"><img src="/icons/blank.gif" alt="[ICO]"></th><th><a
href="?C=N;O=D">Name</a></th><th><a href="?C=M;O=A">Last modified</a></th><th><a
href="?C=S;O=A">Size</a></th><th><a href="?C=D;O=A">Description</a></th></tr>
   <tr><th colspan="5"><hr></th></tr>
<tr><td valign="top"><img src="/icons/back.gif" alt="[PARENTDIR]"></td><td><a
href="/">Parent Directory</a>            </td><td>&nbsp;</td><td align="right">  -
</td><td>&nbsp;</td></tr>
   <tr><th colspan="5"><hr></th></tr>
</table>
</body></html>

[student@workstation public_html]$ echo hello_from_host_volume >
~/public_html/index.html
[student@workstation public_html]$  curl localhost:8083/~megatron/
hello_from_host_volume
```

How to install software from Red Hat repositories?

https://access.redhat.com/solutions/1443553
enable repos in container: https://access.redhat.com/solutions/1443553

""

Entitlement information from the host is injected into the container when the first yum command in the container is run.

Thus, containers are not entitled, but they can access any repository the host can access based on those entitlements, even if the repositories are disabled on the host

NOTE: Until the first yum command is run, /etc/yum.repos.d/redhat.repo contains no repositories, so yum-config-manager will not enable/disable anything.

""

```
RUN yum repolist --disablerepo=* && \
        yum-config-manager --disable \* > /dev/null && \
        yum-config-manager --enable rhel-7-server-rpms > /dev/null
```

Otherwise, ubi.repo are available if you are using ubi container images:

```
[root@8213cb08ef88 /]# yum repolist
Updating Subscription Management repositories.
Unable to read consumer identity
This system is not registered to Red Hat Subscription Management. You can use
subscription-manager to register.
Red Hat Universal Base Image 8 (RPMs) - BaseOS
        120 kB/s | 759 kB     00:06
Red Hat Universal Base Image 8 (RPMs) - AppStream
        1.5 MB/s | 3.1 MB     00:02
Red Hat Universal Base Image 8 (RPMs) - CodeReady Builder
        5.9 kB/s | 9.1 kB     00:01
repo id                                   repo name
        status
ubi-8-appstream                                   Red Hat Universal Base Image 8 (RPMs) -
AppStream                                 797
ubi-8-baseos                                      Red Hat Universal Base Image 8 (RPMs) -
BaseOS                                    663
ubi-8-codeready-builder                           Red Hat Universal Base Image 8 (RPMs) -
CodeReady Builder                                 12
[root@8213cb08ef88 /]# yum search openjdk
Updating Subscription Management repositories.
Unable to read consumer identity
This system is not registered to Red Hat Subscription Management. You can use
subscription-manager to register.
Last metadata expiration check: 0:00:12 ago on Thu Dec 12 21:55:15 2019.
```

```
================================================================================
Name & Summary Matched: openjdk
================================================================================
java-11-openjdk.x86_64 : OpenJDK Runtime Environment 11
java-1.8.0-openjdk.x86_64 : OpenJDK Runtime Environment 8
java-11-openjdk-devel.x86_64 : OpenJDK Development Environment 11
java-1.8.0-openjdk-devel.x86_64 : OpenJDK Development Environment 8
java-11-openjdk-headless.x86_64 : OpenJDK Headless Runtime Environment 11
java-1.8.0-openjdk-headless.x86_64 : OpenJDK Headless Runtime Environment 8
```

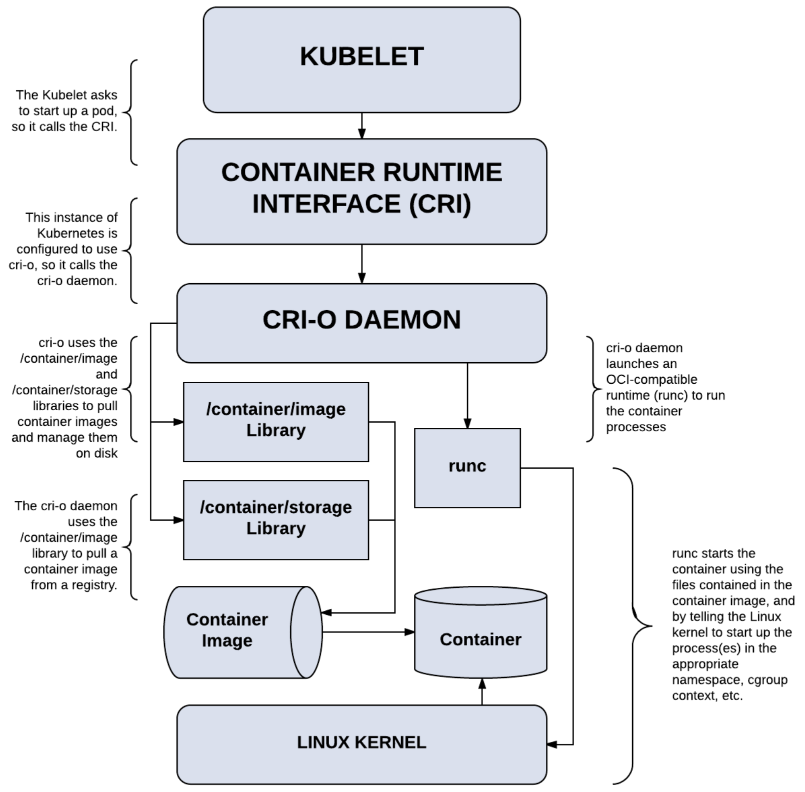
```
[root@8213cb08ef88 /]# ls /etc/yum.repos.d/
redhat.repo  ubi.repo
```

CHAPTER 6: DEPLOYING CONTAINERIZED APPLICATIONS ON OPENSHIFT

DESCRIBING KUBERNETES AND OPENSHIFT ARCHITECTURE

What is crio ?

Cri-o: container runtime used in k8s/ocp. It's what k8s depends on to run containers. see https://www.redhat.com/en/blog/introducing-cri-o-10


KUBELET
The Kubelet asks to start up a pod, so it calls the CRI.
CONTAINER RUNTIME INTERFACE (CRI)
This instance of Kubernetes is configured to use cri-o, so it calls the cri-o daemon.
CRI-O DAEMON
cri-o uses the /container/image and /container/storage libraries to pull container images and manage them on disk
cri-o daemon launches an OCI-compatible runtime (runc) to run the container processes
/container/image Library
runc
/container/storage Library
The cri-o daemon uses the /container/image library to pull a container image from a registry.
runc starts the container using the files contained in the container image, and by telling the Linux kernel to start up the process(es) in the appropriate namespace, cgroup context, etc.
Container Image
Container
LINUX KERNEL

(Using DO280 environment)

Classroom Identifier   vwermreqvvomqez200819
Cluster ID   bb8bca34-2206-4483-8e6f-2b14a70b1547
Cluster Username   kubeadmin
Cluster Password   IWX4R-Q8qMj-yTImo-A4Zms
Cluster API URL   api.ocp-vwermreqvvomqez200819.do280.rht-na.nextcle.com:6443
State   installing
Classroom Status   OpenShift Installation Completed

```
[ablum@badger ocp4]$ oc login -u kubeadmin -p IWX4R-Q8qMj-yTImo-A4Zms
api.ocp-vwermreqvvomqez200819.do280.rht-na.nextcle.com:6443
[ablum@badger ocp4]$ oc debug node/ip-10-0-136-0.ec2.internal
Starting pod/ip-10-0-136-0ec2internal-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.0.136.0
If you don't see a command prompt, try pressing enter.
sh-4.2#
sh-4.2# chroot /host /bin/bash
```

```
[root@master0 /]# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet
  Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset: enabled)


[root@master0 /]# systemctl status crio
● crio.service - Open Container Initiative Daemon
  Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; vendor preset: disabled)


[root@master0 /]# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
[root@master0 /]#
[root@master0 /]#
[root@master0 /]# podman ps -a
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES

[root@master0 /]# crictl -h
NAME:
  crictl - client for CRI

[root@master0 /]# crictl ps
```

https://blog.openshift.com/crictl-vs-podman/

NOTE1: crictl is not a fully featured as podman. It can't build nor restart containers. They both use runc, but differently so they won't "see" the same running containers. Crictl sees what k8s sees.

NOTE2: podman and cri-o do share the same image storage libraries so a podman rmi will remove the image from local storage on that node from cri-o (will have to re-pull the image from registry)

What makes a control node ?

- Kube-apiserver
- Kube-controller-manager - The Controller Manager Server watches etcd for changes to objects such as replication, namespace, and serviceaccount controller objects, and then uses the API to enforce the specified state. Several such processes create a cluster with one active leader at a time.
- Kube-scheduler

- Etcd - etcd stores the persistent master state while other components watch etcd for changes to bring themselves into the specified state.

```
[root@cluster-master-0 ~]# crictl ps | grep etcd
a9e431dc52745
643c21638c1c966fe18ca1cc8547dd401df70e85d83ca6de76b9a7957703b993
        About an hour ago   Running         etcd-member
```

```
[root@cluster-master-0 ~]# crictl ps | grep kube
```

https://kubernetes.io/docs/concepts/overview/components/

Podman and skopeo are still available, point out how to use the authfile:

```
[root@master0 /]# skopeo inspect docker://registry.redhat.io/rhel7
FATA[0000] unable to retrieve auth token: invalid username/password
[root@master0 /]# skopeo inspect --authfile=/var/lib/kubelet/config.json
docker://registry.redhat.io/rhel7
```

(useful for recovering from control plane certificate expiry: https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/html/backup_and_restore/disaster-recovery#dr-scenario-3-recovering-expired-certs_dr-recovering-expired-certs)

CREATING KUBERNETES RESOURCES

GUIDED PRACTICE: Deploying a Database Server on OpenShift

```
[student@workstation ~]$ lab openshift-resources start
```

Setting up workstation for the Guided Exercise: Deploying a Database Server on OpenShift

```
Verifying the OpenShift cluster is running:
· Log in on OpenShift.......................................  SUCCESS
· Check the internal registry is up and running...............  SUCCESS
· Ensuring the 'rhn-support-ablum-mysql-openshift' project is absent  SUCCESS
```

What is oc ?

First, Let’s understand the “oc” or openshift client cli better. https://github.com/openshift/oc

Navigate to https://mirror.openshift.com/pub/openshift-v4/ then clients > oc. Click on the appropriate version (should try to match major versions to cluster version).

```
[student@workstation ~]$ mkdir work
[student@workstation ~]$ cd work/
[student@workstation work]$ wget
https://mirror.openshift.com/pub/openshift-v4/clients/oc/4.6/linux/oc.tar.gz
[student@workstation work]$ tar xvzf oc.tar.gz
oc
[student@workstation work]$ file oc
oc: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs),
for GNU/Linux 2.6.32, BuildID[sha1]=b175d93b9990cdeb07fc6a7c603dd3d8ded14c6f, stripped

[student@workstation work]$ ./oc version
Client Version: 4.6.44
Kubernetes Version: v1.19.0+8d12420

[student@workstation work]$ oc completion bash > oc_completion
[student@workstation work]$ head oc_completion

# bash completion for oc                              -*- shell-script -*-

__oc_debug()
{
        if [[ -n ${BASH_COMP_DEBUG_FILE} ]]; then
        echo "$*" >> "${BASH_COMP_DEBUG_FILE}"
```

If you wanted to use this version you could replace the one that comes with our training environment. Let's not do that. Here is what we get with our environment:

```
[student@workstation work]$ cd ..
[student@workstation ~]$ rm -rf work/
[student@workstation ~]$

[student@workstation work]$ which oc
/usr/local/bin/oc
[student@workstation work]$ oc version
Client Version: 4.5.4
Kubernetes Version: v1.18.3+002a51f

[student@workstation work]$ head /etc/bash_completion.d/oc

# bash completion for oc                          -*- shell-script -*-

__oc_debug()
{
        if [[ -n ${BASH_COMP_DEBUG_FILE} ]]; then
        echo "$*" >> "${BASH_COMP_DEBUG_FILE}"
```

There are alot of subcommands used with oc:

```
[student@workstation ~]$ oc  [TAB][TAB]
adm             cancel-build    delete          extract         logs            policy
rollback        set
annotate        cluster-info    describe        get             new-app         port-forward    rollout
        start-build
api-resources   completion      diff            idle            new-build       process         rsh
        status
api-versions    config          edit            image           new-project     project         rsync
        tag
apply           convert         ex              import-image    observe         projects        run
        version
attach          cp              exec            label           options         proxy           scale
        wait
auth            create          explain         login           patch           registry        secrets
        whoami
autoscale       debug           expose          logout          plugin          replace
serviceaccounts
```

Getting help on any of them:

```
[student@workstation ~]$ **oc logs --help**
Print the logs for a resource
```

Why not man pages ? We didn’t install oc via rpm - many customers don’t use RHEL as their base laptop or workstation. The oc client is available for Windows and MacOS as well.

Let’s login into the shared OCP cluster provisioned by our learning environment:

```
[student@workstation ~]$ cat /usr/local/etc/ocp4.config
[student@workstation ~]$ source /usr/local/etc/ocp4.config

[student@workstation ~]$ oc login --help

  # Log in to the given server with the given credentials (will not prompt interactively)
  oc login localhost:8443 --username=myuser --password=mypass

[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} -p
${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
```

For the future, I’ll just the alias we setup in bashrc:

```
[student@workstation ~]$ alias dl
alias dl='oc login -u ${RHT_OCP4_DEV_USER} -p ${RHT_OCP4_DEV_PASSWORD}
${RHT_OCP4_MASTER_API}'
[student@workstation ~]$ dl
Login successful.

You don't have any projects. You can try to create a new project, by running

        oc new-project <projectname>
```

How to create my first openshift project ?

Well, we don’t have any projects...ok, let’s change that:

[student@workstation ~]$ **oc new-project ${RHT_OCP4_DEV_USER}-mysql-openshift**
[student@workstation ~]$ **oc new-app -h**
Point out::
  # Use a MySQL image in a private registry to create an app and override application artifacts' names
  oc new-app --docker-image=myregistry.com/mycompany/mysql --name=private

  -e, --env=[]: Specify a key-value pair for an environment variable to set into each container.

--as-deployment-config

Starting in OCP4.5, oc new-app will create k8s deployments (by default) instead of deploymentconfigs.

In our new project:
[student@workstation ~]$ **oc project**
Using project "gwohys-mysql-openshift" on server "https://api.na46.prod.nextcle.com:6443".

How to create my first application in openshift?

Let’s start with a simple example: https://hub.docker.com/r/openshift/hello-openshift/

[student@workstation ~]$ **skopeo inspect docker://docker.io/openshift/hello-openshift**

The code is available here:
https://github.com/openshift/origin/blob/master/examples/hello-openshift/hello_openshift.go

Deploying using the one from docker.io might result in the following errors:

ERRO[0008] error searching registry "docker.io": couldn't search registry "docker.io": error pinging docker registry index.docker.io: Get https://index.docker.io/v2/: dial tcp: lookup index.docker.io on 172.25.250.254:53: server misbehaving

Or

```
W0119 08:12:47.671222   13832 dockerimagelookup.go:237] container image registry lookup
failed: docker.io/openshift/hello-openshift:latest: toomanyrequests: You have reached your pull
rate limit. You may increase the limit by authenticating and upgrading:
https://www.docker.com/increase-rate-limit
error: unable to locate any local docker images with name
"docker.io/openshift/hello-openshift:latest"
```

Then, use personal one cloned from docker.io: to quay.io:

```
[student@workstation ~]$ oc new-app --image=quay.io/ajblum/hello-openshift:latest

[student@workstation ~]$ oc status
In project ablum-mytest on server https://api.na46.prod.nextcle.com:6443

svc/hello-openshift - 172.30.182.91 ports 8080, 8888
  deployment/hello-openshift deploys istag/hello-openshift:latest
        deployment #2 running for 27 seconds - 1 pod
        deployment #1 deployed 28 seconds ago


1 info identified, use 'oc status --suggest' to see details.
[student@workstation ~]$ oc get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/hello-openshift-7b74767ff6-28hf8   1/1   Running   0          39s

NAME                    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)             AGE
service/hello-openshift   ClusterIP   172.30.182.91   <none>        8080/TCP,8888/TCP   40s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-openshift   1/1        1          1         40s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-openshift-7b74767ff6   1         1         1         39s
replicaset.apps/hello-openshift-9c8f5d9f      0         0         0         40s

NAME                                        IMAGE REPOSITORY
        TAGS   UPDATED
imagestream.image.openshift.io/hello-openshift
default-route-openshift-image-registry.apps.na46.prod.nextcle.com/ablum-mytest/hello-openshift
latest   39 seconds ago
```

Looking at any one of these resources:

```
[student@workstation ~]$ oc describe pod/hello-openshift-57749cf8d4-6cx4c
```

```
Labels:         deployment=hello-openshift
                pod-template-hash=57749cf8d4



containers:
  hello-openshift:
        Container ID:
cri-o://6acc97c88bb6062d85f55296da532b44b4ab881fb1a2a1359bd636d56bed3680
        Image:
docker.io/openshift/hello-openshift@sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3
e7c9f556c1ba5baf47e2e
        Image ID:
docker.io/openshift/hello-openshift@sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3
e7c9f556c1ba5baf47e2e
        Ports:          8080/TCP, 8888/TCP
        Host Ports:     0/TCP, 0/TCP
        State:          Running
        Started:        Thu, 16 Sep 2021 15:24:26 -0400
        Ready:          True



Events:
  Type  Reason          Age   From              Message
  ----  ------          ----  ----          -------
  Normal  Scheduled           29s   default-scheduler  Successfully assigned
zrsqwh-mysql-openshift/hello-openshift-57749cf8d4-6cx4c to na46-5m9nf-worker-0-fbmq4
  Normal  AddedInterface  27s   multus          Add eth0 [10.128.4.218/23]
  Normal  Pulled          27s   kubelet         Container image
"docker.io/openshift/hello-openshift@sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3
e7c9f556c1ba5baf47e2e" already present on machine
  Normal  Created         26s   kubelet         Created container hello-openshift
  Normal  Started         26s   kubelet         Started container hello-openshift
```

To see a yaml representation (instead of this “pretty” format):

```
[student@workstation ~]$ oc get pods hello-openshift-57749cf8d4-6cx4c -o yaml
```

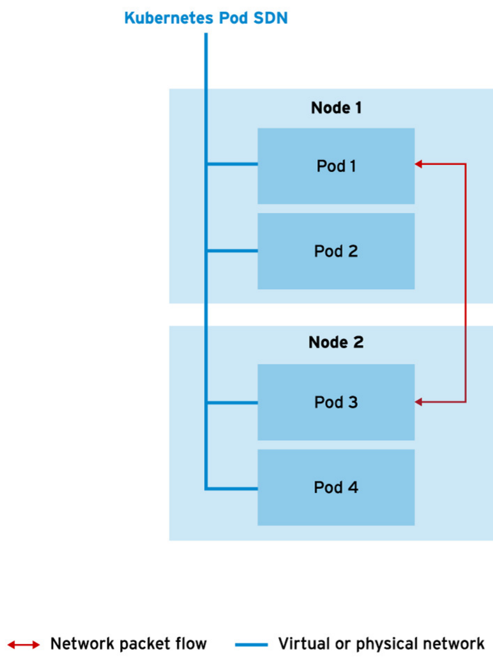
WHOA!!! That's a lot of stuff….I don't understand any of that !

What are PODS?

```
[student@workstation ~]$ oc explain pods
KIND:  Pod
VERSION:  v1

DESCRIPTION:
        Pod is a collection of containers that can run on a host. This resource is
        created by clients and scheduled onto hosts.
```

```
[student@workstation ~]$ oc explain pods.spec
```

All the containers in a pod share the same networking namespace. Different pods are connected to each other by the k8s pod sdn, a networking overlay implemented by opensvswitch (ovs) running on each node:



What are those managedFields ?

https://access.redhat.com/solutions/5332041

```
  managedFields:
  - apiVersion: v1
          fieldsType: FieldsV1
          fieldsV1:
          f:metadata:
          f:annotations:
          .: {}
          f:openshift.io/generated-by: {}
          f:generateName: {}
          f:labels:
          .: {}
          f:deployment: {}
          f:pod-template-hash: {}
          f:ownerReferences:
          .: {}
          k:{"uid":"b06524c9-cae7-41c4-88bc-9113c9ff21ed"}:
          .: {}
          f:apiVersion: {}
          f:blockOwnerDeletion: {}
          f:controller: {}
          f:kind: {}
          f:name: {}
          f:uid: {}
```

Interesting discussion on this issue upstream:
https://github.com/kubernetes/kubernetes/issues/90066

Where do we specify the pod that we want to see deployed ? There are several ways to do that, but one that's used here by the `oc new-app` command is a deployment:

```
[student@workstation ~]$ oc get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
hello-openshift   1/1    1           1           7m52s
```

```
[student@workstation ~]$ oc explain deployment
```

```
[student@workstation ~]$ oc explain deployment.spec.template
KIND:  Deployment
VERSION:  apps/v1

RESOURCE: template <Object>

DESCRIPTION:
        Template describes the pods that will be created.

        PodTemplateSpec describes the data a pod should have when created from a
        template

[student@workstation ~]$ oc get deployment hello-openshift -o yaml

  template:
        metadata:
        annotations:
        openshift.io/generated-by: OpenShiftNewApp
        creationTimestamp: null
        labels:
        deployment: hello-openshift

        spec:
        containers:
        - image:
docker.io/openshift/hello-openshift@sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3
e7c9f556c1ba5baf47e2e
        imagePullPolicy: IfNotPresent
        name: hello-openshift
        ports:
        - containerPort: 8080
        protocol: TCP
        - containerPort: 8888
        protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
```

Notice the pod running here inherited the label defined in the template spec of the deployment:

```
[student@workstation ~]$ oc get pods --show-labels
NAME                          READY   STATUS     RESTARTS   AGE   LABELS
```

```
hello-openshift-57749cf8d4-6cx4c   1/1       Running   0              11m
deployment=hello-openshift,pod-template-hash=57749cf8d4
```

You can use this label to select matching resources:

```
[student@workstation ~]$ oc get pods -l deployment=hello-openshift
NAME                               READY   STATUS     RESTARTS   AGE
hello-openshift-57749cf8d4-6cx4c   1/1       Running   0              13m
```

What are DEPLOYMENT CONFIGURATION vs DEPLOYMENT?

https://docs.openshift.com/container-platform/4.5/applications/deployments/what-deployments-are.html#what-deployments-are

OpenShift Container Platform deployments from DeploymentConfigs also provide the ability to transition from an existing deployment of an image to a new one and also define hooks to be run before or after creating the ReplicationController.

One important difference between Deployments and DeploymentConfigs is the properties of the CAP theorem that each design has chosen for the rollout process. DeploymentConfigs prefer consistency, whereas Deployments take availability over consistency.

For DeploymentConfigs, if a node running a deployer Pod goes down, it will not get replaced. The process waits until the node comes back online or is manually deleted. Manually deleting the node also deletes the corresponding Pod. This means that you can not delete the Pod to unstick the rollout, as the kubelet is responsible for deleting the associated Pod.

However, Deployments rollouts are driven from a controller manager. The controller manager runs in high availability mode on masters and uses leader election algorithms to value availability over consistency. During a failure it is possible for other masters to act on the same Deployment at the same time, but this issue will be reconciled shortly after the failure occurs

How do you delete this application ? Let's try:

```
[student@workstation ~]$ oc delete pod/hello-openshift-7b74767ff6-28hf8
```

```
(short pause)
[student@workstation ~]$ **oc get pods**
NAME                                   READY   STATUS      RESTARTS   AGE
hello-openshift-7b74767ff6-529h6   1/1        Running   0               11s
```

It’s back ! This is due to the replicaset (controller) insuring that our desired state is met:

```
[student@workstation ~]$ oc get replicaset
NAME                        DESIRED   CURRENT   READY   AGE
hello-openshift-57749cf8d4   1          1          1         14m
hello-openshift-5df9dfbb6c   0          0          0         14m

[student@workstation ~]$ oc describe replicaset hello-openshift-57749cf8d4

Events:
  Type  Reason           Age   From                 Message
  ----     ------              ----  ----                    -------
  Normal  SuccessfulCreate  14m   replicaset-controller  Created pod:
hello-openshift-57749cf8d4-6cx4c
  Normal  SuccessfulCreate  75s   replicaset-controller  Created pod:
hello-openshift-57749cf8d4-x48z
```

What are REPLICATION CONTROLLERS vs REPLICASETS?

DeploymentConfigs involve one or more *ReplicationControllers*, which contain a point-in-time record of the state of a DeploymentConfig as a Pod template.

Similarly, Deployments involve one or more *ReplicaSets*.

The difference between a ReplicaSet and a ReplicationController is that a ReplicaSet supports set-based selector requirements whereas a replication controller only supports equality-based selector requirements. *Set-based* label requirements allow filtering keys according to a set of values.

Consider this set-based selector,
environment in (production, qa)
Selects all resources with key equal to environment and value equal to production or qa.

Consider this equality-based selector:
Environment = production
selects all resources with key equal to environment and value equal to production

So, how do we delete this application ?

It was the deployment that defined what we wanted, this is what we must delete:

```
[student@workstation ~]$ oc delete deployment.apps/hello-openshift
deployment.apps "hello-openshift" deleted
```

There are some other resources left as well (route, service, imagestream):

```
[student@workstation ~]$ oc get all --show-labels
NAME                    TYPE          CLUSTER-IP  EXTERNAL-IP   PORT(S)              AGE
LABELS
service/hello-openshift   ClusterIP   172.30.182.91   <none>        8080/TCP,8888/TCP   21m
app.kubernetes.io/component=hello-openshift,app.kubernetes.io/instance=hello-openshift,app=
hello-openshift
```

```
[student@workstation ~]$ oc delete all -l app=hello-openshift
service "hello-openshift" deleted
imagestream.image.openshift.io "hello-openshift" deleted
route.route.openshift.io "hello-openshift" deleted
```

Let's try to deploy again using --as-deployment-config

```
[student@workstation ~]$  oc new-app --image=quay.io/ajblum/hello-openshift:latest
--as-deployment-config
```

```
[student@workstation ~]$ oc get all
NAME                    READY   STATUS              RESTARTS   AGE
pod/hello-openshift-1-deploy   0/1     Completed              0        27s
pod/hello-openshift-1-gn6kd   1/1     Running        0        24s
```

This time, the deployment is rolled out using a "deploy" pod running our namespace. This represents a fundamental difference between a deployment and a deploymentconfig.

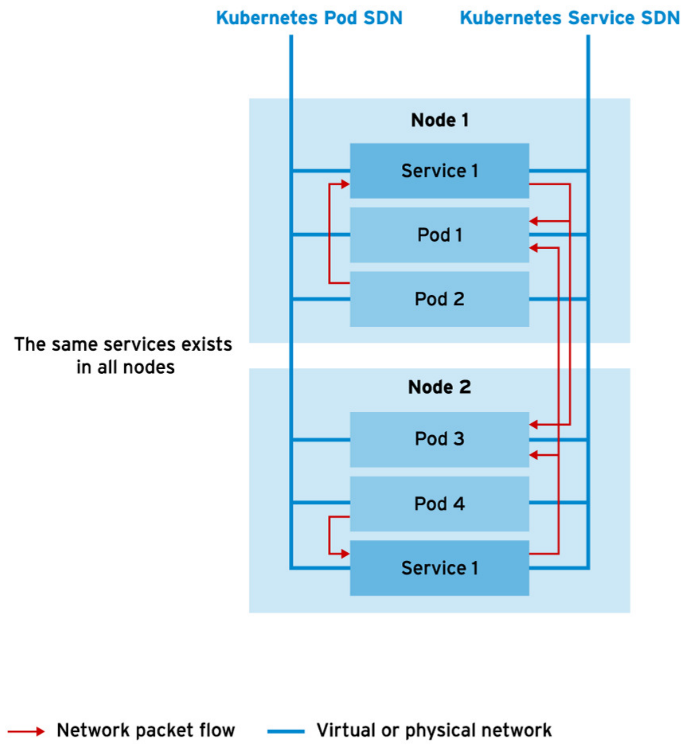
See
https://docs.openshift.com/container-platform/4.7/applications/deployments/what-deployments-are.html

What are SERVICES ?

Every pod is assigned a unique IP addressed by the pod SDN. When a new pod is created, it will use a different IP (depending on which node it is assigned to):

```
[student@workstation ~]$ **oc get pods -o wide**
NAME                  READY   STATUS      RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
hello-openshift-1-7sln2        1/1      Running        0        34s   10.128.5.87
na46-5m9nf-worker-0-fbmq4   <none>         <none>
hello-openshift-1-deploy   0/1          Completed   0          37s   10.128.5.86
na46-5m9nf-worker-0-fbmq4   <none>         <none>
```

What is the best way for applications in different pods to communicate ? They could use the pod IP, but that would be tough for the applications to keep track of. A service makes this easier, but will require a whole different SDN:



```
[student@workstation ~]$ **oc get services**
NAME            TYPE            CLUSTER-IP           EXTERNAL-IP   PORT(S)               AGE
hello-openshift   ClusterIP   172.30.106.151   <none>        8080/TCP,8888/TCP   37m
```

```
[student@workstation ~]$
[student@workstation ~]$
[student@workstation ~]$ oc describe service hello-openshift
Name:              hello-openshift
Namespace:         zrsqwh-mysql-openshift
Labels:            app=hello-openshift
                   app.kubernetes.io/component=hello-openshift
                   app.kubernetes.io/instance=hello-openshift
Annotations:       openshift.io/generated-by: OpenShiftNewApp
Selector:          deploymentconfig=hello-openshift


[student@workstation ~]$ oc get endpoints
NAME              ENDPOINTS                             AGE
hello-openshift   10.128.5.87:8080,10.128.5.87:8888   40m

[student@workstation ~]$ oc get pods -l deploymentconfig=hello-openshift
NAME                          READY   STATUS      RESTARTS   AGE
hello-openshift-1-7sln2   1/1   Running   0              44m
[student@workstation ~]$ oc get pods -l deploymentconfig=hello-openshift -o wide
NAME                          READY   STATUS      RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
hello-openshift-1-7sln2   1/1   Running   0              44m   10.128.5.87
na46-5m9nf-worker-0-fbmq4   <none>        <none>
```

If we delete the pod, watch what happens...The endpoint will change to use the new pod's IP, but the service IP won't change:

```
[student@workstation ~]$ oc get endpoints
NAME              ENDPOINTS                             AGE
hello-openshift   10.128.5.115:8080,10.128.5.115:8888   46m
[student@workstation ~]$ oc get pods -l deploymentconfig=hello-openshift -o wide
NAME                          READY   STATUS      RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
hello-openshift-1-dhsvj   1/1   Running   0              56s   10.128.5.115
na46-5m9nf-worker-0-fbmq4   <none>        <none>
[student@workstation ~]$ oc get service
NAME              TYPE          CLUSTER-IP          EXTERNAL-IP   PORT(S)              AGE
hello-openshift   ClusterIP   172.30.106.151   <none>        8080/TCP,8888/TCP   46m
```

What is an IMAGESTREAM ?

```
[student@workstation ~]$ oc explain is
KIND:  ImageStream
VERSION:  image.openshift.io/v1

DESCRIPTION:
        An ImageStream stores a mapping of tags to images, metadata overrides that
        are applied when images are tagged in a stream, and an optional reference
        to a container image repository on a registry. Users typically update the
        spec.tags field to point to external images which are imported from
        container registries using credentials in your namespace with the pull
        secret type, or to existing image stream tags and images which are
        immediately accessible for tagging or pulling. The history of images
        applied to a tag is visible in the status.tags field and any user who can
        view an image stream is allowed to tag that image into their own image
        streams. Access to pull images from the integrated registry is granted by
        having the "get imagestreams/layers" permission on a given image stream.
        Users may remove a tag by deleting the imagestreamtag resource, which
        causes both spec and status for that tag to be removed. Image stream
        history is retained until an administrator runs the prune operation, which
        removes references that are no longer in use. To preserve a historical
        image, ensure there is a tag in spec pointing to that image by its digest.
```

```
[student@workstation ~]$ oc explain istag
KIND:  ImageStreamTag
VERSION:  image.openshift.io/v1

DESCRIPTION:
        ImageStreamTag represents an Image that is retrieved by tag name from an
        ImageStream. Use this resource to interact with the tags and images in an
        image stream by tag, or to see the image details for a particular tag. The
        image associated with this resource is the most recently successfully
        tagged, imported, or pushed image (as described in the image stream
        status.tags.items list for this tag). If an import is in progress or has
        failed the previous image will be shown. Deleting an image stream tag
        clears both the status and spec fields of an image stream. If no image can
        be retrieved for a given tag, a not found error will be returned.
```

```
[student@workstation ~]$ oc get is
NAME            IMAGE REPOSITORY
TAGS  UPDATED
hello-openshift
default-route-openshift-image-registry.apps.na46.prod.nextcle.com/zrsqwh-mysql-openshift/hell
o-openshift   latest   47 minutes ago
[student@workstation ~]$ oc describe is hello-openshift
Name:                   hello-openshift
Namespace:      zrsqwh-mysql-openshift
Created:        47 minutes ago
Labels:                 app=hello-openshift
                app.kubernetes.io/component=hello-openshift
                app.kubernetes.io/instance=hello-openshift
Annotations:    openshift.io/generated-by=OpenShiftNewApp
                openshift.io/image.dockerRepositoryCheck=2021-09-16T19:40:15Z
Image Repository:
default-route-openshift-image-registry.apps.na46.prod.nextcle.com/zrsqwh-mysql-openshift/hell
o-openshift
Image Lookup:           local=false
Unique Images:          1
Tags:           1

latest
  tagged from docker.io/openshift/hello-openshift

  *
docker.io/openshift/hello-openshift@sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3
e7c9f556c1ba5baf47e2e
        47 minutes ago
```

These abstractions are useful for handling changes in image registries. They can be used to trigger builds and deployments when images are pushed to registries.

NOTE: These do not contain the actual image data (ie the lower dirs). They are pointers.

https://docs.openshift.com/container-platform/4.8/openshift_images/images-understand.html

What are PERSISTENT VOLUMES and PERSISTENT VOLUME CLAIMS ?

```
[student@workstation ~]$ oc explain pv
KIND:  PersistentVolume
VERSION:  v1

DESCRIPTION:
        PersistentVolume (PV) is a storage resource provisioned by an
        administrator. It is analogous to a node. More info:
        https://kubernetes.io/docs/concepts/storage/persistent-volumes

[student@workstation ~]$ oc explain pvc
KIND:  PersistentVolumeClaim
VERSION:  v1

DESCRIPTION:
        PersistentVolumeClaim is a user's request for and claim to a persistent
        volume
```

When making the request for storage for your application (ie the **pvc**), you will need to define characteristics that the persistent volume controller (a control loop that runs within the kube-controller-manager) uses to match the available persistent volumes (the **pv**).

Once a match is made it is said to be “bound” and can be used (ie mounted) within a container in a pod.

NOTE: Developers do not permissions to manipulate or view persistent volumes (pv) only the claims (pvc)

So, how do we test our hello-world application ?

One way to test this application is with `oc port-forward`

```
[student@workstation ~]$ nohup oc port-forward hello-openshift-5fffbfb958-nxwgn
8080:8080 &
[1] 70971
[student@workstation ~]$ nohup: ignoring input and appending output to 'nohup.out'
```

```
[student@workstation ~]$ curl localhost:8080
Hello OpenShift!
[student@workstation ~]$ fg
nohup oc port-forward hello-openshift-5fffbfb958-nxwgn 8080:8080
^C[student@workstation ~]$
```

For this application, it's not possible to create an interactive shell. Why ?

```
[student@workstation ~]$ oc rsh hello-openshift-1-l68pp
ERRO[0000] exec failed: unable to start container process: exec: "/bin/sh": stat /bin/sh: no such
file or directory
command terminated with exit code 255
```

In this case, consider the Containerfile (Dockerfile) used to create this image:

https://github.com/openshift/origin/blob/master/examples/hello-openshift/Dockerfile

```
FROM scratch
```

There is no /bin/sh so no way to run a command that doesn't exist. You would need to either rebuild this image or use a different image for debugging like:

```
[student@workstation ~]$ oc debug pod/hello-openshift-1-l68pp --image
registry.access.redhat.com/ubi8:latest
Starting pod/hello-openshift-1-l68pp-debug, command was: /hello-openshift
Pod IP: 10.129.8.120
If you don't see a command prompt, try pressing enter.
sh-4.4$ ls
sh-4.4$ env | grep SERVICE
HELLO_OPENSHIFT_SERVICE_PORT_8080_TCP=8080
HELLO_OPENSHIFT_SERVICE_HOST=172.30.91.2
KUBERNETES_SERVICE_PORT_HTTPS=443
HELLO_OPENSHIFT_SERVICE_PORT=8080
KUBERNETES_SERVICE_PORT=443
HELLO_OPENSHIFT_SERVICE_PORT_8888_TCP=8888
KUBERNETES_SERVICE_HOST=172.30.0.1
sh-4.4$ whoami
1001180000
```

Let's clean up and complete this exercise:

```
[student@workstation ~]$ oc delete deploymentconfig.apps.openshift.io/hello-openshift
deploymentconfig.apps.openshift.io "hello-openshift" deleted
[student@workstation ~]$ oc delete all -l app=hello-openshift
service "hello-openshift" deleted
imagestream.image.openshift.io "hello-openshift" deleted
```

How to deploy mysql database in openshift using a template?

Now, let’s deploy a mysql database using a prebuilt template:

```
[student@workstation ~]$ oc new-app --template=mysql-persistent -p MYSQL_USER=user1
-p MYSQL_PASSWORD=mypa55 -p MYSQL_DATABASE=testdb -p
MYSQL_ROOT_PASSWORD=r00tpa55 -p VOLUME_CAPACITY=10Gi
[student@workstation ~]$ oc status -h
[student@workstation ~]$ oc status
In project rhn-support-ablum-mysql-openshift on server
https://api.ocp-na2.prod.nextcle.com:6443

svc/mysql-openshift - 172.30.192.167:3306
  dc/mysql-openshift deploys istag/mysql-openshift:latest
        deployment #1 deployed 45 seconds ago - 1 pod


3 infos identified, use 'oc status --suggest' to see details.
[student@workstation ~]$ oc get all
NAME                     READY   STATUS      RESTARTS   AGE
pod/mysql-openshift-1-deploy   0/1    Completed   0          3m40s
pod/mysql-openshift-1-zlzbr   1/1      Running        0        3m37s

NAME                              DESIRED   CURRENT   READY   AGE
replicationcontroller/mysql-openshift-1   1     1         1         3m41s

NAME                     TYPE           CLUSTER-IP           EXTERNAL-IP   PORT(S)     AGE
service/mysql-openshift   ClusterIP   172.30.139.239   <none>       3306/TCP   3m42s

NAME                                     REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfig.apps.openshift.io/mysql-openshift   1      1         1
config,image(mysql-openshift:latest)
```

```
NAME                                IMAGE REPOSITORY
        TAGS  UPDATED
imagestream.image.openshift.io/mysql-openshift
default-route-openshift-image-registry.apps.na45.prod.nextcle.com/rhn-support-ablum-mysql-op
enshfit/mysql-openshift   latest   About an hour ago
```

Maybe, you see this failure:

```
  Warning  FailedAttachVolume  37s (x4 over 3m44s)  attachdetach-controller
AttachVolume.Attach failed for volume "pvc-b253a9e9-7eee-4a3b-bc74-93c99380f070" :
Volume "8e5a6442-df79-4127-87b9-8b824d0c3662" failed to be attached within the alloted time
  Warning  FailedMount       25s (x2 over 2m42s)  kubelet                Unable to attach or
mount volumes: unmounted volumes=[mysql-data], unattached volumes=[mysql-data
default-token-pdcdd]: timed out waiting for the condition
```

Retry with:

```
[student@workstation ~]$ oc rollout latest dc/mysql
deploymentconfig.apps.openshift.io/mysql rolled out
```

If this still fails with storage try a different deployment using:
https://catalog.redhat.com/software/containers/search?q=mysql

```
[student@workstation ~]$ oc new-app --as-deployment-config
--docker-image=registry.redhat.io/rhel8/mysql-80 --name=mysql-openshift -e
MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 -e MYSQL_DATABASE=testdb -e
MYSQL_ROOT_PASSWORD=r00tpa5
```

OR

```
[student@workstation ~]$ oc get templates -n openshift | grep mysql
mysql-ephemeral                          MySQL database service, without persistent
storage. For more information abou...   8 (3 generated)   3

[student@workstation ~]$ oc delete all --all
pod "mysql-2-bzjgl" deleted
pod "mysql-2-deploy" deleted
replicationcontroller "mysql-1" deleted
replicationcontroller "mysql-2" deleted
```

```
service "mysql" deleted
deploymentconfig.apps.openshift.io "mysql" deleted
[student@workstation ~]$ oc delete secrets mysql
secret "mysql" deleted
[student@workstation ~]$ oc new-app --template=mysql-ephemeral  -p
MYSQL_USER=user1 -p MYSQL_PASSWORD=mypa55 -p MYSQL_DATABASE=testdb -p
MYSQL_ROOT_PASSWORD=r00tpa55

[student@workstation ~]$ oc get pods
NAME            READY   STATUS      RESTARTS   AGE
mysql-1-deploy   0/1   Completed   0          2m26s
mysql-1-kgx5g         1/1       Running         0         2m23s

[student@workstation ~]$ oc logs mysql-1-kgx5g
[student@workstation ~]$ nohup oc port-forward mysql-1-kgx5g 3306:3306 &
[1] 3626
[student@workstation ~]$ nohup: ignoring input and appending output to 'nohup.out'

[student@workstation ~]$ mysql -uuser1 -pmypa55 --protocol tcp -h localhost
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 35
Server version: 8.0.21 Source distribution

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| testdb             |
+--------------------+
2 rows in set (0.00 sec)

mysql> exit
Bye
```

```
[student@workstation ~]$ **fg**
nohup oc port-forward mysql-1-kgx5g 3306:3306
**^C**[student@workstation ~]$
```

CLEANUP

```
[student@workstation ~]$ oc delete project ${RHT_OCP4_DEV_USER}-mysql-openshift
project.project.openshift.io "rhn-support-ablum-mysql-openshift" deleted
[student@workstation ~]$ lab openshift-resources finish

Completing the Guided Exercise: Deploying a Database Server on OpenShift

 · Deleting the 'rhn-support-ablum-mysql-openshift' project....  SUCCESS
```

How can we better understand the different networking layers in k8s ?

WITHIN A CONTAINER

containers within a Pod can all reach each other’s ports on localhost.

This also means that containers within a Pod must coordinate port usage, but this is no different from processes in a VM

```
[student@workstation php-helloworld]$ oc rsh php-helloworld-1-wk49d
sh-4.2$ lsof -i -P
COMMAND PID        USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
httpd    1 1008920000 3u  IPv4 54259547     0t0  TCP *:8080 (LISTEN)
httpd    1 1008920000 4u  IPv4 54259553     0t0  TCP *:8443 (LISTEN)

sh-4.2$ curl localhost:8080
Hello, World! php version is 7.3.11

sh-4.2$ curl -k https://localhost:8443
Hello, World! php version is 7.3.11
```

Processes running within the same container share the same networking in the same way as processes running in the same virtual machine:

```
sh-4.2$ ps -ef
UID      PID   PPID  C STIME TTY             TIME CMD
1008920+       1        0  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+      35        1  0 18:35 ?    00:00:00 /usr/bin/cat
1008920+      36        1  0 18:35 ?    00:00:00 /usr/bin/cat
1008920+      37        1  0 18:35 ?    00:00:00 /usr/bin/cat
1008920+      38        1  0 18:35 ?    00:00:00 /usr/bin/cat
1008920+      39        1  0 18:35 ?    00:00:01 httpd -D FOREGROUND
1008920+      46        1  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+      55        1  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+      56        1  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+      57        1  0 18:35 ?    00:00:01 httpd -D FOREGROUND
1008920+      58        1  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+      66        1  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+      69        1  0 18:35 ?    00:00:00 httpd -D FOREGROUND
1008920+     103        1  0 19:20 ?    00:00:00 httpd -D FOREGROUND
1008920+     109        0  0 19:20 pts/0          00:00:00 /bin/sh
1008920+     215      109  0 19:42 pts/0      00:00:00 ps -ef
```

Let’s modify the depolymentconfig and add another container running in the same pod:

```
[student@workstation php-helloworld]$ oc edit
deploymentconfig.apps.openshift.io/php-helloworld

          - image:
image-registry.openshift-image-registry.svc:5000/rhn-support-ablum-route/php-helloworld@sha2
56:40891a7ac25f42cb7d90e6b6959a054cca04e28c9bafbe77ffdfb60327d00f32
          imagePullPolicy: Always
          name: php-helloworld
          ports:
          - containerPort: 8080
            protocol: TCP
          - containerPort: 8443
            protocol: TCP
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          - image: registry.access.redhat.com/rhel7
            imagePullPolicy: Always
            name: test1
            command: ["/usr/bin/sleep","5000"]
            resources: {}
        dnsPolicy: ClusterFirst
        terminationGracePeriodSeconds: 30
```

```
[student@workstation php-helloworld]$ oc describe pod php-helloworld-3-9gl4f
IP:                 10.129.21.132

[student@workstation php-helloworld]$ oc rsh -c test1 pod/php-helloworld-3-9gl4f
sh-4.2$
sh-4.2$

sh-4.2$ curl 10.129.21.132:8080
Hello, World! php version is 7.1.30

sh-4.2$ curl localhost:8080
Hello, World! php version is 7.1.30
```

Wait….what ? The httpd process isn't running here….

```
sh-4.2$ ps -ef
UID       PID   PPID  C STIME TTY          TIME CMD
1008920+    1      0  0 20:11 ?        00:00:00 /usr/bin/sleep 5000
1008920+   28      0  0 20:21 pts/0    00:00:00 /bin/sh
1008920+   39     28  0 20:22 pts/0    00:00:00 ps -ef
```

Again. Containers in the same pod are like processes in the same VM

POD TO POD

Let's fire up a new pod. This can help with port management (different procs trying to listen on the same port inside the same pod) along with providing scaling benefits:

```
[student@workstation ~]$ oc new-app --name mypod
--docker-image="registry.access.redhat.com/rhel7"
```

We will need to edit the pod to add a command that

```
[student@workstation ~]$ oc edit deployment mypod
        spec:
        containers:
        - image:
registry.access.redhat.com/rhel7@sha256:d8e52fab67bc27384fe5f4022e8c5e5d83a7901b83df
7ed41d4a216aea57b44c
        imagePullPolicy: Always
        name: mypod
        command: ["/usr/bin/sleep","5000"]
```

```
[student@workstation ~]$ oc get pods -l app=mypod -o wide
NAME           READY   STATUS     RESTARTS   AGE     IP                NODE
NOMINATED NODE   READINESS GATES
mypod-2-6hgpn    1/1   Running    0            2m1s   10.130.21.64   ip-10-0-220-55.ec2.internal
<none>         <none>
```

```
[student@workstation ~]$ oc get pods -l app=php-helloworld -o wide
NAME                     READY   STATUS     RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
php-helloworld-3-9gl4f   2/2    Running    0            34m   10.129.21.132
ip-10-0-218-157.ec2.internal   <none>         <none>
```

So, two separate pods, with different ips but both in the "Pod SDN".
Let's test Pod to Pod communications by connecting to the php-helloworld's IP from a shell in mypod's only container:

```
[student@workstation ~]$ oc rsh mypod-2-6hgpn
sh-4.2$ curl 10.129.21.132:8080
```

```
Hello, World! php version is 7.1.30
```

This works. So, pod-to-pod communication work using the pod SDN.

But, what happens when pods are restarted...

```
[student@workstation ~]$ oc delete pod php-helloworld-3-9gl4f
pod "php-helloworld-3-9gl4f" deleted
[student@workstation ~]$  oc get pods -l app=php-helloworld -o wide
NAME                      READY   STATUS     RESTARTS   AGE   IP            NODE
NOMINATED NODE   READINESS GATES
php-helloworld-3-5vx7d   2/2  Running   0              43s   10.129.21.183
ip-10-0-218-157.ec2.internal   <none>          <none>
```

The IP associated with the php-helloworld pod has changed.

```
[student@workstation ~]$ oc rsh mypod-2-6hgpn
sh-4.2$ curl 10.129.21.132:8080
curl: (7) Failed connect to 10.129.21.132:8080; No route to host
```

So, it would be difficult for containers running in different pods to keep track of the podIP as it might change over time as pods are restarted or scaled.

What can help ? Services:

POD TO SERVICE

```
[student@workstation php-helloworld]$ oc get svc
NAME            TYPE            CLUSTER-IP          EXTERNAL-IP   PORT(S)               AGE
php-helloworld   ClusterIP   172.30.237.166   <none>         8080/TCP,8443/TCP   110m
[student@workstation php-helloworld]$ oc rsh -c test1 pod/php-helloworld-3-9gl4f

sh-4.2$ curl 172.30.237.166:8080
Hello, World! php version is 7.1.30
```

Pods are able to use the service IP which will load balance the requests to pods that are labeled the same as the service selector.

Under the hood, a kubernetes service called “kube-proxy” handles this using iptables -L -t nat rules on the schedulable nodes.

How can you discover a service ?

1. Using oc commands (ie using the openshift API)

```
[student@workstation ~]$ oc get service
NAME           TYPE          CLUSTER-IP          EXTERNAL-IP   PORT(S)             AGE
php-helloworld   ClusterIP   172.30.237.166   <none>        8080/TCP,8443/TCP   164m
[student@workstation ~]$ oc get endpoints
NAME           ENDPOINTS                           AGE
php-helloworld   10.129.21.183:8443,10.129.21.183:8080   165m
```

2. DNS each service is dynamically assigned:

   SVC_NAME.PROJECT_NAME.svc.cluster.local

```
[student@workstation ~]$ oc get svc
NAME           TYPE          CLUSTER-IP          EXTERNAL-IP   PORT(S)             AGE
php-helloworld   ClusterIP   172.30.237.166   <none>      8080/TCP,8443/TCP   179m

[student@workstation ~]$ oc project
Using project "rhn-support-ablum-route" on server
"https://api.ocp-na2.prod.nextcle.com:6443".

[student@workstation ~]$ oc rsh mypod-2-6hgpn
sh-4.2$ curl php-helloworld.rhn-support-ablum-route.svc.cluster.local:8080
Hello, World! php version is 7.1.30

sh-4.2$ cat /etc/resolv.conf
search rhn-support-ablum-route.svc.cluster.local svc.cluster.local cluster.local ec2.internal
nameserver 172.30.0.10
options ndots:5
```

3. Environment variables

For each service inside an OpenShift project, the following environment variables are automatically defined and injected into containers for all pods inside the same project:

• SVC_NAME_SERVICE_HOST is the service IP address.
• SVC_NAME_SERVICE_PORT is the service TCP port.

```
sh-4.2$ env | grep PHP_HELLOWORLD
PHP_HELLOWORLD_PORT=tcp://172.30.237.166:8080
PHP_HELLOWORLD_SERVICE_PORT_8443_TCP=8443
```

```
PHP_HELLOWORLD_SERVICE_PORT_8080_TCP=8080
PHP_HELLOWORLD_PORT_8443_TCP_ADDR=172.30.237.166
PHP_HELLOWORLD_PORT_8080_TCP_PORT=8080
PHP_HELLOWORLD_SERVICE_HOST=172.30.237.166
PHP_HELLOWORLD_SERVICE_PORT=8080
PHP_HELLOWORLD_PORT_8443_TCP=tcp://172.30.237.166:8443
PHP_HELLOWORLD_PORT_8080_TCP_ADDR=172.30.237.166
PHP_HELLOWORLD_PORT_8443_TCP_PORT=8443
PHP_HELLOWORLD_PORT_8080_TCP=tcp://172.30.237.166:8080
PHP_HELLOWORLD_PORT_8443_TCP_PROTO=tcp
PHP_HELLOWORLD_PORT_8080_TCP_PROTO=tcp

sh-4.2$ curl
${PHP_HELLOWORLD_SERVICE_HOST}:${PHP_HELLOWORLD_SERVICE_PORT}
Hello, World! php version is 7.1.30
```

NOTE: The pod must be created AFTER a given service or environment variables won't be available. For example,

```
[student@workstation ~]$ oc create service clusterip myservice --tcp=12345:8080
service/myservice created

[student@workstation ~]$ oc get services
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
myservice      ClusterIP   172.30.96.95     <none>        12345/TCP           11s
php-helloworld ClusterIP   172.30.237.166   <none>        8080/TCP,8443/TCP   3h7m
```

Let's see if there are any environment variables with MYSERVICE:

```
[student@workstation ~]$ oc rsh mypod-2-6hgpn
sh-4.2$ env | grep MYSERVICE
sh-4.2$
sh-4.2$ exit
exit
command terminated with exit code 1
```

Let's try again after restarting a pod:

```
[student@workstation ~]$ oc delete pod mypod-2-6hgpn
pod "mypod-2-6hgpn" deleted

[student@workstation ~]$ oc get pods -l app=mypod
NAME           READY   STATUS    RESTARTS   AGE
```

```
mypod-2-s8wqw   1/1 Running   0              58s
```

```
[student@workstation ~]$ oc exec mypod-58d86ccb46-prw7k -- env | grep MYSERVICE
MYSERVICE_PORT_12345_TCP=tcp://172.30.96.95:12345
MYSERVICE_SERVICE_HOST=172.30.96.95
MYSERVICE_SERVICE_PORT_12345_8080=12345
MYSERVICE_PORT=tcp://172.30.96.95:12345
MYSERVICE_PORT_12345_TCP_ADDR=172.30.96.95
MYSERVICE_PORT_12345_TCP_PORT=12345
MYSERVICE_PORT_12345_TCP_PROTO=tcp
MYSERVICE_SERVICE_PORT=12345
```

(of course there are no endpoints so http traffic wont actually get handled)

```
[student@workstation ~]$ oc get endpoints
NAME           ENDPOINTS                               AGE
myservice      <none>                                 5m23s
php-helloworld   10.129.21.183:8443,10.129.21.183:8080   3h12m
```

EXTERNAL TO SERVICE (ROUTE)

The “Ingress router” uses DNS wildcards to expose a service external to the OCP cluster. The route created points to a service (that is managed by kube-proxy).

```
[student@workstation ~]$ oc get routes
NAME           HOST/PORT                                              PATH   SERVICES     PORT
         TERMINATION   WILDCARD
php-helloworld   php-helloworld-rhn-support-ablum-route.apps.ocp-na2.prod.nextcle.com
php-helloworld   8080-tcp              None
```

```
[student@workstation ~]$ oc describe route php-helloworld
Name:                    php-helloworld
Namespace:    rhn-support-ablum-route
Created:          3 hours ago
Labels:                  app=php-helloworld
Annotations:    openshift.io/host.generated=true
Requested Host:         php-helloworld-rhn-support-ablum-route.apps.ocp-na2.prod.nextcle.com
                          exposed on router default (host apps.ocp-na2.prod.nextcle.com) 3 hours ago
Path:             <none>
```

```
TLS Termination:    <none>
Insecure Policy:    <none>
Endpoint Port:             8080-tcp

Service:    php-helloworld
Weight:            100 (100%)
Endpoints:    10.129.21.183:8443, 10.129.21.183:8080
```

Suppose we want to change the default so that a different hostname is used (maybe one we control the DNS records for). To configure the ingress router to use that name to point to a given service lets use:

```
[student@workstation ~]$  source /usr/local/etc/ocp4.config
[student@workstation ~]$  oc expose service php-helloworld
--name=${RHT_OCP4_DEV_USER}-xyz
route.route.openshift.io/rhn-support-ablum-xyz expose

[student@workstation ~]$ curl
rhn-support-ablum-xyz-rhn-support-ablum-route.apps.ocp-na2.prod.nextcle.com
Hello, World! php version is 7.1.30
```

Of course, you can’t just use ANY fqdn, even those you don’t manage (or own). Consider:

```
[student@workstation ~]$ oc expose service php-helloworld
--hostname=www.example.com
route.route.openshift.io/php-helloworld exposed

[student@workstation ~]$ oc get routes
NAME                    HOST/PORT                                              PATH
SERVICES     PORT            TERMINATION   WILDCARD
php-helloworld          www.example.com
php-helloworld   8080-tcp                None
rhn-support-ablum-xyz
rhn-support-ablum-xyz-rhn-support-ablum-route.apps.ocp-na2.prod.nextcle.com
php-helloworld   8080-tcp                None

[student@workstation ~]$ oc describe route php-helloworld
Name:                        php-helloworld
Namespace:    rhn-support-ablum-route
Created:           About a minute ago
Labels:                      app=php-helloworld
Annotations:    <none>
Requested Host:          www.example.com
```

```
[student@workstation ~]$ curl www.example.com
curl: (6) Could not resolve host: www.example.com; Unknown error
```

For routes secured using x509 certificates consider:

```
[student@workstation ~]$ oc create route -h
```

```
[student@workstation php-helloworld]$ oc expose service php-helloworld --name ablum-test
--hostname=rhn-support-ablum-foo.apps.na45.prod.nextcle.com
route.route.openshift.io/ablum-test exposed
```

To clean up run:

```
[student@workstation ~]$ lab openshift-routes finish
```

In SUMMARY:

https://kubernetes.io/docs/concepts/cluster-administration/networking/
There are really 4 networking problems to solve:

1. Highly-coupled container-to-container communications: this is solved by pods and localhost communications.
2. Pod-to-Pod communications: this is the primary focus of this document.
3. Pod-to-Service communications: this is covered by services.
4. External-to-Service communications: this is covered by services.

CREATING APPLICATIONS WITH SOURCE-TO-IMAGE

Study figure 6.8

1. Start a container from a base container image called the builder image, which includes a programming language runtime and essential development tools such as compilers and package managers.

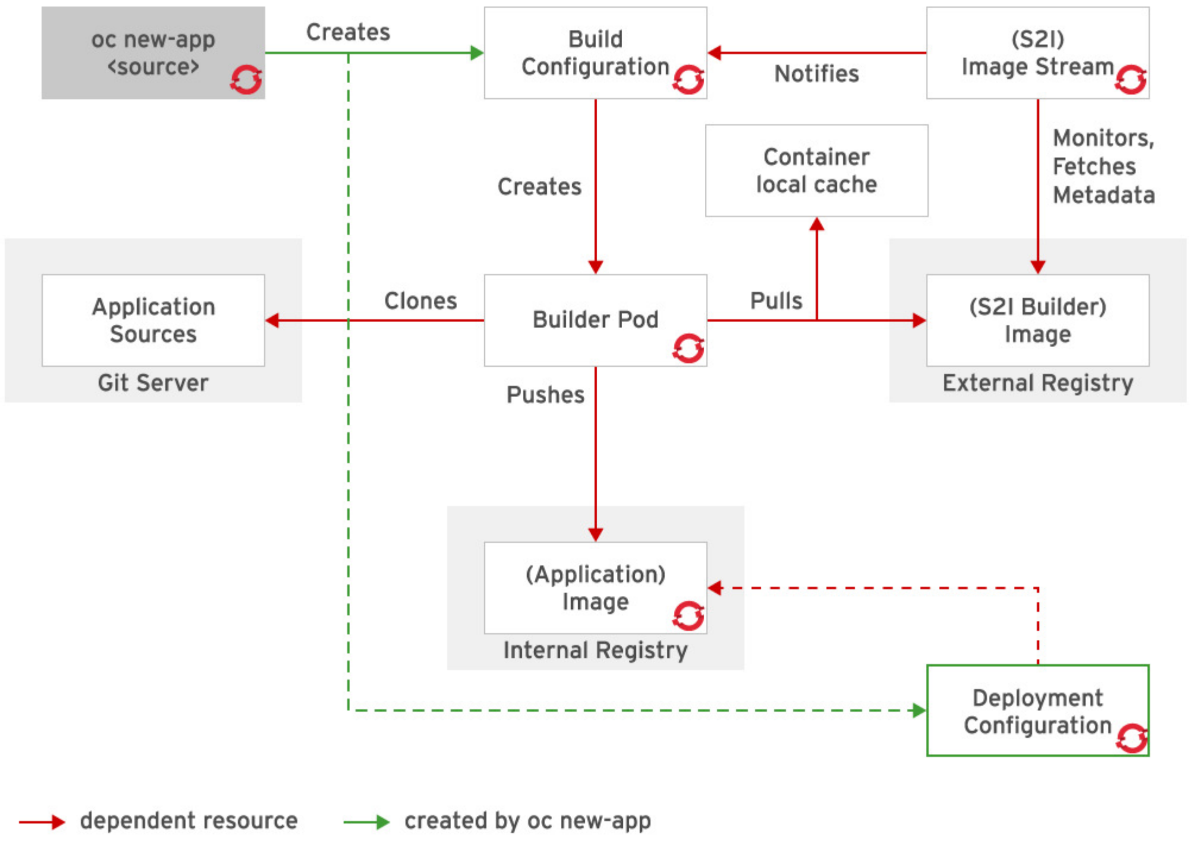
2. Fetch the application source code, usually from a Git server, and send it to the container.

3. Build the application binary files inside the container.

Build tools are run at this point:

- Yum
- Bundler (for ruby)
- Maven (for java)
- Npm (nodejs)
- Gcc (for C programs)

4. Save the container, after some clean up, as a new container image, which includes the programming language runtime and the application binaries.



GUIDED PRACTICE

```
[student@workstation ~]$ lab openshift-s2i start
```

The first part of the s2i process involves building a container.

How to inspect a builder image ?

Let's inspect a sample "builder" image that assists in building a container for a webserver serving up a web application written in php.

Navigate to https://catalog.redhat.com/software/containers/explore and search for php, check the "builder" facet in the search

Look for the card rhsc/php-73-rhel7. Click get this image.

Let's pull it with podman so we can easily inspect it. We will have to log in to get this one:

```
[root@workstation ~]# podman pull registry.access.redhat.com/rhscl/php-73-rhel7
Trying to pull registry.access.redhat.com/rhscl/php-73-rhel7...
  unsupported: This repo requires terms acceptance and is only available on registry.redhat.io
Error: error pulling image "registry.access.redhat.com/rhscl/php-73-rhel7": unable to pull registry.access.redhat.com/rhscl/php-73-rhel7: unable to pull image: Error initializing source docker://registry.access.redhat.com/rhscl/php-73-rhel7:latest: Error reading manifest latest in registry.access.redhat.com/rhscl/php-73-rhel7: unsupported: This repo requires terms acceptance and is only available on registry.redhat.io
```

```
[root@workstation ~]# podman login registry.redhat.io
Username: rhn-support-ablum
Password:
Login Succeeded!
[root@workstation ~]# podman pull registry.redhat.io/rhscl/php-73-rhel7

[root@workstation ~]# podman inspect fd | less

                    "STI_SCRIPTS_URL=image:///usr/libexec/s2i",
                    "STI_SCRIPTS_PATH=/usr/libexec/s2i",

            "Cmd": [
                    "/bin/sh",
                    "-c",
                    "$STI_SCRIPTS_PATH/usage"
            ]
```

```
[root@workstation ~]# podman run fd
This is a S2I PHP-7.3 rhel base image:

To use it in Openshift, run:

oc new-app php:7.3~https://github.com/sclorg/cakephp-ex.git

To access the application:

oc get pods
oc exec <pod> -- curl 127.0.0.1:8080

Alternatively, to run the image directly using podman or docker, or how to use it as a parent
image in a Dockerfile, see documentation at
https://github.com/sclorg/s2i-php-container/blob/master/7.3/README.md


[root@workstation ~]# podman run -it --entrypoint /bin/bash --user 0 fd

bash-4.2# cd /usr/libexec/s2i/
bash-4.2# ls
assemble  run  save-artifacts  usage
bash-4.2# cat usage
bash-4.2# cat run
#!/bin/bash
(this is what will be run later in the deployed application)

bash-4.2# cat assemble


#!/bin/bash

set -e

shopt -s dotglob
```

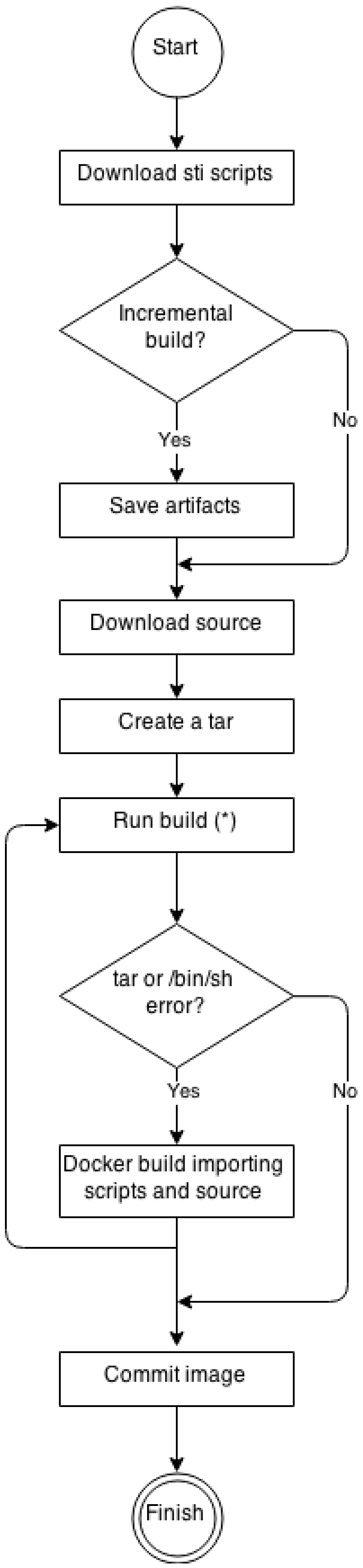
echo "---> Installing application source..."
mv /tmp/src/* ./   <-- Moves the php source code to the container working dir ….. THINK COPY Dockerfile instruction

if [-f composer.json]; then
  echo "Found 'composer.json', installing dependencies using composer.phar... "

^-- Installs dependencies needed by the application …. THINK various RUN instructions

...SNIP…

From
https://docs.openshift.com/container-platform/4.2/builds/build-strategies.html#builds-strategy-s2i-override-builder-image-scripts_build-strategies


Start
Download sti scripts
Incremental build?
Yes
No
Save artifacts
Download source
Create a tar
Run build (*)
tar or /bin/sh error?
Yes
No
Docker build importing scripts and source
Commit image
Finish

```
[student@workstation 5.6]$ cd ~/DO180-apps/
[student@workstation DO180-apps]$ git checkout master
Already on 'master'
[student@workstation DO180-apps]$
[student@workstation DO180-apps]$
[student@workstation DO180-apps]$ git checkout -b s2i
Switched to a new branch 's2i'
[student@workstation DO180-apps]$ git push -u origin s2i
Username for 'https://github.com': ajblum
Password for 'https://ajblum@github.com':
Total 0 (delta 0), reused 0 (delta 0)
[student@workstation DO180-apps]$ cat php-helloworld/index.php
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
?>

[student@workstation DO180-apps]$ source /usr/local/etc/ocp4.config

[student@workstation DO180-apps]$ oc login -u ${RHT_OCP4_DEV_USER} -p
${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.

You don't have any projects. You can try to create a new project, by running

        oc new-project <projectname>

[student@workstation DO180-apps]$ oc new-project ${RHT_OCP4_DEV_USER}-s2i

[student@workstation DO180-apps]$ oc new-app -h

  -i, --image-stream=[]: Name of an image stream to use in the app.

        --context-dir='': Context directory to be used for the build.

        --name='': Set name to use for generated application artifacts

[student@workstation DO180-apps]$ oc get is -n openshift
```

NOTE: image streams don't actually hold the container image data but they are more powerful in that they are aware of changes in the repo that can be used as a trigger for (re)deployment of an application.

OpenShift components such as builds and deployments can watch an image stream to receive notifications when new images are added and react by performing a build or a deployment.

https://blog.openshift.com/image-streams-faq/

```
[student@workstation DO180-apps]$ oc get is -n openshift php
NAME   IMAGE REPOSITORY                                                        TAGS
UPDATED
php    default-route-openshift-image-registry.apps.ocp-na2.prod.nextcle.com/openshift/php
7.0,7.1,7.2,latest   2 months ago

[student@workstation DO180-apps]$ oc describe is php -n openshift


7.3 (latest)
  tagged from registry.redhat.io/rhscl/php-73-rhel7:latest
        prefer registry pullthrough when referencing this tag

  Build and run PHP 7.3 applications on RHEL 7. For more information about using this builder
image, including OpenShift considerations, see
https://github.com/sclorg/s2i-php-container/blob/master/7.3/README.md

7.2
  tagged from registry.redhat.io/rhscl/php-72-rhel7:latest
        prefer registry pullthrough when referencing this tag

  Build and run PHP 7.2 applications on RHEL 7. For more information about using this builder
image, including OpenShift considerations, see
https://github.com/sclorg/s2i-php-container/blob/master/7.2/README.md
```

Let’s use the latest image stream tag 7.3:

```
[student@workstation DO180-apps]$ oc new-app -i php:7.3 --name=php-helloworld
https://github.com/${RHT_OCP4_GITHUB_USER}/DO180-apps#s2i --context-dir
php-helloworld

[student@workstation DO180-apps]$ oc get pods
NAME                   READY   STATUS     RESTARTS   AGE
```

```
php-helloworld-1-build   1/1    Running   0                 18s

[student@workstation DO180-apps]$ oc get bc
NAME             TYPE   FROM           LATEST
php-helloworld   Source   Git@s2i   1
[student@workstation DO180-apps]$ oc logs -f bc/php-helloworld

…
Writing manifest to image destination
Storing signatures
Successfully pushed
image-registry.openshift-image-registry.svc:5000/rhn-support-ablum-s2i/php-helloworld@sha25
6:d39d3b8e0e1d5b95c7ef06b755bdbd538bae0b8139e12ef6b138c5fc5238c51f
Push successful

[student@workstation DO180-apps]$ oc logs deployment.apps/php-helloworld -f
-> Cgroups memory limit is set, using HTTPD_MAX_REQUEST_WORKERS=102
..SNIP..
[Fri Jan 24 18:15:47.579737 2020] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.34 (Red
Hat) OpenSSL/1.0.2k-fips configured -- resuming normal operations
[Fri Jan 24 18:15:47.579776 2020] [core:notice] [pid 1] AH00094: Command line: 'httpd -D
FOREGROUND'
```

How did the builder pod build the image ?

Let’s look at that builder pod and what containers ran there:

```
[student@workstation DO180-apps]$ oc get pod php-helloworld-1-build -o
go-template='{{range .spec.Initcontainers}}{{.name}}{{end}}'
sti-build
```

But, there also “init containers” in there responsible for pulling the source code (git) and creating the Dockerfile that includes in its FROM the builder image we gave on the command line:

```
[student@workstation DO180-apps]$ oc get pod php-helloworld-1-build -o
go-template='{{range .spec.initContainers}} {{.name}} {{end}}'
 git-clone  manage-dockerfile
```

The image used for these are all the same.

If we wanted to inject special environment variables into the build we could build include those in our builder image or customize the assemble scripts. But, if we want just a certain env variable set, we could use from `**oc new-app -h**`:

--build-env=[]: Specify a key-value pair for an environment variable to set into each build image.

Useful for programs like npm (package manager for nodejs) to customize the registry used.

```
[student@workstation DO180-apps]$ oc get service
NAME           TYPE         CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
php-helloworld   ClusterIP   172.30.201.129   <none>       8080/TCP,8443/TCP   4m10s
[student@workstation DO180-apps]$ oc expose service php-helloworld --name
${RHT_OCP4_DEV_USER}-helloworld
route.route.openshift.io/rhn-support-ablum-helloworld exposed
[student@workstation DO180-apps]$ oc get route
NAME                    HOST/PORT                                                  PATH
SERVICES      PORT          TERMINATION   WILDCARD
rhn-support-ablum-helloworld
rhn-support-ablum-helloworld-rhn-support-ablum-s2i.apps.na45.prod.nextcle.com
php-helloworld   8080-tcp                None
[student@workstation DO180-apps]$ curl
rhn-support-ablum-helloworld-rhn-support-ablum-s2i.apps.na45.prod.nextcle.com
Hello, World! php version is 7.3.11
```

Let’s change the source code of our php application and start a new s2i build.

```
[student@workstation DO180-apps]$ cd ~/DO180-apps/php-helloworld/
[student@workstation php-helloworld]$ vim index.php

<?php
print "Hello, World! php version is " . PHP_VERSION . "<br>";
print "ablum was here";
?>
```

```
[student@workstation php-helloworld]$ git add .
[student@workstation php-helloworld]$ git commit -m 'Added a message to index page
content.'
[s2i e5fde79] Added a message to index page content.
```

```
 1 file changed, 1 insertion(+)
[student@workstation php-helloworld]$ git push origin s2i
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 419 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ajblum/DO180-apps.git
   f7cd896..e5fde79  s2i -> s2i
```

Now, kick off another build:

```
student@workstation php-helloworld]$ oc get builds
NAME              TYPE    FROM          STATUS      STARTED          DURATION
php-helloworld-1  Source  Git@f7cd896   Complete    23 minutes ago   2m38s
[student@workstation php-helloworld]$ oc get bc
NAME              TYPE    FROM          LATEST
php-helloworld    Source  Git@s2i       1
[student@workstation php-helloworld]$ oc start-build php-helloworld
build.build.openshift.io/php-helloworld-2 started

[student@workstation php-helloworld]$ oc get builds
NAME              TYPE    FROM          STATUS      STARTED          DURATION
php-helloworld-1  Source  Git@f7cd896   Complete    23 minutes ago   2m38s
php-helloworld-2  Source  Git@s2i         Pending

[student@workstation php-helloworld]$ oc logs -f bc/php-helloworld
...SNIP…
Successfully pushed
image-registry.openshift-image-registry.svc:5000/rhn-support-ablum-s2i/php-helloworld@sha25
6:2eb9366fb146a7bdd9b8ae08c77203b95f3fb5127ab8c321a87d279c3a62e58d
Push successful
```

The new build result in a new image to the default-route-openshift-image-registry.apps.ocp-na2.prod.nextcle.com/rhn-support-ablum-s2i/php-helloworld repository.

Consider the imagestream:

```
[student@workstation php-helloworld]$ oc describe is php-helloworld
```

As a result of this change a new deployment is triggered:

```
[student@workstation php-helloworld]$ oc get dc
NAME            REVISION   DESIRED   CURRENT   TRIGGERED BY
php-helloworld   2       1         1         config,image(php-helloworld:latest)
[student@workstation php-helloworld]$ oc logs -f dc/php-helloworld
```

And a new pod is running:

```
[student@workstation php-helloworld]$ oc get pods
NAME                    READY   STATUS      RESTARTS   AGE
php-helloworld-1-build          0/1      Completed   0          26m
php-helloworld-1-deploy   0/1          Completed   0          24m
php-helloworld-2-build          0/1      Completed   0          3m9s
php-helloworld-2-deploy   0/1          Completed   0          40s
php-helloworld-2-p48dv          1/1      Running        0         30s
```

What about the message ?

```
[student@workstation php-helloworld]$ curl
rhn-support-ablum-helloworld-rhn-support-ablum-s2i.apps.ocp-na2.prod.nextcle.com
Hello, World! php version is 7.2.24
ablum was here
```

Let's clean up:

```
[student@workstation php-helloworld]$ lab openshift-s2i finish
```

EXTRA PRACTICE WITH S2I

```
[student@workstation ~]$ oc new-project ablum-s2i-practice

[student@workstation ~]$ oc new-app -h
```

```
 # Create a Ruby application based on the provided [image]~[source code] combination
  oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git
```

Navigate to https://github.com/sclorg/ruby-ex and show $ oc new-app openshift/ruby:25~https://github.com/< yourusername >/ruby-ex

“These steps assume your OpenShift deployment has the default set of ImageStreams defined”

Where are the default imagestream resources ? In the openshift project:

```
[student@workstation ~]$ oc get is -n openshift
```

```
[student@workstation ~]$ oc get is ruby -n openshift
NAME   IMAGE REPOSITORY                                                    TAGS
UPDATED
ruby   default-route-openshift-image-registry.apps.ocp-na2.prod.nextcle.com/openshift/ruby
2.3,2.4,2.5,latest   2 months ago
```

```
[student@workstation ~]$ oc describe is ruby -n openshift
2.5 (latest)
  tagged from registry.redhat.io/rhscl/ruby-25-rhel7:latest
          prefer registry pullthrough when referencing this tag
```

This is a “builder” image. See
https://catalog.redhat.com/software/containers/search?q=rhscl/ruby

Builder Image
Platform for building and running Ruby 2.5 applications

```
[student@workstation ~]$ oc -o json new-app
openshift/ruby:2.5~https://github.com/sclorg/ruby-ex.git
```

Study figure 6.8 while inspecting

```
[student@workstation ~]$ oc new-app
openshift/ruby:2.5~https://github.com/sclorg/ruby-ex.git
[student@workstation ~]$ oc get all
[student@workstation ~]$ oc logs pod/ruby-ex-1-build -f
Getting image source signatures
Copying blob
sha256:8ba884070f611d31cb2c42eddb691319dc9facf5e0ec67672fcfa135181ab3df
```

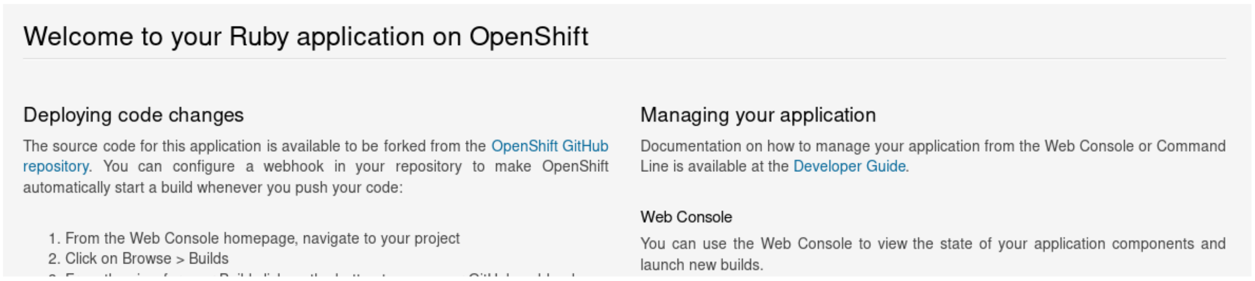
```
[student@workstation ~]$ oc logs bc/ruby-ex -f
```

Refer again to figure 6.8 during the build
https://github.com/sclorg/ruby-ex

To trigger a new build:
$ **oc get bc**
$ **oc start-build <build_name>**

```
[student@workstation ~]$ oc expose service/ruby-ex

[student@workstation ~]$ oc get routes

NAME         HOST/PORT                                              PATH   SERVICES   PORT
TERMINATION   WILDCARD

ruby-ex   ruby-ex-ablum-s2i-practice.apps.ocp-na2.prod.nextcle.com          ruby-ex
8080-tcp                None
```

http://ruby-ex-ablum-s2i-practice.apps.ocp-na2.prod.nextcle.com/



Welcome to your Ruby application on OpenShift

Deploying code changes

The source code for this application is available to be forked from the OpenShift GitHub repository. You can configure a webhook in your repository to make OpenShift automatically start a build whenever you push your code:

1. From the Web Console homepage, navigate to your project
2. Click on Browse > Builds

Managing your application

Documentation on how to manage your application from the Web Console or Command Line is available at the Developer Guide.

Web Console

You can use the Web Console to view the state of your application components and launch new builds.

Let’s look at the builder image that was used in myfourthapp:

```
[student@workstation ~]$ oc describe buildconfig.build.openshift.io/ruby-ex
[student@workstation ~]$ oc describe build.build.openshift.io/ruby-ex-1
Strategy:         Source
URL:              https://github.com/sclorg/ruby-ex.git
Commit:                     c00ecd7 (Merge pull request #25 from pvalena/master)
Author/Committer:    Honza Horak / GitHub
From Image:                 DockerImage
image-registry.openshift-image-registry.svc:5000/openshift/ruby@sha256:9866398704db9
207862bdb930b1dba4139dbaf71c6eaa6d084ea036478b28de9
```

```
[student@workstation ~]$ oc get is -n openshift
[student@workstation ~]$ oc describe is ruby -n openshift
Image Repository:    image-registry.openshift-image-registry.svc:5000/openshift/ruby
2.5 (latest)
```

```
  tagged from docker.io/centos/ruby-25-centos7:latest
        prefer registry pullthrough when referencing this tag

  Build and run Ruby 2.5 applications on CentOS 7. For more information about using this
builder image, including OpenShift considerations, see
https://github.com/sclorg/s2i-ruby-container/blob/master/2.5/README.md.
  Tags: builder, ruby
  Supports: ruby:2.5, ruby
  Example Repo: https://github.com/sclorg/ruby-ex.git

  *
docker.io/centos/ruby-25-centos7@sha256:9866398704db9207862bdb930b1dba4139dbaf
71c6eaa6d084ea036478b28de9
        3 months ago
```

Let’s pull this image from docker.io and take a look at it:

```
[student@workstation ~]$ sudo podman pull
registry.redhat.io/rhscl/ruby-25-rhel7@sha256:d773b37c133b1a59dd11e69c801a8021bfcc
6c065d1dcb8c19d2a42402596235
```

(maybe you need to login)

```
[student@workstation ~]$ sudo podman images

registry.redhat.io/rhscl/ruby-25-rhel7   none          7ca2412cdd19   3 months ago   543MB

[student@workstation ~]$ sudo podman inspect 7ca2412cdd19

...
                "STI_SCRIPTS_URL=image:///usr/libexec/s2i",
                "STI_SCRIPTS_PATH=/usr/libexec/s2i",
...
        "Entrypoint": [
                "container-entrypoint"
        ],
        "Cmd": [
                "/bin/sh",
                "-c",
                "$STI_SCRIPTS_PATH/usage"
        ],
[student@workstation ~]$ sudo podman run -it --entrypoint /bin/bash -u 0 7ca2412cdd19

bash-4.2$ cd /usr/libexec/s2i
bash-4.2$ ls
assemble  run  usage
bash-4.2$ cat assemble
…
  echo "---> Running 'bundle install ${ADDTL_BUNDLE_ARGS}' ..."
  bundle install --path ./bundle ${ADDTL_BUNDLE_ARGS}
…
```

Let's go back to the bc logs and compare:

```
[student@workstation ~]$ oc logs bc/ruby-ex | grep Running
---> Running 'bundle install --retry 2 --deployment --without development:test' ...
Running `bundle clean --verbose` with bundler 1.16.1
```

How to write your own s2i scripts?

How to create custom s2i builder images: https://blog.openshift.com/create-s2i-builder-image/

Inspecting a builder image, making a small change:

```
[root@workstation ~]# podman run -it -u root registry.redhat.io/ubi7/python-27 /bin/bash

(app-root)
(app-root)cd /usr/libexec/s2i/
(app-root)ls
assemble  init-wrapper   run  usage
(app-root)vi /usr/libexec/s2i/assemble
# set permissions for any installed artifacts
fix-permissions /opt/app-root
touch /opt/app-root/ablum_was_here

(app-root)exit
exit
[root@workstation ~]# podman ps -a
```

```
CONTAINER ID   IMAGE                                    COMMAND              CREATED
STATUS                   PORTS   NAMES                  IS INFRA
14fe9771dd0c   registry.redhat.io/ubi7/python-27:latest   container-entrypoin...   3 minutes ago
Exited (127) 4 seconds ago              distracted_chandrasekhar   false
b162b86d5f05   registry.redhat.io/ubi7/python-27:latest   container-entrypoin...   15 minutes ago
Exited (0) 4 minutes ago                happy_bohr              false
17c997ad2b33   registry.redhat.io/ubi8/python-27:latest   container-entrypoin...   28 minutes ago
Exited (127) 20 minutes ago             fervent_euclid          false
33365bae4853   registry.redhat.io/ubi8/python-27:latest   container-entrypoin...   30 minutes ago
Exited (0) 29 minutes ago               upbeat_aryabhata              false
[root@workstation ~]# podman commit 14fe9771dd0c do180:latest
[root@workstation ~]# podman push c487c0a98379 quay.io/ajblum/mytest:latest
```

```
[student@workstation ~]$ oc new-project s2i-fun
```

```
[student@workstation ~]$ oc new-app
quay.io/ajblum/mytest~https://github.com/OpenShiftDemos/os-sample-python.git
```

https://docs.openshift.com/container-platform/4.2/builds/build-strategies.html#builds-strategy-s2i-override-builder-image-scripts_build-strategies

1. Provide an ***assemble***, ***run***, or ***save-artifacts*** script in the ***.s2i/bin*** directory of your application source repository, or

2. Provide a URL of a directory containing the scripts as part of the strategy definition. For example:

```
strategy:

  sourceStrategy:

    from:

      kind: "ImageStreamTag"

      name: "builder-image:latest"

    scripts: "http://somehost.com/scripts_directory"
```

https://docs.openshift.com/container-platform/4.2/builds/build-strategies.html#images-create-s2i_build-strategies

Also, Look for “”**Writing s2i scripts**””

How can you use the s2i build images from redhat's registry?

$ **oc new-project javafun**

Try:

```
[student@workstation ~]$ oc run openjdk11
--image=registry.redhat.io/openjdk/openjdk-11-rhel7
[student@workstation ~]$ oc get all
NAME                      READY   STATUS             RESTARTS   AGE
pod/openjdk11-1-deploy    1/1     Running            0          5m
pod/openjdk11-1-fj8bt     0/1     ImagePullBackOff   0          5m
```

Create a "Registry Service Account" on https://access.redhat.com/terms-based-registry/
https://access.redhat.com/terms-based-registry/#/token/ablum-rhel8-training/openshift-secret

```
[student@workstation ~]$ vi ablum-secret.yaml
secret/1979710-ablum-rhel8-training-pull-secret created
[student@workstation ~]$ oc create -f ablum-secret.yaml

[student@workstation ~]$ oc edit deploymentconfig.apps.openshift.io/openjdk11
...
        spec:
        containers:
        - image: registry.redhat.io/openjdk/openjdk-11-rhel7
        imagePullPolicy: Always
        name: openjdk11
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        dnsPolicy: ClusterFirst
        restartPolicy: Always
        schedulerName: default-scheduler
        securityContext: {}
        terminationGracePeriodSeconds: 30
        imagePullSecrets:
        - name: 1979710-ablum-rhel8-training-pull-secret

[student@workstation ~]$ oc get all
NAME                      READY   STATUS             RESTARTS   AGE
pod/openjdk11-1-deploy    0/1     Error              0          21m
pod/openjdk11-2-hd9nb     0/1     CrashLoopBackOff   4          2m


[student@workstation ~]$ oc logs pod/openjdk11-2-hd9nb
```

Starting the Java application using /opt/jboss/container/java/run/run-java.sh ...

ERROR Neither $JAVA_MAIN_CLASS nor $JAVA_APP_JAR is set and 0 JARs found in /deployments (1 expected)

There isn't anything compiled (ie no jar) to execute. Why ? This is a "builder" image see:

https://access.redhat.com/containers/?tab=images#/registry.access.redhat.com/openjdk/openjdk-11-rhel7

So, let's use s2i to build an application using it. Let's try one from the openshift-quickstarts:

```
[student@workstation ~]$ oc delete deploymentconfig.apps.openshift.io/openjdk11
deploymentconfig.apps.openshift.io "openjdk11" deleted
[student@workstation ~]$ oc new-app
registry.redhat.io/openjdk/openjdk-11-rhel7~https://github.com/jboss-openshift/openshift-quickstarts --context-dir=undertow-servlet

W1002 12:26:19.215464       4811 dockerimagelookup.go:236] Docker registry lookup failed: Get https://registry.redhat.io/v2/openjdk/openjdk-11-rhel7/manifests/latest: unauthorized: Please login to the Red Hat Registry using your Customer Portal credentials. Further instructions can be found here: https://access.redhat.com/articles/3399531

error: unable to locate any images in image streams, local docker images with name "registry.redhat.io/openjdk/openjdk-11-rhel7"
```

```
[student@workstation ~]$ oc get serviceaccounts
NAME        SECRETS   AGE
builder 2       13m
default 2       13m
deployer   2          13m
```

```
[student@workstation ~]$ oc get secrets
NAME                          TYPE                                DATA   AGE
1979710-ablum-rhel8-training-pull-secret   kubernetes.io/dockerconfigjson          1          27m
builder-token-db4fb    kubernetes.io/service-account-token   3       12m
builder-token-gc6dw            kubernetes.io/service-account-token   3       12m
default-dockercfg-62t9g        kubernetes.io/dockercfg               1       12m
default-token-5vkjr    kubernetes.io/service-account-token   3       12m
default-token-hf589    kubernetes.io/service-account-token   3       12m
deployer-dockercfg-ttl54   kubernetes.io/dockercfg            1       12m
deployer-token-9jt9s           kubernetes.io/service-account-token   3       12m
deployer-token-p9krp           kubernetes.io/service-account-token   3       12m
```

```
[student@workstation ~]$ oc get sa default -o yaml
apiVersion: v1
imagePullSecrets:
- name: default-dockercfg-62t9g
```

Go to Download ablum-rhel8-training-secret.yaml and paste contents into a .yaml:

```
$ vi ablum-pull-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: 1979710-ablum-rhel8-training-pull-secret
data:
  .dockerconfigjson: eyJhdXRocyI…
type: kubernetes.io/dockerconfigjson


[student@workstation ~]$ oc create -f ablum-pull-secret.yaml
secret/1979710-ablum-rhel8-training-pull-secret created

[student@workstation ~]$ oc secrets link --for=pull default
1979710-ablum-rhel8-training-pull-secret

[student@workstation ~]$  oc secrets link --for=mount builder
1979710-ablum-rhel8-training-pull-secret

[student@workstation ~]$ oc get sa default -o yaml
apiVersion: v1
imagePullSecrets:
- name: default-dockercfg-62t9g
- name: 1979710-ablum-rhel8-training-pull-secret


[student@workstation ~]$ oc new-app
registry.redhat.io/openjdk/openjdk-11-rhel7~https://github.com/jboss-openshift/openshift-
quickstarts --context-dir=undertow-servlet

--> Found Docker image 781deed (4 weeks old) from registry.redhat.io for
"registry.redhat.io/openjdk/openjdk-11-rhel7"

[student@workstation ~]$ oc expose service/openshift-quickstarts

[student@workstation ~]$ curl openshift-quickstarts-javafun.apps.cluster.lab.example.com

Hello World
```

How to import your own images ?

```
[student@workstation ~]$  oc new-project myproject

[student@workstation ~]$ oc import-image myproject/mytest --from=quay.io/ajblum/mytest
--confirm --all --scheduled=true

[student@workstation ~]$ oc get all

NAME                              IMAGE REPOSITORY                                        TAGS
        UPDATED

imagestream.image.openshift.io/mytest
image-registry.openshift-image-registry.svc:5000/myproject/mytest   1.0,latest   2 minutes ago
```

Now, try to push a new image to the public registry:

```
[root@workstation ~]# podman login quay.io
Username: ajblum
Password:
Login Succeeded!
[root@workstation ~]# podman push 089c3c916a95 quay.io/ajblum/mytest:latest
```

Monitor [student@workstation ~]$ **oc describe imagestream.image.openshift.io/mytest** for changes to the sha for the istag "latest"

```
[student@workstation ~]$ oc new-app myproject/mytest:latest
```

Now, from a private repo:

https://docs.openshift.com/container-platform/4.2/openshift_images/managing-images/using-image-pull-secrets.html

(show private docker.io repo docker.io/ajblum/do180)

```
[student@workstation ~]$ oc new-project test1

[student@workstation ~]$ vim secret
```

```
[student@workstation ~]$ oc create secret docker-registry ajblum
--docker-server=docker.io --docker-username=ajblum --docker-password=`cat secret`

[student@workstation ~]$ oc secrets link default ajblum --for=pull
[student@workstation ~]$ oc describe sa default
Name:          default
Namespace:            test1
Labels:        <none>
Annotations:          <none>
Image pull secrets:  default-dockercfg-8g2q6
               ajblum
Mountable secrets:   default-token-v4g2f
               default-dockercfg-8g2q6
Tokens:        default-token-blgq4
               default-token-v4g2f
Events:        <none>

[student@workstation ~]$ oc new-app --docker-image=docker.io/ajblum/do180:latest
[student@workstation ~]$ oc expose service/do180

[student@workstation ~]$ curl do180-test1.apps.cluster.lab.example.com
Hello OpenShift!
```

How can you push images directly into the internal image registry ?

https://docs.openshift.com/container-platform/4.7/registry/securing-exposing-registry.html

```
[student@workstation ~]$ kl
[student@workstation ~]$ oc edit configs.imageregistry.operator.openshift.io
spec:
  defaultRoute: true

[student@workstation ~]$ oc get routes
[student@workstation ~]$ sudo vi /etc/containers/registries.conf
[registries.insecure]
registries = ['default-route-openshift-image-registry.apps.ocp4.example.com']
[student@workstation ~]$ oc whoami -t
[student@workstation ~]$ podman login -u kubeadmin -p $(oc whoami -t)
default-route-openshift-image-registry.apps.ocp4.example.com
```

Now, deploy an application from an imagestream

```
[student@workstation ~]$ oc project developer-route
[student@workstation ~]$ oc login -u developer -p developer
[student@workstation ~]$ oc create imagestream php-hello-dockerfile
[student@workstation ~]$ oc login -u admin -p redhat
[student@workstation ~]$  oc whoami -t
[student@workstation ~]$  oc get routes -A
[student@workstation ~]$ podman push 4dc9cf1e0b6d
default-route-openshift-image-registry.apps.ocp4.example.com/developer-route/php-hello-docke
rfile
[student@workstation ~]$  oc login -u developer -p developer
[student@workstation ~]$ oc get is
[student@workstation ~]$  oc new-app -i php-hello-dockerfile --name php-helloworld
[student@workstation ~]$ oc expose service/php-helloworld
[student@workstation ~]$  oc get routes
[student@workstation ~]$   curl php-helloworld-developer-route.apps.ocp4.example.com
[student@workstation ~]$  podman logout
default-route-openshift-image-registry.apps.ocp4.example.com
```

Can you use a build strategy based on a Containerfile/Dockerfile ?

https://docs.openshift.com/container-platform/4.8/cicd/builds/build-strategies.html#builds-strategy-docker-build_build-strategies

[student@workstation ~]$ **oc new-app --help**

```
        --strategy=: Specify the build strategy to use if you don't want to detect
(docker|pipeline|source).
```

```
[student@workstation ~]$ mkdir mytest
[student@workstation ~]$ cd mytest
[student@workstation mytest]$ vim Dockerfile
FROM registry.access.redhat.com/ubi8:latest
MAINTAINER Andrew Blum <ablum@redhat.com>
ENTRYPOINT ["/usr/bin/sleep","5000"]

[student@workstation mytest]$  cd ~
[student@workstation ~]$ oc new-project ablum-mytest
[student@workstation ~]$ oc new-app --name mytest --strategy=docker
/home/student/mytest
```

```
[student@workstation ~]$ oc start-build buildconfig.build.openshift.io/mytest --from-dir
/home/student/mytest/
Uploading directory "/home/student/mytest" as binary input for the build ...
[student@workstation ~]$ oc logs buildconfig.build.openshift.io/mytest
Receiving source from STDIN as archive ...
Replaced Dockerfile FROM image registry.access.redhat.com/ubi8:latest
Caching blobs under "/var/cache/blobs".

Pulling image
registry.access.redhat.com/ubi8@sha256:5e334d76fc059f7b44ee8fc2da6a2e8b240582d02143
64c8c88596d20b33d7f1 ...
Getting image source signatures
Copying blob
sha256:262268b65bd5f33784d6a61514964887bc18bc00c60c588bc62bfae7edca46f1
Copying blob
sha256:06038631a24a25348b51d1bfc7d0a0ee555552a8998f8328f9b657d02dd4c64c
Copying config
sha256:53ce4390f2adb1681eb1a90ec8b48c49c015e0a8d336c197637e7f65e365fa9e
Writing manifest to image destination
Storing signatures
Adding transient rw bind mount for /run/secrets/rhsm
STEP 1: FROM
registry.access.redhat.com/ubi8@sha256:5e334d76fc059f7b44ee8fc2da6a2e8b240582d02143
64c8c88596d20b33d7f1
STEP 2: MAINTAINER Andrew Blum <ablum@redhat.com>
--> de87c382d1b
STEP 3: ENTRYPOINT ["/usr/bin/sleep","5000"]
--> 2cf98c8c01a
STEP 4: ENV "OPENSHIFT_BUILD_NAME"="mytest-2"
"OPENSHIFT_BUILD_NAMESPACE"="ablum-mytest"
--> 111fc391266
STEP 5: LABEL "io.openshift.build.name"="mytest-2"
"io.openshift.build.namespace"="ablum-mytest"
STEP 6: COMMIT temp.builder.openshift.io/ablum-mytest/mytest-2:f6fb5ea5

[student@workstation ~]$ oc get pods
NAME                      READY   STATUS      RESTARTS   AGE
mytest-2-build            0/1     Completed   0          41s
mytest-656cb4fc4c-dlz5h   1/1     Running     0          24s
[student@workstation ~]$
[student@workstation ~]$
[student@workstation ~]$ oc rsh mytest-656cb4fc4c-dlz5h
sh-4.4$ ls /proc
```

```
1
sh-4.4$ cat /proc/1/cmdline
/usr/bin/coreutils--coreutils-prog-shebang=sleep/usr/bin/sleep5000sh-4.4$
```

Now, we can try to make a change:

```
[student@workstation ~]$ vi mytest/Dockerfile
ENTRYPOINT ["/usr/bin/sleep","22000"]

[student@workstation ~]$ oc start-build bc/mytest --from-dir /home/student/mytest/

[student@workstation ~]$ oc rsh mytest-699c4f694c-r7qjr
sh-4.4$ cat /proc/1/cmdline
/usr/bin/coreutils--coreutils-prog-shebang=sleep/usr/bin/sleep22000
```

.

CREATING APPLICATIONS WITH THE OPENSHIFT WEB CONSOLE

For extra information regarding the topology view

https://docs.openshift.com/container-platform/4.7/applications/application_life_cycle_management/odc-viewing-application-composition-using-topology-view.html

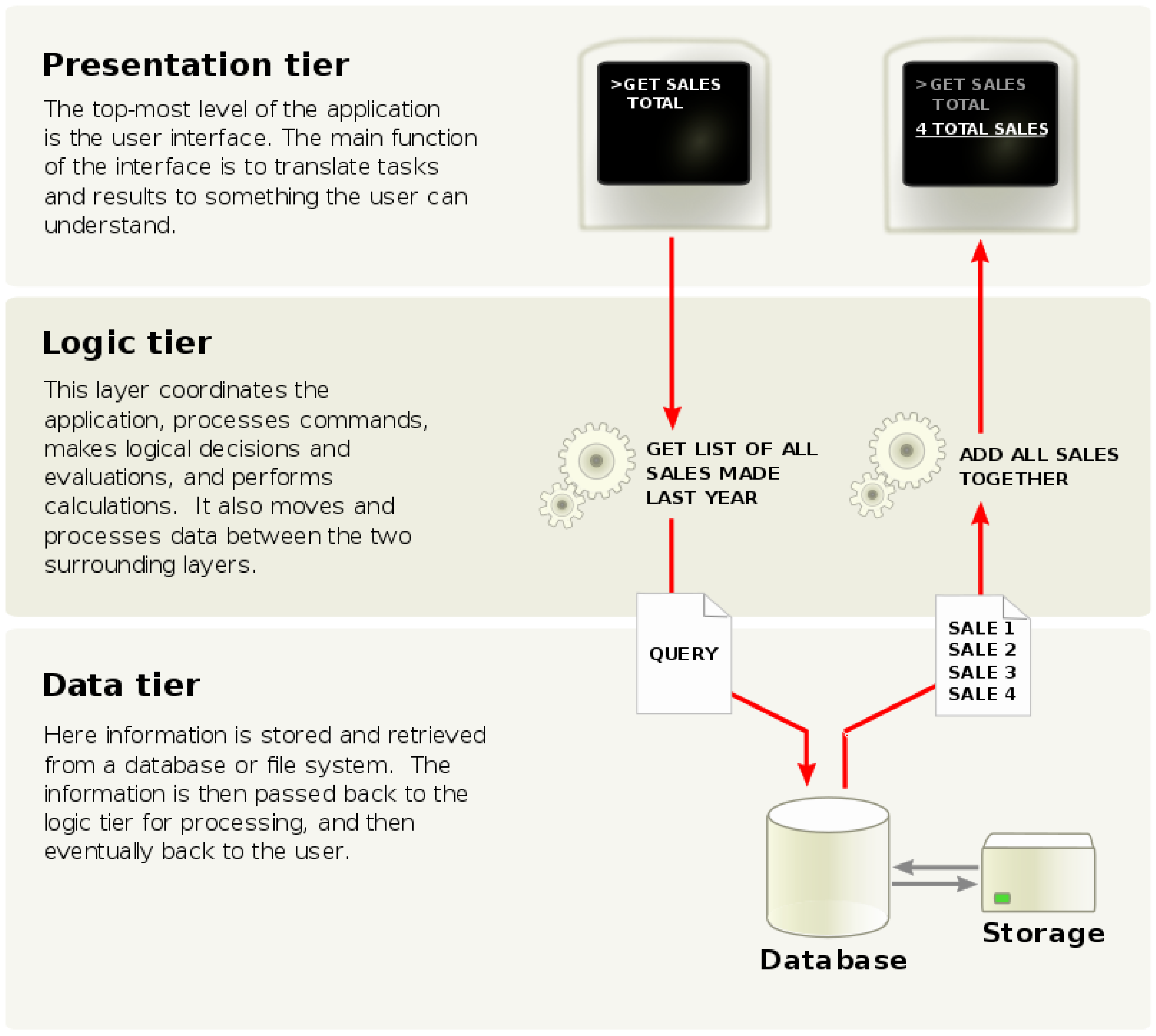
CHAPTER 7 DEPLOYING MULTI-CONTAINER APPLICATIONS

Objective: Describe considerations for containerizing applications with multiple container images.
Objective: Deploy a multi-container application on OpenShift using a template.

CONSIDERATIONS FOR MULTI-CONTAINER APPLICATIONS

What are application tiers ?

Multi-tier applications can benefit from being deployed in separate containers (or pods).
https://en.wikipedia.org/wiki/Multitier_architecture


Presentation tier
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.
>GET SALES TOTAL
>GET SALES TOTAL
4 TOTAL SALES
Logic tier
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.
GET LIST OF ALL SALES MADE LAST YEAR
ADD ALL SALES TOGETHER
QUERY
SALE 1
SALE 2
SALE 3
SALE 4
Data tier
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.
Database
Storage

https://upload.wikimedia.org/wikipedia/commons/thumb/5/51/Overview_of_a_three-tier_application_vectorVersion.svg/800px-Overview_of_a_three-tier_application_vectorVersion.svg.png

Applications that are designed in “tiers” make good candidates to separate ie:

- Presentation
- Logic (Business)
- Data (Persistence)

Openshift has advantages over podman with applications like this. See:
Figure 7.1: A restart breaks three-tiered application links

Intro to the “TODO List” application used in this course

Figure 7.2: To Do List application logical architecture

How to use podman pod to deploy multiple containers ?

```
[student@workstation ~]# **git clone https://github.com/ajblum/sampleapp.git**
[studnet@workstation sampleapp]# **cd sampleapp/**
[student@workstation sampleapp]# **podman build -t nodejs:1.0 .**
STEP 1: FROM ubi8/nodejs-12
STEP 2: ADD app-src .
f170b8ebcb1bf745f7ff56d3ae398e464a33b2ba91c1d7240ee5ff624ddf411c
STEP 3: RUN npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
up to date in 0.296s
found 0 vulnerabilities

fc17e774ec331fbe4d7e34b176e8fe422522ab1b11cf72055ea55dbbc0e6dccc
STEP 4: CMD npm run -d start
STEP 5: COMMIT nodejs:1.0

[student@workstation sampleapp]# **podman pod create --name myapp1**
[student@workstation sampleapp]# **podman pod list**
POD ID          NAME  STATUS        CREATED       # OF CONTAINERS   INFRA ID
c7cdb8663e1b   myapp1   Created   8 seconds ago   1                ac53be7d166b

[student@workstation sampleapp]# **podman create --pod myapp1 -d nodejs:1.0**
[student@workstation sampleapp]# **podman create --pod myapp1 -d ubi8:latest sleep 5000**
[student@workstation sampleapp]# **podman pod start myapp1**
[student@workstation sampleapp]$ **podman ps -a**
CONTAINER ID  IMAGE                                   COMMAND             CREATED
STATUS                PORTS  NAMES
d1b2833b71ad  registry.access.redhat.com/ubi8:latest      sleep 5000              11 seconds
ago  Up 4 seconds ago                  condescending_archimedes
199f1fe4eb1e  localhost/nodejs:1.0                    /bin/sh -c npm ru...  19 seconds ago  Up 5
seconds ago             boring_hellman
2ad4c15c175a  k8s.gcr.io/pause:3.1                                        30 seconds ago  Up 5
seconds ago             ada250d7db62-infra
```

```
[root@workstation sampleapp]# podman exec -it d1b2833b71ad /bin/bash
[root@myapp1 /]# yum install procps-ng
root@myapp1 /]# ps -ef
UID          PID     PPID  C STIME TTY          TIME CMD
root         1       0  0 15:33 ?     00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep
/usr/bin/sleep 5000
root         6       0  0 15:34 pts/0         00:00:00 /bin/bash
root         42      6  0 15:34 pts/0         00:00:00 ps -ef

[root@myapp1 /]# curl localhost:3000
Hello World[root@myapp1 /]# exit
```

```
[student@workstation sampleapp]$ podman pod list
POD ID          NAME  STATUS        CREATED       # OF CONTAINERS   INFRA ID
ada250d7db62   myapp1   Running   4 minutes ago   3                 2ad4c15c175a
contains containers and cannot be removed: container already exists
[student@workstation sampleapp]$ podman pod rm ad -f
```

How to deploy the todo list application in pod ?

```
[student@workstation ~]$ lab multicontainer-design start

[student@workstation networked]$ podman rm -a -f

[student@workstation networked]$ podman pod create --name todo -p 30081:30080

[student@workstation networked]$ podman pod list
POD ID          NAME          STATUS        CREATED       INFRA ID       # OF CONTAINERS
93eb660d7472  todo          Created       8 minutes ago  44ae8346607b  1

[student@workstation networked]$ podman pod inspect todo
(notice the infra pod, lets check to see what image it uses)

[student@workstation networked]$ podman ps -a
```

```
44ae8346607b  registry.access.redhat.com/ubi8/pause:latest                    10 minutes
ago  Created            0.0.0.0:30081->30080/tcp  93eb660d7472-infra

[student@workstation networked]$ **podman images**
(determine container image id)
[student@workstation networked]$ **podman inspect d41a45d48071**
```

The pause container provides a way to run a container indefinitely. It will wait for interruption signals which terminate its execution.

We need this to hold our shared networking namespace used across the pod.

It runs a program (catatonit -P) https://github.com/openSUSE/catatonit “A container init that is so simple it's effectively brain-dead.” haha

```
[student@workstation ~]$ cd
/home/student/DO180/labs/multicontainer-design/deploy/nodejs/nodejs-source/models

[student@workstation models]$ vi db.js

module.exports.params = {
  dbname: process.env.MYSQL_DATABASE,
  username: process.env.MYSQL_USER,
  password: process.env.MYSQL_PASSWORD,
  params: {
        host: '127.0.0.1',
        port: '3306',
        dialect: 'mysql'
  }
};
```

Pods will share networking namespace, thus 127.0.0.1 can be used for the nodetodo application to connect to the mysql database. Next, need to change the name of the image that will be built so as not to conflict with the one built previously

```
[student@workstation nodejs]$ cd
/home/student/DO180/labs/multicontainer-design/deploy/nodejs
```

Now, build this as before:

```
[student@workstation nodejs]$ ./build.sh
Preparing build folder
STEP 1: FROM registry.redhat.io/rhel8/nodejs-12:1
```

```
...SNIP...
Storing signatures
0f684b55c295d2b1870842bebc1b992d6b105abdb6cdc5482d72fd7ec78145f0
[student@workstation nodejs]$
```

Now, we will configure persistent storage just like what was done in the ./run.sh script:

```
[student@workstation nodejs]$ cd networked/
[student@workstation nodejs]$ cat run.sh
```

(we wont use this start but get some reminders about how to create a persistent volume)

```
[student@workstation networked]$ mkdir -p work1/data
[student@workstation networked]$ sudo chcon -Rt container_file_t work1
[student@workstation networked]$ podman unshare chown -R 27:27 work1
```

Now, we'll create our two containers but notice now we add in the --pod :

```
[student@workstation networked]$ podman create --pod todo --name mysql -e
MYSQL_DATABASE=items -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 -e
MYSQL_ROOT_PASSWORD=r00tpa55 -v $PWD/work1/data:/var/lib/mysql/data
registry.redhat.io/rhel8/mysql-80:1
cc0fbb27e73259c118729beb278cd4c982adca951b89dbb1a16c99d7ab198108
```

Notice, we don't need to configure mysql to use a host port since the networking will be shared within the pod.

```
[student@workstation networked]$ podman create --pod todo --name todoapi -e
MYSQL_DATABASE=items -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55
do180/todonodejs
64d59133e1af963fb64b04f96fe0bbc0700802797f84d3325143f8daacd5cb7a
```

To start the application (and both the containers) use `podman pod start` like:

```
[student@workstation networked]$ podman pod list
[student@workstation networked]$ podman pod inspect 5ae83fc0d4cf
[student@workstation networked]$ podman pod start todo

[student@workstation networked]$ podman ps
```

```
CONTAINER ID  IMAGE                          COMMAND    CREATED      STATUS
PORTS                  NAMES
cbacd5f9e5d5  localhost/do180/todonodejs-pod:latest  ./run.sh      2 minutes ago  Up 2 minutes
ago  0.0.0.0:30081->30080/tcp  todoapi
362498e6f62e  registry.redhat.io/rhel8/mysql-80:1   run-mysqld  3 minutes ago  Up 2 minutes
ago  0.0.0.0:30081->30080/tcp  mysql
```

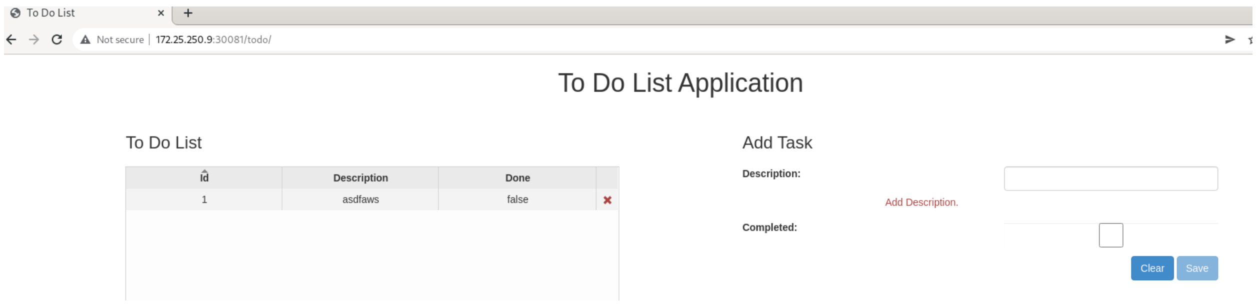
We will need to create one table in mysql manually for this application to work correctly:

```
[student@workstation networked]$ podman exec -it mysql /bin/bash
bash-4.4$ mysql -uroot

mysql> use items
Database changed
mysql> CREATE TABLE `Item` (`id` BIGINT not null auto_increment primary key,
`description` VARCHAR(100), `done` BIT);
Query OK, 0 rows affected (0.05 sec)
mysql> INSERT INTO `Item` (`id`,`description`,`done`) VALUES (1,'Pick up newspaper', 0);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO `Item` (`id`,`description`,`done`) VALUES (2,'Buy groceries', 1);
Query OK, 1 row affected (0.02 sec)
```

Now, test:

```
[student@workstation networked]$ curl http://127.0.0.1:30081/todo/api/items/1
{"id":1,"description":"Pick up newspaper","done":false}
[student@workstation networked]$ curl http://127.0.0.1:30081/todo/api/items/2
{"id":2,"description":"Buy groceries","done":true}

[student@workstation ~]$ sudo firewall-cmd --add-port 30081/tcp
success
```

Go to http://172.25.250.9:30081/todo/ and put some data in.



id	Description	Done
1	asdfaws	false

```
[student@workstation networked]$ curl http://127.0.0.1:30081/todo/api/items/1
{"id":1,"description":"asdfaws","done":false}

[student@workstation networked]$ podman pod stop todo
21e7a8c60238df5d37ecdab7e5b7dbe87a0fb0fc7af5e4b0b460a2bccaac2362


[student@workstation networked]$ podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES

[student@workstation networked]$ podman ps
CONTAINER ID  IMAGE                               COMMAND     CREATED        STATUS
PORTS                    NAMES
cbacd5f9e5d5  localhost/do180/todonodejs-pod:latest  ./run.sh      9 minutes ago  Up 24
seconds ago  0.0.0.0:30081->30080/tcp  todoapi
362498e6f62e  registry.redhat.io/rhel8/mysql-80:1   run-mysqld  9 minutes ago  Up 24 seconds
ago  0.0.0.0:30081->30080/tcp  mysql

[student@workstation networked]$ podman pod rm todo -f
21e7a8c60238df5d37ecdab7e5b7dbe87a0fb0fc7af5e4b0b460a2bccaac2362
```

TO enable systemd to manage the containers/pod:
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index#assembly_porting-containers-to-systemd-using-podman_building-running-and-managing-containers

```
[student@workstation ~]$ lab multicontainer-design finish

Cleaning up the lab for Guided Exercise: Connecting Web Application and MySQL Container

 · Stopping mysql container...................................  SUCCESS
 · Removing mysql container...................................  SUCCESS
 · Stopping todoapi container.................................  SUCCESS
 · Removing todoapi container.................................  SUCCESS
 · Removing registry.redhat.io/rhel8/mysql-80:1 image..........  SUCCESS
 · Removing registry.redhat.io/rhel8/nodejs-12:1 image.........  SUCCESS
 · Removing do180/todonodejs image............................  SUCCESS
 · Removing the project directory.............................  SUCCESS
 · Removing the solution directory............................  SUCCESS
```

DEPLOYING A MULTI-CONTAINER APPLICATION ON OPENSHIFT

DEPLOYING A MULTI-CONTAINER APPLICATION ON OPENSHIFT USING A TEMPLATE

Objective: After completing this section, students should be able to deploy a multicontainer application on OpenShift using a template.

Creating applications in Openshift require defining properties (like a name) for many resources. Often they are the same value for many of these resources:

- BuildConfig
- DeploymentConfig
- Service
- Route

Templates provide a way to simplify creation of resources that an application requires by including all the resources together as well as parameters that can be reused - like a name.

Extra TEMPLATE fun

```
[student@workstation ~]$ lab openshift-resources start
[student@workstation ~]$
[student@workstation ~]$  oc new-project ablum-template-fun
[student@workstation ~]$ oc get templates -n openshift


[student@workstation ~]$ oc get template httpd-example -n openshift
NAME           DESCRIPTION                                                   PARAMETERS
OBJECTS
httpd-example   An example Apache HTTP Server (httpd) application that serves static
content....   9 (3 blank)   5
[student@workstation ~]$ oc describe template httpd-example -n openshift
[student@workstation ~]$ oc get template httpd-example -n openshift -o yaml > httpd.yaml

[student@workstation ~]$ oc process -f httpd.yaml --parameters
NAME                DESCRIPTION
GENERATOR           VALUE
NAME                The name assigned to all of the frontend objects defined in this
template.                                    httpd-example

[student@workstation ~]$ oc process -f httpd.yaml -p NAME=myserver
[student@workstation ~]$ oc process -f httpd.yaml -p NAME=myserver -o yaml >
httpd-processed.yaml
[student@workstation ~]$ oc project
Using project "ablum-template-fun" on server "https://api.ocp-na2.prod.nextcle.com:6443".
[student@workstation ~]$ oc create -f httpd-processed.yaml
service/myserver created
route.route.openshift.io/myserver created
imagestream.image.openshift.io/myserver created
buildconfig.build.openshift.io/myserver created
deploymentconfig.apps.openshift.io/myserver created

[student@workstation ~]$ oc logs -f bc/myserver
[student@workstation ~]$ oc logs -f dc/myserver

[student@workstation ~]$ curl
myserver-ablum-template-fun.apps.ocp-na2.prod.nextcle.com | grep Welcome
  % Total      % Received % Xferd  Average Speed   Time Time    Time  Current
                                   Dload  Upload   Total   Spent Left  Speed
  0       0         0         0         0         0         0         0 --:--:-- --:--:-- --:--:--   0  <title>Welcome to
OpenShift</title>
100 37451  100 37451         0         0   355k         0 --:--:-- --:--:-- --:--:--  358k
        <h1>Welcome to your static httpd application on OpenShift</h1>
```

```
[student@workstation ~]$ oc delete project ablum-template-fun
project.project.openshift.io "ablum-template-fun" deleted
```

Additional practice if needed

```
[student@workstation networked]$ oc get template mysql-ephemeral -n openshift
NAME            DESCRIPTION                                                                 PARAMETERS
OBJECTS
mysql-ephemeral   MySQL database service, without persistent storage. For more information
abou...   8 (3 generated)   3

[student@workstation networked]$ oc get template mysql-ephemeral -n openshift -o yaml |
less

apiVersion: template.openshift.io/v1
kind: Template
...SNIP...
metadata:
  annotations:
        tags: database,mysql                              <--- Makes it easier to find in UI
  labels:
        samplesoperator.config.openshift.io/managed: "true"
  name: mysql-ephemeral                            <-- Template name (not the objects)
objects:                                   <-- these are the ocp resources to be created
- apiVersion: v1
  kind: Secret
...SNIP...
        name: ${DATABASE_SERVICE_NAME}    <- notice how this parameter is reused
...SNIP...
parameters:
...SNIP..
```

```
- description: The name of the OpenShift Service exposed for the database.
  displayName: Database Service Name
  name: DATABASE_SERVICE_NAME            <- here is defines the value of the parm
  required: true
  value: mysql
...SNIP...
- description: Password for the MySQL connection user.
  displayName: MySQL Connection Password
  from: '[a-zA-Z0-9]{16}'                <- this will generate a password using regex
  generate: expression
  name: MYSQL_PASSWORD
  required: true
```

Parameters

Parameters in templates have default values, but they are optional and can be replaced when processing a template:

```
[student@workstation template-fun]$ oc describe template mysql-ephemeral -n openshift

        Name:           MYSQL_PASSWORD
        Display Name:   MySQL Connection Password
        Description:    Password for the MySQL connection user.
        Required:       true
        Generated:      expression
        From:           [a-zA-Z0-9]{16}
```

An easy way to view the parameters is using the oc process command with --parameters:

```
[student@workstation template-fun]$ oc process --parameters mysql-ephemeral -n
openshift
NAME                  DESCRIPTION                                              GENERATOR
VALUE
MEMORY_LIMIT          Maximum amount of memory the container can use.
        512Mi
NAMESPACE             The OpenShift Namespace where the ImageStream resides.
        openshift
DATABASE_SERVICE_NAME   The name of the OpenShift Service exposed for the database.
        mysql
```

```
MYSQL_USER          Username for MySQL user that will be used for accessing the database.
expression      user[A-Z0-9]{3}
MYSQL_PASSWORD          Password for the MySQL connection user.
expression      [a-zA-Z0-9]{16}
MYSQL_ROOT_PASSWORD          Password for the MySQL root user.
expression      [a-zA-Z0-9]{16}
MYSQL_DATABASE            Name of the MySQL database accessed.
          sampledb
MYSQL_VERSION              Version of MySQL image to be used (5.7, or latest).
          5.7
```

Let's actually use this template to generate values to these parameters:

```
[student@workstation ~]$ oc get template mysql-ephemeral -n openshift -o yaml >
mysql-ephemeral.yaml
```

```
[student@workstation ~]$ oc process -f mysql-ephemeral.yaml -o yaml >
mysql-ephemeral-processed.yaml
```

```
[student@workstation template-fun]$ vim mysql-ephemeral-processed.yaml
```

```
  stringData:
          database-name: sampledb
          database-password: tqm8vlBxdxuLU7L0
          database-root-password: U2U20TmNGYQ5jxD4
          database-user: userQLD
```

Suppose we want to use this template, but override some parameters:

```
[student@workstation ~]$ oc process -f mysql-ephemeral.yaml  -p MYSQL_USER='dev' -p
MYSQL_PASSWORD='$P4SSD' -p MYSQL_DATABASE='bank' -o yaml >
mysql-ephemeral-processed.yaml
[student@workstation ~]$ vim mysql-ephemeral-processed.yaml
```

```
  stringData:
          database-name: bank
          database-password: $P4SSD
          database-root-password: fEjPviDan6DujROp
          database-user: dev
```

To create the actual resources

```
[student@workstation ~]$ **oc create -f mysqlprocessed.yaml**
secret/mysql created
service/mysql created
deploymentconfig.apps.openshift.io/mysql created

[student@workstation ~]$ **oc get secrets**
[student@workstation ~]$ **oc describe secrets mysql**

[student@workstation ~]$ **oc get all**

[student@workstation template-fun]$ **oc logs pod/mysql-1-h996k -f**

=> sourcing 20-validate-variables.sh ...
=> sourcing 25-validate-replication-variables.sh ...
=> sourcing 30-base-config.sh ...
---> 20:53:54   Processing basic MySQL configuration files ...
=> sourcing 60-replication-config.sh ...
=> sourcing 70-s2i-config.sh ...
---> 20:53:54   Processing additional arbitrary  MySQL configuration provided by s2i ...
=> sourcing 10-mysql57.cnf ...
=> sourcing 40-paas.cnf ...
=> sourcing 50-my-tuning.cnf ...
---> 20:53:54   Initializing database ...


[student@workstation template-fun]$ nohup **oc port-forward pod/mysql-1-h996k 3306:3306 &**
Forwarding from 127.0.0.1:3306 -> 3306
Forwarding from [::1]:3306 -> 3306

(open a 2nd terminal)

[root@workstation ~]# **mysql -u dev -p'$P4SSD' -h localhost --protocol tcp**

MySQL [(none)]> **show databases;**
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bank               |
+--------------------+
```

CLEANUP

```
[student@workstation template-fun]$ oc delete project template-fun
project.project.openshift.io "template-fun" deleted
```

Other ways to use templates to process and create them:

```
$ oc new-app --template=mysql-ephemeral
$ oc process openshift//mysql-ephemeral
$ oc process -f mysql-ephemeral-template.yaml
```

Are there best practices guides available for creating OCP templates ?

The community maintains this https://github.com/openshift/library which includes several templates as well as a link to this document https://docs.okd.io/latest/dev_guide/templates.html#writing-templates.

My recommendation is to start with an existing template, preferably one already in ocp4 (ie `oc get templates -n openshift`).

Templates themselves can be published to Openshift cluster so other devs can build an app from it.

```
[student@workstation deploy-multicontainer]$ oc create -f todo-template.yaml
[student@workstation deploy-multicontainer]$ oc create -f todo-template.yaml -n openshift
```

Templates vs. Helm charts

https://github.com/helm/helm/tree/master/docs/examples/nginx/templates

CONFIGURING PERSISTENT STORAGE FOR OPENSHIFT APPLICATIONS [NO LONGER POSSIBLE]

How do we provide persistent storage to an application ? How did we do it using podman ?

Ocp “pools” persistent storage as a cluster-wide resource. So, how can we avoid committing the same storage volume to multiple projects/applications ? With a reservation or ***claim***.

PersistentVolumeClaim (pvc) and PersistentVolume (pv)

- AWS Elastic Block Store (EBS)
- Fibre Channel
- HostPath
- iSCSI
- NFS
- VMWare vSphere

```
[student@workstation ~]$ oc get pv
Error from server (Forbidden): persistentvolumes is forbidden: User "rhn-support-ablum" cannot
list resource "persistentvolumes" in API group "" at the cluster scope
```

https://docs.openshift.com/container-platform/4.1/storage/persistent-storage/persistent-storage-nfs.html

Let's get an NFS server running:

```
[student@workstation ~]$ ssh root@services
Last login: Tue Jul 23 16:55:20 2019 from 172.25.250.250
[root@services ~]#
[root@services ~]#
[root@services ~]# cat /etc/exports
[root@services ~]# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.6 (Maipo)
```

```
#!/bin/bash

yum install nfs-utils rpcbind -y
useradd -u 5555 nfsuser
mkdir -p /opt/openshift
chown :5555 /opt/openshift
chmod 2770 /opt/openshift
```

```
ls -ld /opt/openshift
echo '/opt/openshift *(all_squash,anongid=5555,rw,sync)' > /etc/exports
cat /etc/exports
firewall-cmd --zone=public --add-port=2049/tcp --permanent
firewall-cmd --reload
firewall-cmd --list-all
systemctl start rpcbind
systemctl start nfs
systemctl enable nfs-server
exportfs -v
```

```
[root@services ~]# yum install nfs-utils rpcbind
[root@services ~]# useradd -u 5555 nfsuser
[root@services ~]# mkdir  -p /opt/openshift
[root@services ~]# chown :5555 /opt/openshift/
[root@services ~]# chmod 2770 /opt/openshift/
[root@services ~]# ls -ld /opt/openshift/
drwxrws---. 2 root nfsuser 6 Aug  9 14:41 /opt/openshift/
[root@services ~]# vi /etc/exports
/opt/openshift *(all_squash,anongid=5555,rw,sync)
[root@services ~]# firewall-cmd --zone=public --add-port=2049/tcp --permanent
success
[root@services ~]# firewall-cmd --reload
success
[root@services ~]# firewall-cmd --list-all
[root@services ~]# systemctl start rpcbind
[root@services ~]# systemctl start nfs
[root@services ~]# systemctl enable nfs-server
```

Test as a client:

```
[root@workstation ~]# mkdir /opt/data
[root@workstation ~]# mount -t nfs services.lab.example.com:/opt/openshift /opt/data
[root@workstation ~]# touch /opt/data/file1
[root@workstation ~]# ssh root@services ls -l /opt/openshift
Warning: Permanently added 'services,172.25.250.13' (ECDSA) to the list of known hosts.
total 0
-rw-r--r--. 1 nfsnobody nfsuser 0 Aug 18 21:45 file1
[root@workstation ~]# umount /opt/data/
```

Create a pv in ocp:

```
[student@workstation ~]$ lab openshift-resources start
[student@workstation ~]$ cat .kubeadmin
jSJuP-HbEee-gx4pL-LIer5
[student@workstation ~]$ oc login -u kubeadmin -p jSJuP-HbEee-gx4pL-LIer5

[student@workstation template-fun]$ oc new-project pv-fun
```

https://docs.openshift.com/container-platform/4.1/storage/persistent-storage/persistent-storage-nfs.html

```
[student@workstation template-fun]$ vi pv01.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv01
spec:
  capacity:
        storage: 5Gi
  accessModes:
  - ReadWriteOnce
  nfs:
        path: /opt/openshift
        server: 172.25.250.13
  persistentVolumeReclaimPolicy: Retain
```

```
[student@workstation ~]$ oc create -f pv01.yaml
persistentvolume/pv01 created

[student@workstation ~]$ oc get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM
STORAGECLASS   REASON   AGE
pv01   5Gi        RWO            Retain           Available           31s

[student@workstation ~]$ oc describe pv pv01
Name:          pv01
```

```
Labels:         <none>
Annotations:    <none>
Finalizers:     [kubernetes.io/pv-protection]
StorageClass:
Status:         Available
Claim:
Reclaim Policy:  Retain
Access Modes:          RWO
Capacity:       5Gi
Node Affinity:   <none>
Message:
Source:
        Type:   NFS (an NFS mount that lasts the lifetime of a pod)
        Server: 172.25.250.13
        Path:   /opt/openshift
        ReadOnly:  false
Events:         <none>
```

Now, let's try to use this storage:

```
[student@workstation ~]$ oc new-app
--docker-image=registry.access.redhat.com/rhscl/mysql-57-rhel7 -e
MYSQL_ROOT_PASSWORD=passwd
```

```
[student@workstation template-fun]$ vim nfs-claim.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
        - ReadWriteOnce
  resources:
        requests:
        storage: 5Gi
```

```
[student@workstation ~]$ oc get pv
```

```
[student@workstation template-fun]$ oc create -f nfs-claim.yaml
```

```
persistentvolumeclaim/nfs-claim1 created

[student@workstation ~]$ oc get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
pv01   5Gi        RWO            Retain           Bound  pv-fun/nfs-claim1                        23m
[student@workstation ~]$ oc get pvc
NAME          STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-claim1   Bound     pv01     5Gi        RWO                           14s

[student@workstation ~]$ oc edit deploymentconfig.apps.openshift.io/mysql-57-rhel7

  spec:
    containers:
    - env:
      - name: MYSQL_ROOT_PASSWORD
        value: passwd
      image:
registry.access.redhat.com/rhscl/mysql-57-rhel7@sha256:225ccc7a059a8e615ce3e5f4aa6fae5
07a9e03b4562dccf3a98b09ff69bcdeb7
      imagePullPolicy: Always
      name: mysql-57-rhel7
      ports:
      - containerPort: 3306
        protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: "/opt/data"
        name: mypv
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
    volumes:
    - name: mypv
      persistentVolumeClaim:
        claimName: nfs-claim1
```

```
[student@workstation ~]$ oc rsh pod/mysql-57-rhel7-3-rl4sp

sh-4.2$ df /opt/data
Filesystem                 1K-blocks       Used Available Use% Mounted on
172.25.250.13:/opt/openshift 104845312 5324544  99520768   6% /opt/data
sh-4.2$ touch /opt/data/test1
touch: cannot touch '/opt/data/test1': Permission denied
sh-4.2$ ls -ld /opt/data
drwxrws---. 3 root 5555 22 Aug 20 14:33 /opt/data

sh-4.2$ id
uid=1000410000 gid=0(root) groups=0(root),1000410000
```

```
[root@cluster-worker-1 ~]# journalctl -a | grep -i avc

Aug 23 16:05:18 cluster-worker-1 kernel: type=1400 audit(1566576318.390:8): avc:  denied  {
write } for  pid=38574 comm="touch" name="openshift" dev="0:465" ino=92276291
scontext=system_u:system_r:container_t:s0:c0,c21 tcontext=system_u:object_r:nfs_t:s0
tclass=dir permissive=0
```

```
[root@cluster-worker-1 ~]# setsebool -P virt_use_nfs 1
[root@workstation ~]# ssh core@worker0 sudo  setsebool -P virt_use_nfs 1
[root@workstation ~]# ssh core@master0 sudo  setsebool -P virt_use_nfs 1
```

Changes made to nodes could be handled by the Node Tuning Operator:
https://docs.openshift.com/container-platform/4.2/scalability_and_performance/using-node-tuning-operator.html

```
[student@workstation ~]$ oc delete pod/mysql-57-rhel7-2-z7vk4
pod "mysql-57-rhel7-2-z7vk4" deleted

sh-4.2$ touch /opt/data/test1
```

Other considerations::

Security Context Constraints (scc)

```
[student@workstation ~]$ oc adm policy add-scc-to-user anyuid -z default
```

Regular filesystem permissions (adding a group to the user running in the container):

```
[student@workstation ~]$ oc edit deploymentconfig.apps.openshift.io/mysql-57-rhel7

  spec:
    containers:
    - env:
      - name: MYSQL_ROOT_PASSWORD
        value: passwd
      image:
registry.access.redhat.com/rhscl/mysql-57-rhel7@sha256:225ccc7a059a8e615ce3e5f4aa6fae5
07a9e03b4562dccf3a98b09ff69bcdeb7
      imagePullPolicy: Always
      name: mysql-57-rhel7
      ports:
      - containerPort: 3306
        protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /opt/data
        name: mypv
    securityContext:
      supplementalGroups: [5555]
```

CHAPTER 8: TROUBLESHOOTING

How would one collect an application core from an application that crashes while running in a container ?

KCS indicates that the recommended approach is to create a PVC for the pod with the crashing application. Then, add a mount inside the pod to use the PV. The application would also need to be configured to dump to that location. The underlying container image might need a few more yum install to get symbols for some common libraries installed. Here are two examples from OCP3 that should work in OCP4:

https://access.redhat.com/solutions/3124061
https://access.redhat.com/solutions/3374631
https://developers.redhat.com/blog/2020/01/09/debugging-applications-within-red-hat-openshift-containers

I doubt you'd want to actually do any debugging (ie gdb) in the container. So, after the core is generated copying it off with scp or rsync would be what I'd suggest. I asked an SME in sbr-shift about this and here was his response:

""
Application cores have always been a problem, we really should be using a tool like FAF https://retrace.fedoraproject.org/faf/summary/ and ABRT to collect this information, review and analyze it.
""

How to check node module version using npm

```
[ablum@badger ~]$ npm view express@4.14.2
--registry="http://nexus-common.apps.na45.prod.nextcle.com/repository/npm-proxy"
[ablum@badger ~]$ npm view express@4.14.1
--registry="http://nexus-common.apps.na45.prod.nextcle.com/repository/npm-proxy"

express@4.14.1 | MIT | deps: 26 | versions: 264

[ablum@badger ~]$ npm view express versions
--registry="http://nexus-common.apps.na45.prod.nextcle.com/repository/npm-proxy"
```

This indicates that 4.17.1 is the latest available from this registry while 4.14.0 and 4.14.1 is the only available in 4.14.x.

Note this really just a proxy to the public npm registry: `npm view express --registry="https://registry.npmjs.org"`

After discussing with the developer, any 4.14.x version of express dependency will do. So, we can change:

```
[student@workstation DO180-apps]$ vi ~/DO180-apps/nodejs-helloworld/package.json

"dependencies": {
"express": "4.14.x"
}

[student@workstation DO180-apps]$ git commit -am "Fixed Express release"
...output omitted...
1 file changed, 1 insertion(+), 1 deletion(-)
[student@workstation DO180-apps]$ git push

[student@workstation DO180-apps]$ oc start-build bc/nodejs-hello

[student@workstation DO180-apps]$ oc logs -f bc/nodejs-hello

[student@workstation DO180-apps]$ oc get pods
```

How to use public nexus registry instead of the one in DO180

Sometimes we are having an issue with the nexus repo in the training environment:

```
$ npm view express
--registry="http://nexus-common.apps.ocp-na2.prod.nextcl.com/repository/npm-proxy"
npm ERR! Unexpected token < in JSON at position 0 while parsing near '<!DOCTYPE html
PUBLI...'

$ npm view express --registry="https://registry.npmjs.org"

express@4.17.1 | MIT | deps: 30 | versions: 264
Fast, unopinionated, minimalist web framework
```

```
[student@workstation DO180-apps]$ oc edit bc nodejs-hello

  strategy:
        sourceStrategy:
        env:
        - name: npm_config_registry
        value: https://registry.npmjs.org
        from:


[student@workstation DO180-apps]$ oc start-build bc/nodejs-hello
[student@workstation DO180-apps]$ oc logs bc/nodejs-hello

Successfully pushed
image-registry.openshift-image-registry.svc:5000/rhn-support-ablum-nodejs/nodejs-hello@sha2
56:7f19fae9e58909e0aaf918ba6b5b6f54598265d2514193da67258db741fb15ae
Push successful
```

Build was successful ! What about the deployment ?

```
[student@workstation nodejs-helloworld]$ oc get pods
NAME                          READY   STATUS              RESTARTS   AGE
nodejs-hello-1-build          0/1     Error               0          15m
nodejs-hello-2-build          0/1     Completed           0          114s
nodejs-hello-85b47f4f4c-58zgb   0/1     CrashLoopBackOff   3          80s
[student@workstation nodejs-helloworld]$ oc logs nodejs-hello-85b47f4f4c-58zgb
Environment:
   DEV_MODE=false
   NODE_ENV=production
   DEBUG_PORT=5858
Launching via npm...
npm info it worked if it ends with ok
npm info using npm@6.14.8
npm info using node@v12.19.1
npm ERR! missing script: start
npm timing npm Completed in 82ms

npm ERR! A complete log of this run can be found in:
npm ERR!     /opt/app-root/src/.npm/_logs/2021-07-02T14_40_07_931Z-debug.log
```

```
[student@workstation DO180-apps]$ vi ~/DO180-apps/nodejs-helloworld/package.json

"description": "Hello World!",
"main": "app.js",
  "scripts": {"start": "node app.js"},



[student@workstation DO180-apps]$ git commit -am "Added start up script"
...output omitted...
1 file changed, 3 insertions(+)
[student@workstation DO180-apps]$ git push

[student@workstation DO180-apps]$ oc start-build bc/nodejs-hello

[student@workstation DO180-apps]$  oc logs dc/nodejs-hello

> nodejs-helloworld@1.0.0 start /opt/app-root/src
> node app.js

Example app listening on port 8080!

[student@workstation DO180-apps]$  oc get service
NAME          TYPE          CLUSTER-IP  EXTERNAL-IP  PORT(S)    AGE
nodejs-hello  ClusterIP  172.30.208.10  <none>   8080/TCP  48m
[student@workstation nodejs-helloworld]$ oc expose service nodejs-hello
route.route.openshift.io/nodejs-hello exposed
[student@workstation nodejs-helloworld]$ oc get routes
NAME          HOST/PORT                                          PATH  SERVICES    PORT
        TERMINATION  WILDCARD
nodejs-hello  nodejs-hello-rhn-support-ablum-nodejs.apps.na45.prod.nextcle.com
nodejs-hello  8080-tcp                None
[student@workstation nodejs-helloworld]$ curl
nodejs-hello-rhn-support-ablum-nodejs.apps.na45.prod.nextcle.com
Hello World!

[student@workstation nodejs-helloworld]$
[student@workstation nodejs-helloworld]$ cd  ~
[student@workstation ~]$ lab troubleshoot-s2i finish
```

EXTRA Troubleshooting TIPS and TRICKS

Port-forward

For applications that are serving up content (listening on certain ports) the **podman port-forward** or even the **oc port-forward** can be used to verify the application endpoint directly independent of service/routes.

Another good use for port-forward is to gather additional diagnostics from an application. Some applications can be configured to allow debugging through a network port (ie java vi Java Debug Wire Protocol or JDWP). Once the app is configured (via standalone.conf) to enable remote debugging, the **oc port-forward** command can be used to establish a connection to the app running in the container.

```
[student@workstation ~]$ oc new-project ablum-mytest

[student@workstation ~]$ oc new-app --docker-image=openshift/hello-openshift --name
hello

[student@workstation ~]$ oc port-forward pod/hello-7764b7f5f8-kpmzr 9091:8888
Forwarding from 127.0.0.1:9091 -> 8888
Forwarding from [::1]:9091 -> 8888
^C[student@workstation ~]$ nohup oc port-forward pod/hello-7764b7f5f8-kpmzr 9091:8888
&
[1] 9501
nohup: ignoring input and appending output to ‘nohup.out’

[student@workstation ~]$ curl localhost:9091
Hello OpenShift!
[student@workstation ~]$
```

Maybe we can use a packet analyzer….

```
[student@workstation ~]$ sudo tcpdump -i any -s0 port 9091

[student@workstation ~]$ fg
nohup oc port-forward pod/hello-7764b7f5f8-kpmzr 9091:8888
^C[student@workstation ~]$
```

Logs and Events

Check the logs !

```
# podman logs
# cc logs
```

```
[student@workstation ~]$ oc logs pod/hello-7764b7f5f8-kpmzr
serving on 8888
serving on 8080
Servicing request.
Servicing request.
Servicing request.
```

```
$ oc get events
$ oc describe
```

```
[student@workstation ~]$ oc describe pod/hello-7764b7f5f8-kpmzr

Events:
  Type  Reason          Age     From                         Message
  ----  ------          ----    ----                         -------
  Normal  Scheduled             <unknown>  default-scheduler                    Successfully
assigned ablum-mytest/hello-7764b7f5f8-kpmzr to na45-mh9qn-worker-l886x
  Normal  AddedInterface  10m          multus                       Add eth0 [10.129.3.248/23]
  Normal  Pulled        10m     kubelet, na45-mh9qn-worker-l886x  Container image
"openshift/hello-openshift@sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3e7c9f556
c1ba5baf47e2e" already present on machine
  Normal  Created       10m     kubelet, na45-mh9qn-worker-l886x  Created container hello
  Normal  Started       10m     kubelet, na45-mh9qn-worker-l886x  Started container hello
```

Access running containers

```
$ podman exec
$ oc rsh
$ oc exec
```

```
[student@workstation ~]$ oc exec -h


Examples:
  # Get output from running 'date' command from pod mypod, using the first container by default
  oc exec mypod -- date

  # Get output from running 'date' command in ruby-container from pod mypod
  oc exec mypod -c ruby-container -- date

  # Switch to raw terminal mode, sends stdin to 'bash' in ruby-container from pod mypod
  # and sends stdout/stderr from 'bash' back to the client
  oc exec mypod -c ruby-container -i -t -- bash -il

[student@workstation ~]$ oc exec -it hello-7764b7f5f8-kpmzr /bin/bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use
kubectl kubectl exec [POD] -- [COMMAND] instead.
ERRO[0000] exec failed: container_linux.go:348: starting container process caused "exec:
\"/bin/bash\": stat /bin/bash: no such file or directory"
exec failed: container_linux.go:348: starting container process caused "exec: \"/bin/bash\": stat
/bin/bash: no such file or directory"
command terminated with exit code 1

[student@workstation ~]$ oc rsh hello-7764b7f5f8-kpmzr
ERRO[0000] exec failed: container_linux.go:348: starting container process caused "exec:
\"/bin/sh\": stat /bin/sh: no such file or directory"
exec failed: container_linux.go:348: starting container process caused "exec: \"/bin/sh\": stat
/bin/sh: no such file or directory"
command terminated with exit code 1
```

What to do when the image used by the running container does not have a shell built in it.

Check out hello-openshift for example
https://github.com/openshift/origin/blob/master/examples/hello-openshift/Dockerfile :

Here, you will need to use alternative troubleshooting like:

1.) Build a new troubleshooting image that has additional tools (like an interactive shell)
2.) Study the application logs
3.) Use port-forward to redirect application traffic to a local debugging program
4.) Create a debug pod:

```
[student@workstation ~]$ oc debug deployment.apps/hello
Starting pod/hello-debug ...

Removing debug pod ...
error: container create failed: time="2020-10-02T11:16:20Z" level=error
msg="container_linux.go:348: starting container process caused \"exec: \\\"/bin/sh\\\": stat
/bin/sh: no such file or directory\""
container_linux.go:348: starting container process caused "exec: \"/bin/sh\": stat /bin/sh: no such
file or directory"

[student@workstation ~]$ oc debug --image=registry.access.redhat.com/ubi8:latest
deployment.apps/hello

sh-4.4$ id
uid=1002450000(1002450000) gid=0(root) groups=0(root),1002450000

sh-4.4$ env | grep HELLO
HELLO_SERVICE_PORT=8080
HELLO_SERVICE_PORT_8888_TCP=8888
HELLO_PORT_8888_TCP=tcp://172.30.89.146:8888
HELLO_PORT_8080_TCP_PROTO=tcp
HELLO_SERVICE_HOST=172.30.89.146
HELLO_PORT_8888_TCP_PORT=8888
HELLO_PORT_8080_TCP_ADDR=172.30.89.146
HELLO_PORT=tcp://172.30.89.146:8080
HELLO_PORT_8888_TCP_PROTO=tcp
HELLO_SERVICE_PORT_8080_TCP=8080
HELLO_PORT_8080_TCP=tcp://172.30.89.146:8080
HELLO_PORT_8888_TCP_ADDR=172.30.89.146
HELLO_PORT_8080_TCP_PORT=8080

sh-4.4$ curl $HELLO_SERVICE_HOST:$HELLO_SERVICE_PORT
Hello OpenShift!
sh-4.4$ exit
exit

Removing debug pod ...
```

Copying files in and out of a container

```
# podman cp
# oc rsync -h
```

Watch out though, there are some caveats:

https://docs.openshift.com/container-platform/4.5/nodes/containers/nodes-containers-copying-files.html

```
[student@workstation ~]$ mkdir data
[student@workstation ~]$ echo hello > data/testfile

[student@workstation ~]$ oc rsync data hello-7764b7f5f8-kpmzr:/
WARNING: cannot use rsync: rsync not available in container
WARNING: cannot use tar: tar not available in container
error: No available strategies to copy.
```