Red Hat Security: Securing Containers & OpenShift (DO425)

Last update: Sun Dec 29 12:46:56 UTC 2019 by @luckylittle

Objectives

1. Understand, identify, and work with containerization features
   (a) Deploy a preconfigured application and identify crucial features such as namespaces, SELinux labels, and cgroups
   (b) Deploy a preconfigured application with security context constraint capabilities and view the application's capability set
   (c) Configure security context constraints
2. Use trusted registries
   (a) Load images into a registry
   (b) Query images in a registry
3. Work with trusted container images
   (a) Identify a trusted container image
   (b) Sign images
   (c) View signed images
   (d) Scan images
   (e) Load signed images into a registry
4. Build secure container images
   (a) Perform simple S2I builds
   (b) Implement S2I build hooks
   (c) Automate builds using Jenkins
   (d) Automate scanning and code validations as part of the build process
5. Control access to OpenShift Container Platform clusters
   (a) Configure users with different permission levels, access, and bindings
   (b) Configure OpenShift Container Platform to use Red Hat Identity Management services (IdM) for authentication
   (c) Query users and groups in IdM
   (d) Log into OpenShift Container Platform using an IdM managed account
6. Configure single sign-on (SSO)
   (a) Install SSO authentication

(b) Configure OpenShift Container Platform to use SSO

(c) Integrate web applications with SSO

7. Automate policy-based deployments

(a) Configure policies to control the use of images and registries

(b) Use secrets to provide access to external registries

(c) Automatically pull and use images from a registry

(d) Use triggers to verify that automated deployments work

8. Manage orchestration

(a) Restrict nodes on which containers may run

(b) Use quotas to limit resource utilization

(c) Use secrets to automate access to resources

9. Configure network isolation

(a) Create software-defined networks (SDN)

(b) Associate containers and projects with SDNs

10. Configure and manage secure container storage

(a) Configure and secure file-based container storage

(b) Configure and secure block-based container storage

1. Describing Host Security Technologies

INTRODUCING THE RHEL AND CRI-O CONTAINER TOOLS

- invoke the `crictl` locally from an OpenShift master or node, not remotely like `oc`
- `crictl` = tool to interface with the CRI-O container engine from Kubernetes

```
NAME:
  crictl - client for CRI
USAGE:
  crictl [global options] command [command options] [arguments...]
VERSION:
  unknown
COMMANDS:
  attach        Attach to a running container
  create        Create a new container
  exec          Run command in a running container
  version       Display runtime version information
  images        List images
  inspect       Display the status of one or more containers
  inspecti      Return the status of one or more images
  inspectp      Display the status of one or more pods
  logs          Fetch the logs of a container
```

```
  port-forward  Forward local port to a pod
  ps            List containers
  pull          Pull an image from registry
  runp          Run a new pod
  rm            Remove one or more containers
  rmi           Remove one or more images
  rmp           Remove one or more pods
  pods          List pods
  start         Start one or more created containers
  info          Display information of the container runtime
  stop          Stop one or more running containers
  stopp         Stop one or more running pods
  update        Update one or more running containers
  config        Get and set crictl options
  stats         List container(s) resource usage statistics
  completion    Output bash shell completion code
  help, h       Show a list of commands or help for one command
GLOBAL OPTIONS:
  --config value, -c value            Location of the client config file (default: "/etc/crictl.yaml")
      [$CRI_CONFIG_FILE]
  --debug, -D                         Enable debug mode
  --image-endpoint value, -i value    Endpoint of CRI image manager service [$IMAGE_SERVICE_ENDPOINT]
  --runtime-endpoint value, -r value  Endpoint of CRI container runtime service (defaul:
      "unix:///var/run/dockershim.sock") [$CONTAINER_RUNTIME_ENDPOINT]
  --timeout value, -t value           Timeout of connecting to the server (default: 10s)
  --help, -h                          show help
  --version, -v                       print the version
```

- `skopeo` = tool to manage container images stored in the local file system and in container registries

```
NAME:
  skopeo - Various operations with container images and container image registries
USAGE:
  skopeo [global options] command [command options] [arguments...]
VERSION:
  0.1.32
COMMANDS:
  copy                    Copy an IMAGE-NAME from one location to another
  inspect                 Inspect image IMAGE-NAME
  delete                  Delete image IMAGE-NAME
  manifest-digest         Compute a manifest digest of a file
  standalone-sign         Create a signature using local files
  standalone-verify       Verify a signature using local files
GLOBAL OPTIONS:
  --debug                 enable debug output
  --policy value          Path to a trust policy file
  --insecure-policy       run the tool without any policy check
  --registries.d DIR      use registry configuration files in DIR (e.g. for container signature storage)
  --override-arch ARCH    use ARCH instead of the architecture of the machine for choosing images
  --override-os OS        use OS instead of the running OS for choosing images
  --command-timeout value  timeout for the command execution (default: 0s)
  --help, -h              show help
  --version, -v           print the version
```

- `podman` = tool to start and manage standalone containers on OCI-compliant container engines (`podman build` = `buildah`)

```
NAME:
  podman - manage pods and images
USAGE:
  podman [global options] command [command options] [arguments...]
VERSION:
```

```
  1.0.5
COMMANDS:
  attach           Attach to a running container
  commit           Create new image based on the changed container
  container        Manage Containers
  build            Build an image using instructions from Dockerfiles
  create           Create but do not start a container
  diff             Inspect changes on container's file systems
  exec             Run a process in a running container
  export           Export container's filesystem contents as a tar archive
  history          Show history of a specified image
  image            Manage images
  images           List images in local storage
  import           Import a tarball to create a filesystem image
  info             Display podman system information
  inspect          Displays the configuration of a container or image
  kill             Kill one or more running containers with a specific signal
  load             Load an image from docker archive
  login            Login to a container registry
  logout           Logout of a container registry
  logs             Fetch the logs of a container
  mount            Mount a working container's root filesystem
  pause            Pauses all the processes in one or more containers
  list, ls, ps     List containers
  pod              Manage pods
  port             List port mappings or a specific mapping for the container
  pull             Pull an image from a registry
  push             Push an image to a specified destination
  restart          Restart one or more containers
  rm               Remove one or more containers
  rmi              Removes one or more images from local storage
  run              Run a command in a new container
  save             Save image to an archive
  search           Search registry for image
  start            Start one or more containers
  stats            Display percentage of CPU, memory, network I/O, block I/O and PIDs for one or more
      containers
  stop             Stop one or more containers
  tag              Add an additional name to a local image
  top              Display the running processes of a container
  umount, unmount  Unmounts working container's root filesystem
  unpause          Unpause the processes in one or more containers
  version          Display the Podman Version Information
  volume           Manage volumes
  wait             Block on one or more containers
  help, h          Shows a list of commands or help for one command
GLOBAL OPTIONS:
  --cgroup-manager value            cgroup manager to use (cgroupfs or systemd, default systemd)
  --cni-config-dir value            path of the configuration directory for CNI networks
  --conmon value                    path of the conmon binary
  --cpu-profile value               path for the cpu profiling results
  --hooks-dir value                 set the OCI hooks directory path (may be set multiple times)
  --log-level value                 log messages above specified level: debug, info, warn, error (default),
      fatal or panic (default: "error")
  --namespace value                 set the libpod namespace, used to create separate views of the
      containers and pods on the system
  --root value                      path to the root directory in which data, including images, is stored
  --tmpdir value                    path to the tmp directory
  --runroot value                   path to the 'run directory' where all state information is stored
  --runtime value                   path to the OCI-compatible binary used to run containers, default is
      /usr/bin/runc
```

```
  --storage-driver value, -s value  select which storage driver is used to manage storage of images and
     containers (default is overlay)
  --storage-opt value               used to pass an option to the storage driver
  --syslog                          output logging information to syslog as well as the console
  --help, -h                        show help
  --version, -v                     print the version
```

- `buildah` = tool to build container images

```
NAME:
  buildah - an image builder
USAGE:
  buildah [global options] command [command options] [arguments...]
VERSION:
  1.5 (image-spec 1.0.0, runtime-spec 1.0.0)
COMMANDS:
  add                          Add content to the container
  build-using-dockerfile, bud  Build an image using instructions in a Dockerfile
  commit                       Create an image from a working container
  config                       Update image configuration settings
  containers                   List working containers and their base images
  copy                         Copy content into the container
  from                         Create a working container based on an image
  images                       List images in local storage
  inspect                      Inspects the configuration of a container or image
  mount                        Mount a working container's root filesystem
  pull                         Pull an image from the specified location
  push                         Push an image to a specified destination
  rename                       Rename a container
  rm, delete                   Remove one or more working containers
  rmi                          removes one or more images from local storage
  run                          Run a command inside of the container
  tag                          Add an additional name to a local image
  umount, unmount              Unmounts the root file system on the specified working containers
  unshare                      Run a command in a modified user namespace
  version                      Display the Buildah Version Information
  help, h                      Shows a list of commands or help for one command
GLOBAL OPTIONS:
  --debug                              print debugging information
  --registries-conf value              path to registries.conf file (not usually used)
     [$REGISTRIES_CONFIG_PATH]
  --registries-conf-dir value          path to registries.conf.d directory (not usually used)
  --root value                         storage root dir (default:
     "/home/lmaly/.local/share/containers/storage")
  --runroot value                      storage state dir (default: "/run/user/104536")
  --storage-driver value               storage driver (default: "overlay")
  --storage-opt value                  storage driver option (default:
     "overlay.override_kernel_check=true")
  --userns-uid-map ctrID:hostID:length  default ctrID:hostID:length UID mapping to use
  --userns-gid-map ctrID:hostID:length  default ctrID:hostID:length GID mapping to use
  --help, -h                           show help
  --version, -v                        print the version
```

INSPECTING THE LINUX NAMESPACES

- `unshare` = command to create new namespaces

```
Usage:
  unshare [options] [<program> [<argument>...]]
Run a program with some namespaces unshared from the parent.
Options:
```

```
  -m, --mount[=<file>]          unshare mounts namespace
  -u, --uts[=<file>]            unshare UTS namespace (hostname etc)
  -i, --ipc[=<file>]            unshare System V IPC namespace
  -n, --net[=<file>]            unshare network namespace
  -p, --pid[=<file>]            unshare pid namespace
  -U, --user[=<file>]           unshare user namespace
  -C, --cgroup[=<file>]         unshare cgroup namespace
  -f, --fork                    fork before launching <program>
      --kill-child[=<signame>]  when dying, kill the forked child (implies --fork); defaults to SIGKILL
      --mount-proc[=<dir>]      mount proc filesystem first (implies --mount)
  -r, --map-root-user           map current user to root (implies --user)
      --propagation slave|shared|private|unchanged
                                modify mount propagation in mount namespace
  -s, --setgroups allow|deny    control the setgroups syscall in user namespaces
  -h, --help                    display this help
  -V, --version                 display version
```

```
unshare --uts --net /bin/sh                                          # starts a shell in new UTS and
    network namespaces
unshare --ipc --uts --net --mount sleep 1h &                         # starts a sleep command in the
    background in four namespaces: IPC, UTS, network, and mount, returns PID e.g. 8672
```

- `lsns` = command to list all the namespaces on the system

```
Usage:
  lsns [options] [<namespace>]
List system namespaces.
Options:
  -J, --json             use JSON output format
  -l, --list             use list format output
  -n, --noheadings       don't print headings
  -o, --output <list>    define which output columns to use
  -p, --task <pid>       print process namespaces
  -r, --raw              use the raw output format
  -u, --notruncate       don't truncate text in columns
  -W, --nowrap           don't use multi-line representation
  -t, --type <name>      namespace type (mnt, net, ipc, user, pid, uts, cgroup)
  -h, --help             display this help
  -V, --version          display version
Available output columns:
        NS  namespace identifier (inode number)
      TYPE  kind of namespace
      PATH  path to the namespace
    NPROCS  number of processes in the namespace
       PID  lowest PID in the namespace
      PPID  PPID of the PID
   COMMAND  command line of the PID
       UID  UID of the PID
      USER  username of the PID
   NETNSID  namespace ID as used by network subsystem
      NSFS  nsfs mountpoint (usually used network subsystem)
```

```
lsns -p 8672                                                         # similar to ls -l /proc/8672/ns
```

- `nsenter` = command to run a program in an existing namespace, if you do not provide a command as argument, `nsenter` runs `/bin/bash`

```
Usage:
  nsenter [options] [<program> [<argument>...]]
Run a program with namespaces of other processes.
Options:
```

```
  -a, --all              enter all namespaces
  -t, --target <pid>     target process to get namespaces from
  -m, --mount[=<file>]   enter mount namespace
  -u, --uts[=<file>]     enter UTS namespace (hostname etc)
  -i, --ipc[=<file>]     enter System V IPC namespace
  -n, --net[=<file>]     enter network namespace
  -p, --pid[=<file>]     enter pid namespace
  -C, --cgroup[=<file>]  enter cgroup namespace
  -U, --user[=<file>]    enter user namespace
  -S, --setuid <uid>     set uid in entered namespace
  -G, --setgid <gid>     set gid in entered namespace
      --preserve-credentials do not touch uids or gids
  -r, --root[=<dir>]     set the root directory
  -w, --wd[=<dir>]       set the working directory
  -F, --no-fork          do not fork before exec'ing <program>
  -Z, --follow-context   set SELinux context according to --target PID
  -h, --help             display this help
  -V, --version          display version
```

```
nsenter -t 8672 --net ip addr                                    # lists the network devices and
    addresses in a network namespace
nsenter -t 8672 --ipc --uts --net --mount                        # execute a shell in multiple
    namespaces at the same time
```

- `runc` = Open Container Initiative runtime

```
USAGE:
  runc [global options] command [command options] [arguments...]
VERSION:
  spec: 1.0.1-dev
COMMANDS:
  checkpoint  checkpoint a running container
  create      create a container
  delete      delete any resources held by the container often used with detached container
  events      display container events such as OOM notifications, cpu, memory, and IO usage statistics
  exec        execute new process inside the container
  init        initialize the namespaces and launch the process (do not call it outside of runc)
  kill        kill sends the specified signal (default: SIGTERM) to the container's init process
  list        lists containers started by runc with the given root
  pause       pause suspends all processes inside the container
  ps          ps displays the processes running inside a container
  restore     restore a container from a previous checkpoint
  resume      resumes all processes that have been previously paused
  run         create and run a container
  spec        create a new specification file
  start       executes the user defined process in a created container
  state       output the state of a container
  update      update container resource constraints
  help, h     Shows a list of commands or help for one command
GLOBAL OPTIONS:
  --debug             enable debug output for logging
  --log value         set the log file path where internal debug information is written (default:
      "/dev/null")
  --log-format value  set the format used by logs ('text' (default), or 'json') (default: "text")
  --root value        root directory for storage of container state (this should be located in tmpfs)
      (default: "/run/user/1000/runc")
  --criu value        path to the criu binary used for checkpoint and restore (default: "criu")
  --systemd-cgroup    enable systemd cgroup support, expects cgroupsPath to be of form "slice:prefix:name"
      for e.g. "system.slice:runc:434234"
  --rootless value    ignore cgroup permission errors ('true', 'false', or 'auto') (default: "auto")
  --help, -h          show help
  --version, -v       print the version
```

When you create a pod, OpenShift runs the 'pod' process to create these namespaces and place the container processes in them. This means that all containers in a pod share the same network, the same System V IPC objects, and have the same host name.

How to find PID in OpenShift?

```
oc describe pod myweb-1-bmmwq | grep 'Container ID'                    # 7084...383d
runc state 7084...383d | grep "pid"                                    # 8997
nsenter -t 8997 -p -r ps -ef                                           # To use the ps command in a
    container namespace, also specify the -r or --root option to get the same file systems as the
    container. This is because the ps command uses the /proc file system to list the processes, and it
    needs to use the one from the container hierarchy and not the one from the host system.
```

SECURING CONTAINERS WITH SELINUX

- With OpenShift, containers processes always get the `container_t` context type when they start, and the files and directories the containers need to access on the host system gets the `container_file_t` context type.

- MCS (Multi Category Security) is a SELinux feature that solves issue of containers protect themselves from other container by taking advantage of the level part of the context: `system_u:system_r:container_t:s0:c4,c9` (s0=sensitives, c4,c9=categories). Sensitivity is not used, but categories are. Category values are between `c0` and `c1023` each. When a container starts, the system assigns two random categories to the processes in the container. But OpenShift behaves differently when assigning the SELinux categories - OpenShift allocates the two random categories at the project level. Therefore, and by default, all pods and containers in a project get the same category pair. This is useful when multiple containers in the project need to access the same shared volume.

MANAGING RESOURCES WITH CGROUPS

```
cat /proc/1938/cgroup                                                  # Systemd automatically organizes its
    services in control group hierarchies. It places all services under the /system.slice control groups
ls -l /sys/fs/cgroup/memory/system.slice/rsyslog.service/              # this is the standard structure
systemd-cgls                                                           # list all the control groups on the
    system
tree -d /sys/fs/cgroup/memory/kubepods.slice/                          # this is OpenShift/Kube specific
    structure
```

LAB 1.1

```
oc login -u developer -p redhat https://master.lab.example.com
oc new-project host-isolation
oc new-app -f ~/DO425/labs/host-isolation/host-isolation.yml
oc describe pod isol | grep 'Node:\|Container ID:'                     # returns node1 and IDs
ssh root@node1.lab.example.com
runc state <ID1,2> | grep pid                                          # returns PID1, PID2
lsns -p <PID1,2>                                                       # lists the namespaces associated
    with the two containers, compare 'uts', 'ipc', 'net' numbers - they are the same for both
nsenter -t <PID1> -n ip addr
nsenter -t <PID1> -p -r ps -ef                                         # container only sees its processes,
    and the PIDs start at 1 inside the container
nsenter -t <PID1> -u hostname                                          # isol
nsenter -t <PID1> -u hostname mynewname                                # change hostname to 'mynewname'
nsenter -t <PID2> -u hostname                                          # mynewname, because the two
    containers share the same UTS namespace
ps -Z -p <PID1>,<PID2>                                                 # both have the same categories -
    e.g. c5,c11
runc state <ID1> | grep rootfs                                         #
    /var/lib/containers/storage/overlay/<ID>/merged/
ls -Zd </var/lib/containers/storage/overlay...>                        # same categories c5,c11
oc get pod isol -o yaml | grep uid                                     # on the workstation get UID of the
    pod, e.g. 8a8285dc-e350-11e8-88a6-52540000fa0a
oc get pod isol -o yaml | grep containerID                             # also look for a containerID of the
    container you want (e.g. httpd)
cd /sys/fs/cgroup/memory/kubepods.slice/kubepods-burstable.slice/      # on the node
```

```
cd kubepods-burstable-pod<UID>.slice/                                    # or cd *8a8285dc*
cd crio-<containerID>.scope/
grep <PID1 or 2> tasks                                                   # returns PID1 or 2
cat memory.limit_in_bytes                                                # e.g. 67108864
cat memory.usage_in_bytes                                                # e.g. 9691136
```

LISTING AVAILABLE LINUX CAPABILITIES & LIMITING THEM

```
capsh --print                                                            # capsh (Capabilities Shell) is a
    utility for viewing the available capabilities on a system for user running it
cat /proc/<PID>/status | grep Cap                                        # capabilities for process: CaCapInh
    (inheritable), CapPrm (permitted), CapEff (permission checks), CapBnd (bounding set), CapAmb (ambient)
capsh --decode=0000003fffffffff                                          # decode HEX
capsh --print -- -c "ping -c 1 localhost"                                # contains cap_net_raw among other
    things
capsh --drop=cap_net_raw --print -- -c "ping -c 1 localhost"             # drops the cap_net_raw capability
    which makes the ping fail
```

MANAGING CAPABILITIES IN CONTAINERS

- Podman possess a set of two options for managing capabilities: `--cap-add` and `--cap-drop`
- Default common capabilities by podman: `cap_chown`, `cap_mknod`, `cap_dac_override`, `cap_audit_write`, `cap_setfcap`, `cap_fsetid`

```
podman run --rm -it --name=capabilities rhel7/7.5 capsh --print
podman run --rm -it --cap-add=sys_time --name=test_caps rhel7/7.5 capsh --print # adds change system time
    capability
```

Avoid `CAP_SYS_ADMIN`, which is too broad.

INTRODUCING SECURE COMPUTING MODE `seccomp`

- If a process attempts to perform a system call that it is not allowed to performed, the process is terminated according the policy that is in place
- Two modes:
- A) allows a process to make only four system calls: `read()`, `write()`, `_exit()`, and `sigreturn()`. With this seccomp mode enabled, processes cannot fork new threads, nor monitor network activity.
- B) seccomp-bpf Kernel extension allows generic system call filtering. For example, you can define a rule that allows a process to only access certain files. seccomp allows you to define a profile that contains a set of filters, which are applied to every system call that submitted from a process to the kernel.

RESTRICTING PROCESSES WITH `seccomp`

- To enable seccomp protection, a parent process sets a profile right before forking a child process
- Podman allows you to use the `--security-opt` option to attach a security profile to your container
- Two annotations:
- A) `seccomp.security.alpha.kubernetes.io/pod`
- B) `container.seccomp.security.alpha.kubernetes.io/<container_name>`

An example of the `custom_policy.json`:

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "architectures": [
    "SCMP_ARCH_X86_64",
```

```
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "names": [
        "chroot"
      ],
      "action": "SCMP_ACT_ERRNO",
      "args": [],
      "comment": "",
      "includes": {
        "caps": [
          "CAP_SYS_CHROOT"
        ]
      },
      "excludes": {}
    }
  ]
}
```

Attach this policy to a container using Podman and test it:

```
sudo podman run --rm -it --security-opt seccomp=custom_policy.json --name chroot rhscl/httpd-24-rhel7
    chroot / # not permitted
grep Seccomp /proc/10/status                                    # inspect the status file of any
    child process running inside the container
```

Limitations:

1. Filtering policies apply to the entire pod or container, and not only to the application running inside a container. Consider also the system calls that the container runtime makes when starting the container.

2. OpenShift does not support yet policy precedence. If developer defines custom profile for their containers, it overrides the default profile by OpenShift.

Identifying System Calls that Processes Invoke:

1. Creating a container with the `SYS_PTRACE` capability. This capability allows a container to trace arbitrary processes using ptrace

2. Invoking the strace command from inside the container.

3. Locating the commands that are invoked.

4. Updating the security policy to include all relevant system calls.

5. Instantiating a new container with the updated security profile.

LAB 1.2

```
oc login -u developer -p redhat https://master.lab.example.com
oc new-project host-privilege
podman search registry.lab.example.com/http
oc new-app -f ~/DO425/labs/host-privilege/host-privilege.yml
oc get pods                                                     # ensure httpd pod is running
oc rsh httpd bash                                               # open bash on the pod
   capsh --print                                                # list current capabilities,
       cap_chown is listed
   touch test-cap
   chown nobody test-cap                                        # it works
   exit
```

```
oc describe pod httpd | grep Node:                                    # retrieve the node where the pod is
    running
oc describe pod httpd | grep cri-o                                    # get the container ID
ssh root@node1                                                        # connect to the node on which your
    container is running
runc state <containerID> | grep pid                                   # inspect the container, get PID
grep Cap /proc/<PID>/status | grep CapInh                             # check the capabilities
capsh --decode=00000000000425fb                                       # obtain the list of capabilities of
    the CapInh - inherited set, cap_chown is present
exit
oc delete pod httpd
cd ~/DO425/labs/host-privilege/
cat host-privilege-drop.yml
  containers:
  - image: rhscl/httpd-24-rhel7:latest
    name: httpd
    securityContext:
      capabilities:
        drop:
        - CHOWN
oc new-app -f host-privilege-drop.yml                                 # recreate the application with this
    template
oc rsh httpd bash
  capsh --print                                                       # cap_chown is not present
  touch test-cap
  chown nobody test-cap                                               # operation not permitted
  exit
oc delete pod httpd
sudo podman run -dit --name dev-seccomp --hostname seccomp-unrestricted
    registry.lab.example.com/rhel7/rhel-tools # create a container dev-seccomp
sudo podman exec dev-seccomp bash                                     # log in to the container
  ps                                                                  # bash PID is 13
  grep Seccomp /proc/13/status                                        # Seccomp: 2 = enabled & filtering
      mode
  chmod 400 /etc/hosts                                                # works
  chown nobody /etc/hosts                                             # works
  exit
cat ~/DO425/labs/host-privilege/seccomp-profile.json                  # fchmodat & fchoumat are
    SCMP_ACT_ERRNO
sudo podman run -dit --name dev-seccomp-restricted --hostname seccomp-restricted --security-opt
    seccomp=seccomp-profile.json registry.lab.example.com/rhel7/rhel-tools:latest
                                                        # second container called dev-seccomp-restricted
    and attach it the security profile by using the --security-opt
sudo podman exec dev-seccomp-restricted bash                          # log in to the container
  chmod 400 /etc/hosts                                                # operation not permitted
  chown nobody /etc/hosts                                             # operation not permitted
  exit
sudo podman stop dev-seccomp && sudo podman rm dev-seccomp            # cleanup
sudo podman stop dev-seccomp-restricted && sudo podman rm dev-seccomp-restricted
```

2. Establishing Trusted Container Images

Red Hat OpenShift Platform v3.11 points to the new `registry.redhat.io` registry by default

```
podman login registry.redhat.io                                       # Podman stores your encrypted
    credentials in a file called /run/user/<UID>/containers/auth.json
skopeo inspect --creds developer1:MyS3cret! docker://registry.redhat.io/rhscl/postgresql-96-rhel7 # Skopeo
    uses the auth.json file from podman login, but you can alternatively do --creds (or
    --src-creds/--dest-creds for copy)
podman logout registry.redhat.io
podman logout --all
```

QUAY

```
podman login quay.apps.lab.example.com
podman pull quay.apps.lab.example.com/developer2/rhel-tools:latest    # or have the rhel-tools public and
    then you don't need auth
```

Configuring Jenkins to support Quay integration

To use Quay as the container image registry, some requirements must be addressed: The Quay repository must use a valid SSL certificate to communicate with OpenShift and Jenkins. If you are using self-signed certificates, each node from OpenShift must have the self-signed certificate in the Quay's certificate directory (`/etc/docker/certs.d/<quay-URI>`). Furthermore, the Jenkins slave container must have the certificate to sign the container image as well as the skopeo command line to push changes to the registry.

LAB 2.1

```
oc login -u admin -p redhat https://master.lab.example.com
oc project quay-enterprise                                          # groups all resources of Quay
oc get pods                                                         # Quay does not support running the
    database on OpenShift, but in the lab it does (you'll see clair, clair-postgresql, quay,
    quay-postgresql, redis)
oc get routes                                                       # e.g.
    https://quay.apps.lab.example.com
# Configure Quay here via web GUI
podman login quay.apps.lab.example.com                              # on the workstation
skopeo copy docker://registry.lab.example.com/httpd:latest
    docker://quay.apps.lab.example.com/exampledev/httpd:latest
podman logout quay.apps.lab.example.com
sudo -i
podman login quay.apps.lab.example.com
podman pull quay.apps.lab.example.com/exampledev/httpd:latest
podman logout quay.apps.lab.example.com
```

Using Images Annotations for Security

```
<namespace containing image's quality>/<vulnerability|policy>.<provider ID>
quality.images.openshift.io/vulnerability.redhatcatalog: {SEE THE EXAMPLE BELOW}
quality.images.openshift.io/vulnerability.jfrog: {SEE THE EXAMPLE BELOW}
quality.images.openshift.io/license.blackduck: {SEE THE EXAMPLE BELOW}
quality.images.openshift.io/vulnerability.openscap: {SEE THE EXAMPLE BELOW}
```

Each annotation supports these fields:

- name (provider display name),
- timestamp of the scan,
- description (not required),
- reference,
- scannerVersion,
- compliant (yes x no),
- summary (label [critical/important], data, severityIndex, reference)

Example:

```
{
  "name": "Red Hat Container Catalog",
  "description": "Container health index",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://access.redhat.com/errata/RHBA-2016:1566",
  "compliant": null,
  "scannerVersion": "1.2",
```

```
  "summary": [
    {
      "label": "Health index",
      "data": "B",
      "severityIndex": 1,
      "reference": null
    }
  ]
}
```

```
oc describe is <image stream> -n openshift                    # read the status of an image stream
    to see the annotation
```

The annotation `images.openshift.io/deny-execution=true` is added by a security scanner, to define a policy that admission plugin prevents images from being retrieved if they are not marked as compliant: `vim /etc/origin/master/master-config.yml`:

```
admissionConfig:
  pluginConfig:
    openshift.io/ImagePolicy:
      configuration:
        kind: ImagePolicyConfig
        apiVersion: v1
        resolveImages: AttemptRewrite
        executionRules:
        - name: execution-denied
          onResources:
          - resource: pods
          reject: true
          matchImageAnnotations:                              # <--
          - key: images.openshift.io/deny-execution           # <--
            value: "true"                                     # <--
          skipOnResolutionFailure: true
```

Image signing and verification

- Signer server is the host responsible for generating the signature that embeds the image manifest digest and publish the signature to the signatures server
- On the server, the `/etc/containers/registries.d/` folder contains configuration files that specify where signatures are stored after their generation and where to download signature for each registry - e.g. `registry.yml`:

```
docker:
  registry.example.com:
    sigstore-staging: file:///var/lib/atomic/sigstore             # stores new sigs under the filesystem
    sigstore: http://sigstore.lab.example.com                     # retrieves them from web
```

Configure clients (nodes pulling images) - you can have different policies for different nodes:

```
vim /etc/containers/policy.json                                   # images require signatures?
```

Allow only images from the registry.lab.example.com server. All other images are rejected:

```
{
  "default": [
    {
      "type": "reject"
    }
  ],
  "transports": {
    "docker-daemon": {
      "": [
        {
          "type": "insecureAcceptAnything"
        }
```

```
      ]
    },
    "docker": {
      "registry.lab.example.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/home/signer/pub/image-signer-pub.gpg"
        }
      ]
    }
  }
}
```

LAB 2.2

```
sudo rngd -r /dev/urandom -o /dev/random
ps ax | grep rngd                                                    # see if the previous command is
    running
gpg2 --gen-key                                                       # generate GPG key pair as a student
  RSA
  1024
  0
  y
  Real name                                                          # "Image Signer"
  Email address                                                      # student@lab.example.com
  O                                                                  # (O)kay
reset
gpg2 --list-keys                                                     # /home/student/.gnupg/pubring.gpg
# Export the public key to the application nodes (import signer_key.pub to node1, 2)
# Signatures are stored by default in the /var/lib/atomic/sigstore directory - this is exposed as
    http://sigstore.lab.example.com
ssh node1 cat /etc/containers/registries.d/registry.yaml
   docker:
      registry.lab.example.com:
         sigstore-staging: file:///var/lib/atomic/sigstore
         sigstore: http://sigstore.lab.example.com
# SSH to connect to node1 in order to ensure that the node can access the signatures over HTTP
ssh node1 curl http://sigstore.lab.example.com                       # ensure that the node can access the
    signatures over HTTP
ssh node2 curl http://sigstore.lab.example.com
podman search registry.lab.example.com/mongodb-32-rhel7              # ensure that the mongodb-32-rhel7
    image is present
skopeo copy --sign-by student@lab.example.com
    docker://registry.lab.example.com/rhscl/mongodb-32-rhel7:latest
    docker://registry.lab.example.com/rhscl/mongodb-32-rhel7:signed
                                                 # --sign-by option allows you to generate a detached
    signature
firefox http://sigstore.lab.example.com                              # look at the signature in rhscl
    folder
skopeo inspect docker://registry.lab.example.com/rhscl/mongodb-32-rhel7:signed | grep Digest # same
    signature
ssh root@node1
vim /etc/containers/policy.json                                      # unsigned images rejected
  ...output omitted...
  {
    "default": [
    {
        "type": "reject"
    }
  ],
```

```
  ...output omitted...
  "docker": {
    "registry.lab.example.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/root/.gnupg/pubring.gpg"
        }
oc login -u developer -p redhat https://master.lab.example.com
oc new-project containers-images
oc new-app -f ~/DO425/labs/containers-images/containers-images.yaml   # contains unsigned httpd and signed
    mongo
oc get pods                                                           # httpd will be ImagePullBackOff,
    because it isn't signed
oc describe pod httpd | grep 'Source image rejected'
oc delete project containers-images
```

Inspecting image layers

```
mkdir /tmp/mongo
skopeo copy docker://registry.lab.example.com/rhscl/mongodb-32-rhel7 dir:/tmp/mongo # extracts the
    rhscl/mongodb-32-rhel7 image in the /tmp/mongo/ directory
ls /tmp/mongo
cat /tmp/mongo/manifest.json
  schemaVersion                                                       # 2 for OCI
  config                                                              # image metadata, location of the
      config in the folder structure
  layers                                                              # base layer first (size, digest ...)
mkdir /tmp/layer
tar xf /tmp/mongo/9205...f599 -C /tmp/layer                           # layer is a compressed tar file
ls /tmp/layer                                                         # bin/ boot/ dev/ ..., other layers
    only contain additions or modifications to those base directories
```

```
# Examining Packages with rpm
rpm -qa                                                               # query all packages installed
rpm -qi <PACKAGE>                                                     # info about the package
rpm -ql <PACKAGE>                                                     # files installed by the specified
    package
rpm -q --scripts                                                      # install/remove scripts

# Listing the RPM Packages in an Image Layer
rpm -q -a --root /tmp/layer                                           # list all packages in layer
rpm -q bash --root /tmp/layer                                         # list package bash in layer
rpm -q bash -i --root /tmp/layer                                      # info about bash in layer
```

Introducing Clair

```
oc logs clair-2-bfp2c | grep fetching                                 # Clair log shows the retrieval of
    the vulnerability metadata from the Internet
```

- Clair only analyses images based on Red Hat Enterprise Linux, Alpine Linux, Debian, Oracle Linux, and Ubuntu because it only retrieves the vulnerabilities from these system vendors or projects
- Clair also limits its scan to the distribution packages and does NOT check vulnerabilities in your application code, or libraries or artifacts retrieved from other sources

LAB 2.3

```
podman login quay.apps.lab.example.com
skopeo copy docker://registry.lab.example.com/rhscl/mongodb-32-rhel7
    docker://quay.apps.lab.example.com/admin/mongodb-32-rhel7
skopeo copy docker://registry.lab.example.com/rhscl/php-56-rhel7
    docker://quay.apps.lab.example.com/admin/php-56-rhel7
```

```
podman logout quay.apps.lab.example.com
# While Quay is scanning, inspect the layers
mkdir /tmp/php
skopeo copy docker://registry.lab.example.com/rhscl/php-56-rhel7 dir:/tmp/php
cat /tmp/php/manifest.json
  {
  "schemaVersion": 2,
  "config": {
    "digest": "sha256:520f...f381"
    },
  },
  "layers": [
    {
      ...
    },
    {
      "digest": "sha256:e0b3...4dad"
    }
  ]
  }
# Object containing more details (build, author, ports, env) about the image is located in:
cat /tmp/php/520f...f381 | python -m json.tool                          # config digest = config folder
# Extract the lowest layer of the image:
mkdir /tmp/php/layer
sudo tar xf e0b3...4dad -C /tmp/php/layer                               # extract lowest layer (last entry)
ls /tmp/php/layer
rpm -qa --root /tmp/php/layer
rpm -q patch --root /tmp/php/layer                                      # retrieve the version of the patch
    package in the layer
sudo rm -rf /tmp/php
# See if Quay found vulnerabilities in the patch package 2.7.1-8.el7
```

FINAL LAB 2

```
TBA
```

3. Implementing Security in the Build Process

Implementing Image Change Triggers

```
oc set triggers bc build-config-name --from-image="imagestream"       # change made to that image stream
    automatically starts a build process, this configures image change trigger in an existing build config.
    Although it says --from-image, it is in fact image stream

# The hook fails if the script or command returns a non-zero exit code or if starting the temporary
    container fails. In that case, the image is not pushed to the registry.
oc set build-hook bc/build-config-name --post-commit --command <> --command <> # create build hook in an
    existing bc (or instead of --command, use --script="/path/to/script")
```

LAB 3.1

```
git clone http://services.lab.example.com/openshift-tasks
cat home/student/DO425/labs/build-s2i/security.txt                      # It uses findbugs-maven-plugin
    profile
cat /home/student/openshift-tasks/pom.xml                               # append to the last closing element
    the content of security.txt
cd openshift-tasks
git add .
git commit -m "Added Security support"
git push
oc login -u developer -p redhat https://master.lab.example.com
oc new-project build-s2i
```

```
oc new-app --build-env
    MAVEN_MIRROR_URL=http://services.lab.example.com:8081/nexus/content/groups/training-java
    jboss-eap71-openshift:latest~http://services.lab.example.com/openshift-tasks
                                # start a S2I build, MAVEN_MIRROR_URL parameter is used by the build to
    download the dependencies from an internal repository as the Internet is not available
oc logs -f builds/openshift-tasks-1                                    # check the build is running
oc set build-hook bc/openshift-tasks --post-commit --script="cd /tmp/src; mvn test -Psecurity" # update
    hook to invoke the security check plugin as part of build
oc export bc/openshift-tasks | grep -A1 postCommit                     # inspect build configuration
  postCommit:
    script: cd /tmp/src; mvn test -Psecurity
oc start-build openshift-tasks                                         # test the build hook, starts build 2
oc logs -f builds/openshift-tasks-2                                    # inspect if the web hook was
    executed. It shows mvn not found + cannot set terminal process group (-1): Inappropriate ioctl for
    device
oc set triggers bc openshift-tasks --from-image='jboss-eap71-openshift:latest' # configure the build
    configuration to execute the build process again when the IS is updated
oc export bc/openshift-tasks | grep -A5 "ImageChange"                  # ensure that the image change
    trigger is created
# Unpack the container image to push the update the classroom registry:
cd ~/DO425/labs/build-s2i/
tar xzf eap71-openshift.tar.gz
skopeo copy oci:eap71-openshift-1.3-17 docker://registry.lab.example.com/jboss-eap-7/eap71-openshift:latest
oc login -u admin -p redhat
oc import-image jboss-eap71-openshift:latest --confirm -n openshift    # update the internal registry with
    the container image from the classroom registry
oc login -u developer -p redhat
oc get builds                                                          # build 3 is running automatically
oc logs -f build/openshift-tasks-3
oc expose svc openshift-tasks
firefox http://openshift-tasks-build-s2i.apps.lab.example.com          # if you see website, it is OK
```

Ensure Jenkins is deployed

```
oc new-app jenkins --param ENABLE_OAUTH=true                           # deploy Jenkins container with
    persistent storage and use the OCP credentials to log in to it
```

To create Jenkins pipeline, you have to create this resource:

```
apiVersion: v1
kind: BuildConfig
//omitted
strategy:
  jenkinsPipelineStrategy:
    jenkinsfile: |-
    def mvnCmd = "mvn -s configuration/cicd-settings.xml"
    pipeline {
      agent {
        label 'maven'
      }
    //stages omitted
    }
  }
  type: JenkinsPipeline
```

```
oc create -f definition.yaml
oc start-build buildconfig-name
```

Integration point in a Jenkins Pipeline - slave hosts are started as containers using a Jenkins Kubernetes plugin:

- `<name>` is the name of the slave used in Jenkinsfile (agent - label)
- `<image>` is the container image to start the build process, there are many types of Jenkins slaves

Example of the agent definition:

```
- apiVersion: v1
  kind: ConfigMap
  data:
    maven-template: |-
      <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
        <inheritFrom></inheritFrom>
        <name>maven</name>                                          # slave name used by Jenkins pipeline
        #configuration omitted
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>registry.lab.example.com/openshift/jenkins-agent-maven-35-centos7</image> # image used to
              start build process
  # configuration omitted
  </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
```

```
# An example of the step in Jenkins pipeline - integrating security check tools:
stage('Validate source code') {
  steps {
    sh "${mvnCmd} deploy -DskipTests=true -P nexus3"
  }
}
```

LAB 3.2

```
oc login -u developer -p redhat https://master.lab.example.com
oc project cicd
oc get pods                                                         # ensure Jenkins & SonarQube are
    running
cat /home/student/DO425/labs/build-pipeline/pipeline-template.yaml
```

```
  def mvnCmd = "mvn -s configuration/cicd-settings.xml"
  pipeline {
    agent {
      label 'maven'
    }
    . . . omitted "Build app", "Test", "Archive app", "Build image", "Deploy DEV"
    stages {
      stage('Promote to STAGE?') {
        agent {
          label 'skopeo'
        }
      // Add skopeo call below.
    }
    . . . ommitted "Deploy STAGE"
```

```
# Skopeo call:
# Implement the image tagging step that tags the image on Quay for the stage environment:
  steps {
    script {
      openshift.withCluster() {
        withCredentials([usernamePassword(credentialsId:"${openshift.project()}-quay-cicd-secret",
            usernameVariable: "QUAY_USER", passwordVariable: "QUAY_PWD")]) { sh "skopeo copy
            docker://quay.apps.lab.example.com/admin/tasks-app:latest
            docker://quay.apps.lab.example.com/admin/tasks-app:stage --src-creds \"$QUAY_USER:$QUAY_PWD\"
            --dest-creds \"$QUAY_USER:$QUAY_PWD\" --src-tls-verify=false --dest-tls-verify=false"
        }
      }
    }
  }
```

```
# Note: Command is also located in /home/student/DO425/labs/build-pipeline/command.txt
cd ~/DO425/labs/build-pipeline
```

```
oc new-app pipeline-template.yaml                                    # deploying template cicd/cicd for
    pipeline-template.yaml to project cicd
oc start-build bc/tasks-pipeline
oc logs build/tasks-pipeline-1
# open Jenkins and promote to stage, inspect security metrics:
https://sonarqube-cicd.apps.lab.example.com
https://quay.apps.lab.example.com
```

FINAL LAB 3

```
TBA
```

4. Managing User Access Control

RBAC resources:

- Users = can make requests to OpenShift API
- Service Accounts = used for delegating certain tasks to OpenShift
- Groups
- Roles = collections of rules
- Rules = define verbs that users/groups can use with a given resource
- Security Context Constraints = control the actions pod/container can perform
- Role bindings = roles to users/groups

```
oc describe clusterrole admin
  Name: admin
  Created:  5 weeks ago
  Labels: kubernetes.io/bootstrapping=rbac-defaults
  Annotations:  openshift.io/description=A user that has edit rights within the project and can change the
      project's membership.
      openshift.io/reconcile-protect=false

  Verbs                      (...)     Resources
  [delete get patch update] (...)     [projects]                    # user can retrieve list of projects,
      delete/modify them
  [get list watch]           (...)     [namespaces]
```

Two levels:

1. Cluster-wide RBAC - applicable across all projects
2. Local RBAC - apply to a given project

```
oc describe rolebinding admin -n developer                           # access local role bindings
  Name: admin
  Namespace:  developer
  Created:  9 minutes ago
  Labels: <none>
  Annotations:  <none>
  Role: /admin
  Users:  developer
  Groups: <none>
  ServiceAccounts:  <none>
  Subjects: <none>
```

```
  Verbs              (...) Resources
  [get list watch]   (...) [namespaces]
  [impersonate]      (...) [serviceaccounts]
  ...output omitted...
```

The following excerpt shows how to include a SCC to a role. This gives privileges to the user or the group that uses this role to access the `restricted-scc` security context constraint:

```
rules:
  apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints                                    # name of the resource group that
      allows you to include a SCC in the role
  verbs:
  - use
  resourceNames:
  - restricted-scc                                                # name of the SCC that is applied to
      all pods or containers that inherit this role
```

```
# Some of the defaults cluster roles that are available in OpenShift - e.g.:
oc describe clusterrole self-provisioner

# Managing Roles
oc create role <name> --verb=<verb> --resource=<resource type> -n <project> # local role for a project
oc create clusterrole <name> --verb=<verb> --resource=<resource>      # cluster role

# Managing Role Bindings - cluster level
oc adm policy add-cluster-role-to-user role user                      # Binds a given role to specified
    users for all projects in the cluster
oc adm policy remove-cluster-role-from-user role user                 # Removes a given role from specified
    users for all projects in the cluster.
oc adm policy add-cluster-role-to-group role group                    # Binds a given role to specified
    groups for all projects in the cluster.
oc adm policy remove-cluster-role-from-group role group               # Removes a given role from specified
    groups for all projects in the cluster.

# Managing Role Bindings - local/project level (-n <PROJECT>, otherwise in current project)
oc adm policy who-can verb resource                                   # Indicates which users can perform
    an action on a resource.
oc adm policy add-role-to-user verb resource                          # Binds a given role to specified
    users in the current project.
oc adm policy remove-role-from-user verb resource                     # Removes a given role from specified
    users in the current project.
oc adm policy remove-user user                                        # Removes specified users and all of
    their roles in the current project.
oc adm policy remove-user user                                        # Removes specified users and all of
    their roles in the current project.
oc adm policy add-role-to-group role group                            # Binds a given role to specified
    groups in the current project.
oc adm policy remove-role-from-group role group                       # Removes a given role from specified
    groups in the current project.
oc adm policy remove-group group                                      # Removes specified groups and all of
    their roles in the current project.

oc adm policy add-role-to-user edit manager -n developer              # Give the manager user the edit role
    in the developer project
oc describe rolebinding manager -n developer
```

Determining User Privileges

```
oc auth can-i delete pods                                             # yes
oc auth can-i create deploymentconfigs                                # yes
oc auth can-i delete users                                            # no - no RBAC policy matched
```

```
oc policy can-i --list                                          # determine all the actions that you
    can perform
```

INTRODUCING TOKENS

```
cat ~/.kube/config
oc whoami --show-token                                          # only for 24 hours
oc sa get-token registry                                        # long-lived token of the registry
    service account
```

LAB 4.1

```
oc login -u developer1 -p redhat https://master.lab.example.com
oc new-project project1
oc new-app -f /home/student/DO425/labs/auth-access/auth-access.yaml   # The application deploys an httpd pod
oc login -u developer2 -p redhat https://master.lab.example.com
oc new-project project2
oc new-app -f /home/student/DO425/labs/auth-access/auth-access.yaml   # The application deploys an httpd pod
oc login -u manager -p redhat
oc get pods -n project1                                         # User "manager" cannot list pods in
    the namespace "project1": no RBAC policy matched
oc get pods -n project2                                         # same error
firefox https://master.lab.example.com                         # log in as admin
  Administration -> Projects -> project1 -> Role bindings -> Subject name: developer1
  Administration -> Projects -> project2 -> Role bindings -> Subject name: developer2
# grant read access to the manager user for the project1 project and read and write accesses for the
    project2 project:
  project1 -> Role Bindings -> Creating Binding -> Namespace role binding:
  Role Binding Name: viewer
  Namespace: project1
  Role Name: view
  Subject: User
  Subject Name: manager

  project2 -> Role Bindings -> Creating Binding -> Namespace role binding:
  Role Binding Name: editor
  Namespace: project2
  Role Name: edit
  Subject: User
  Subject Name: manager
# delete the role binding that gives the developer1 user administrative privileges in the project1 project
  project1 -> Role Bindings -> admin [gear] -> Delete Role Binding
# Test if it worked:
oc login -u developer1 -p redhat
oc delete pod httpd -n project1                                 # forbidden
oc login -u manager -p redhat
oc get pods -n project1                                         # works this time
oc delete pod httpd -n project1                                 # forbidden (only view role in
    project 1)
oc delete pod httpd -n project2                                 # works (edit role in project 2)
oc delete project project2                                      # forbidden, manager only had edit
    role
  Administration -> Projects -> project2 -> Role Bindings -> Creating binding
  Role Binding Name: admin-1
  Namespace: project2
  Role Name: admin
  Subject: User
  Subject Name: manager
oc delete project project2                                      # works now
oc delete user developer2                                       # forbidden, requires cluster-wide
    priv
oc login -u admin -p redhat
```

```
oc delete project project1                                    # works
```

CONFIGURING AN OPENSHIFT IDENTITY PROVIDER FOR RED HAT IDENTITY MANAGEMENT (IdM)

- OpenShift masters can be configured with different identity providers that allow an OpenShift cluster to delegate user authentication and group membership management to different identity stores.

- To configure an OpenShift `LDAPPasswordIdentityProvider` identity provider to integrate with an IdM domain, you need the following information about your IdM domain and servers:
  - DNS domain name of your IdM domain (`organization.example.com`)
  - FQDN of one of your IdM servers (`ldap1.organization.example.com`)
  - LDAP user name with read access of the entire user accounts tree (`uid=admin,cn=users,cn=accounts,dc=organization,dc=e`
  - LDAP container of the user accounts tree (`cn=users,cn=accounts,dc=organization,dc=example,dc=com`)
  - public key TLS certificate of your IdM domain (`/etc/ipa/ca.crt` file in any server or client of your IdM domain)

- Example stanza under the `identityProviders` attribute on the OpenShift master configuration file `/etc/origin/master/master-co`

```
. . .
identityProviders:
- challenge: true
  login: true
  mappingMethod: claim
  name: idm_ldap_auth                                         # can be anything
  provider:
    apiVersion: v1
    kind: LDAPPasswordIdentityProvider
    attributes:
      id:
      - dn
      preferredUsername:
      - uid
    insecure: false                                           # use TLS, refuse invalid certs
    ca: /etc/origin/master/idm-domain-ca.crt                  # public key TLS cert of the IdM
        domain
    url:
        "ldaps://server1.organization.example.net:636/cn=users,cn=accounts,dc=organization,dc=example,dc=com?uid"
        # LDAP search URL for user accounts
    bindDN: "uid=admin,cn=users,cn=accounts,dc=organization,dc=example,dc=com" # username
    bindPassword: "secret"                                    # password
```

SYNCHRONIZING GROUPS BETWEEN OPENSHIFT AND IdM

```
oc adm groups sync --confirm                                  # run regularly to keep OpenShift and
    LDAP groups in sync
oc adm groups sync --help                                     # whitelist & blacklist!
    # Sync all groups from an LDAP server
    oc adm groups sync --sync-config=/path/to/ldap-sync-config.yaml --confirm # see LDAPSyncConfig

    # Sync all groups except the ones from the blacklist file from an LDAP server
    oc adm groups sync --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml
        --confirm

    # Sync specific groups specified in a whitelist file with an LDAP server
    oc adm groups sync --whitelist=/path/to/whitelist.txt --sync-config=/path/to/sync-config.yaml --confirm

    # Sync all OpenShift Groups that have been synced previously with an LDAP server
```

```
    oc adm groups sync --type=openshift --sync-config=/path/to/ldap-sync-config.yaml --confirm

    # Sync specific OpenShift Groups if they have been synced previously with an LDAP server
    oc adm groups sync groups/group1 groups/group2 groups/group3 --sync-config=/path/to/sync-config.yaml
        --confirm

  Options:
    --blacklist='': path to the group blacklist file
    --confirm=false: if true, modify OpenShift groups; if false, display results of a dry-run
    --sync-config='': path to the sync config
    --type='ldap': which groups white- and blacklist entries refer to: ldap,openshift
    --whitelist='': path to the group whitelist file

oc adm groups prune --confirm                                    # deletes all current orphan groups
    from OpenShift
```

Configuring LDAP Synchronization Connection Parameters (similar to LDAPPasswordIdentityProvider):

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldaps://server1.organization.example.com:636                # LDAP URI FQDN
bindDN: uid=admin,cn=users,cn=accounts,dc=organization,dc=example,dc=com # username
bindPassword: secret                                             # password
insecure: false                                                  # use TLS
ca: /home/student/idm-ca.crt                                     # must exist on the client where you
    run the 'oc'
augmentedActiveDirectory:                                        # variation of MS AD schema
  groupsQuery:
    baseDN: "cn=groups,cn=accounts,dc=organization,dc=example,dc=com" # user account container
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn
  groupNameAttributes: [ cn ]
  usersQuery:
  baseDN: "cn=users,cn=accounts,dc=organization,dc=example,dc=com"    # groups container
  scope: sub
  derefAliases: never
  filter: (objectclass=inetOrgPerson)
  pageSize: 0
userNameAttributes: [ uid ]
groupMembershipAttributes: [ memberOf ]
```

MANAGING OPENSHIFT USERS AND IDENTITIES

- You may need to delete the identity resource for a user if the same user name exists on different identity providers.

```
oc get user
  NAME    ...   IDENTITIES
  admin   ...   htpasswd_auth:admin
  idmuser ...   idm_ldap_auth:uid=idmuser,cn=users,cn=accounts,dc=...
```

OpenShift retains identities for deleted users, and these identities may prevent a new user from logging in, if that user has the same name of an old user:

```
oc delete user idmuser
oc delete identity idm_ldap_auth:uid=idmuser,cn=users,cn=accounts,dc=...
```

LAB 4.2

```
scp root@idm:/etc/ipa/ca.crt ~/idm-ca.crt                        # copy the public certificate of root
    CA of the IdM server to the student user home folder on the workstation
```

```
export LDAPTLS_CACERT=~/idm-ca.crt
ldapsearch -x -H ldaps://idm.lab.example.net:636 -D
    "uid=admin,cn=users,cn=accounts,dc=lab,dc=example,dc=net" -w RedHat123^ -b
    "cn=accounts,dc=lab,dc=example,dc=net" -s sub= '(objectClass=posixaccount)' uid # see existing users
ldapsearch -x -H ldaps://idm.lab.example.net:636 -D
    "uid=admin,cn=users,cn=accounts,dc=lab,dc=example,dc=net" -w RedHat123^ -b
    "cn=accounts,dc=lab,dc=example,dc=net" -s sub= '(objectClass=posixgroup)' cn member # see existing
    groups
scp ~/idm-ca.crt root@master:/etc/origin/master/                    # copy the root CA to master
ssh root@master
# Copy & paste LDAPPasswordIdentityProvider from ~/DO425/labs/auth-idm/idm-identity-provider.yaml to the
    master configuration /etc/origin/master/master-config.yaml
master-restart api                                                   # restart master pods ...
master-restart controllers                                           # ... and the node service
systemctl restart atomic-openshift-node
oc login -u admin -p redhat https://master.lab.example.com
oc get user                                                          # only admin & developer
oc get group                                                         # no resources
oc login -u jrdev1 -p redhat
oc project internalapp
oc login -u admin -p redhat
oc get user                                                          # now contains also jrdev1
oc get group                                                         # no resources
oc adm groups sync --confirm --sync-config ~/DO425/labs/auth-idm/idm-sync-config.yaml # synchronize
    OpenShift and IdM groups
oc get group                                                         # ipausers, juniordevs, seniordevs
oc login -u jrdev1 -p redhat
oc project internalapp
oc create cm myconfig --from-literal key1=value1
oc project externalapp
oc login -u srdev1 -p redhat
oc project externalapp
oc create cm myconfig --from-literal key2=value2
oc project internalapp
ssh root@idm
kinit admin                                                          # RedHat123^
ipa user-add srdev2 --first=Developer2 --last=Senior
ipa user-mod srdev2 --password
ipa user-mod srdev2 --setattr=krbPasswordExpiration=20250606060606Z
ipa group-add-member seniordevs --users=srdev2                       # make srdev2 member of the group
oc login -u srdev2 -p redhat
oc project externalapp                                               # you are not member of the project
oc login -u admin -p redhat
oc adm groups sync --confirm --sync-config ~/DO425/labs/auth-idm/idm-sync-config.yaml
oc get group                                                         # seniordevs = srdev1, srdev2
oc login -u srdev2 -p redhat
oc project externalapp                                               # now it works
oc create cm myconfig2 --from-literal key3=value3
```

DEPLOYING SINGLE SIGN-ON ON OPENSHIFT

Passthrough vs. re-encryption SSO templates

Typically coming from `registry.redhat.io/redhat-sso-7/sso72-openshift:1.0`

To integrate an application with Red Hat's SSO server you define and configure, at minimum, one 'realm', one or more 'clients', and one or more 'users':

1. Web console: `https://sso-fqdn/auth/admin`

2. `/bin/kadm.sh`

CONFIGURING AN OPENSHIFT IDENTITY PROVIDER FOR SSO

The `OpenIDIdentityProvider` identity provider allows OpenShift to delegate authentication to a SSO server using the OpenID Connect standard (`master-config.yml`):

```
. . .
identityProviders:
- challenge: true
  login: true
  mappingMethod: claim
  name: sso_realm                                                    # name of your choice
  provider:
    apiVersion: v1
    kind: OpenIDIdentityProvider
    clientID: MyApp                                                  # client ID of existing client on the
        SSO realm
    clientSecret: 41b5677d-09b1-4ffc-890a-99f10204bde9               # secret of the SSO client
    ca: /etc/origin/master/ca-bundle.crt                             # absolute path on the master
    urls:
      authorize: https://sso.server.example.com/auth/realms/MyRealm/protocol/openid-connect/auth
      token: https://sso.server.lab.example.com/auth/realms/MyRealm/protocol/openid-connect/token
      userInfo: https://sso.server.lab.example.com/auth/realms/MyRealm/protocol/openid-connect/userinfo
    claims:
      id:
      - sub
      preferredUsername:
      - preferred_username
      name:
      - name
      email:
      - email
```

The OpenID Connect API endpoints of your SSO realm, that follow the format: `https://<sso-server-fqdn>/auth/realms/<RealmName>`

operation = Name of an OpenID Connect API operation, such as 'auth', 'token', and 'userinfo'

```
curl -sk https://sso-server-fqdn/auth/realms/RealmName /.well-known/openid-configuration | python -m
    json.tool # lists all URLs for a SSO realm, for example the RealmName realm
```

LAB 4.3

```
git clone http://services.lab.example.com/sso72-dev                  # copy of the sso folder of the
    jboss-openshift/application-templates project
less sso72-dev/sso72-https.json | grep sso72-openshift               # sample template to deploy the
    sign-on containerized server
less sso72-dev/sso72-image-stream.json | grep sso72-openshift        # image stream that points to the
    single sign-on container image
podman search registry.lab.example.com/sso                           # is thie image available on the
    classroom registry?
oc login -u admin -p redhat https://master.lab.example.com
oc project openshift
for f in ~/sso72-dev; do oc delete -f $f; done                       # delete existing SSO resources, SSO
    image stream and templates may already exist
for f in ~/sso72-dev; do oc create -f $f; done                       # create new IS and template
    resources for SSO
oc describe istag redhat-sso72-openshift:1.2 | head -n 3             # did IS import container
    'redhat-sso72-openshift'?
~/DO425/labs/auth-sso/provision-storage.sh sso-db                    # provision NFS storage for SSO DB
    and create PV
oc get pv | grep sso-db                                              # verify that the sso-db persistent
    volume is available
oc login -u developer -p redhat
oc new-project websso
oc new-app --template sso72-x509-postgresql-persistent -p APPLICATION_NAME=sso -p DB_USERNAME=sso -p
    DB_PASSWORD=redhat -p VOLUME_CAPACITY=512Mi -p SSO_ADMIN_USERNAME=admin -p SSO_ADMIN_PASSWORD=redhat
                    # use the sso72-x509-postgresql-persistent template to deploy a containerized
    PostgreSQL database and a containerized SSO server
```

```
oc get pvc | grep sso-postgresql-claim
#oc rollout cancel dc/sso                                    # if it takes too long and sso pod is
    in error state
oc get pod | grep sso                                       # 'sso-1-XYZW' and
    'sso-postgresql-1-XYZW' should be running
oc logs sso-1-XYZW | tail -n 2                              # should say started, must not say
    "with errors" or "X services failed or missing dependencies"
firefox https://sso-websso.apps.lab.example.com/auth/admin
```

1. Add realm OpenShift
2. Create the contractordev1 user in the OpenShift realm
3. Assign a non-temporary password to the contractordev1 user
4. Create the MasterAPI client ID in the OpenShift realm with openid-connect client protocol
5. In MasterAPI settings, select Confidential in the 'Access Type' field and type
   https://master.lab.example.com:443/* in the Valid Redirect URIs.
6. Retrieve the client secret for the MasterAPI client's credential tab
7. Log off from the single sign-on web console.

```
firefox https://sso-websso.apps.lab.example.com/auth/realms/OpenShift/account
```

1. Login as contractordev1 and retype the password to activate the account.
2. Sign out

```
ssh root@master
cat ~/DO425/labs/auth-sso/sso-identity-provider.yaml        # copy the content
vim /etc/origin/master/master-config.yaml                   # paste it here = add a new
    OpenIDIdentityProvider, do not delete HTPAsswdPasswordIdentityProvider
master-restart api
master-restart controllers
systemctl restart atomic-openshift-node
oc login -u admin -p redhat
oc login -u contractordev1 -p redhat                        # verify that OpenShift can
    authenticate users from single sign-on
oc project contractorapp
oc create cm myconfig --from-literal key1=value1            # create ConfigMap in the
    contractorapp project to prove user has write access
oc project internalapp                                      # error = You are not a member of
    project "internalapp"
```

FINAL LAB 4

```
TBA
```

5. Controlling the Deployment Environment

REVIEWING SECRETS AND CONFIGMAPS

- You can store the registry credentials in a secret and instruct OpenShift to use that secret when it needs to push and pull images from the registry.

SECRETS

Individual secrets are limited to 1MB in size.

```
oc create secret generic secret_name --from-literal=username=operator --from-literal=password=MyS3cr3T
```

Or in YAML:

```
echo operator | base64                                      # b3BlcmF0b3IK
echo MyS3cr3T | base64                                      # TXlTM2NyM1QK
vim secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: secret_name
data:
  username: b3BlcmF0b3IK                                          # must be base64 encoded
  password: TXlTM2NyM1QK                                          # must be base64 encoded
```

```
oc create -f secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque                                                      # bypasses any validation
data:
  password: dmFsdWUtMQ0K                                          # must be base64 encoded
stringData:
  hostname: myapp.mydomain.com
  admin_password: redhat123                                       # plain text value will automatically
      be converted to base64
```

How to use the above secret in the deployment config:

```
...output omitted...
env:
- name: APP_ADMIN_PASSWORD                                        # environment variable retrieves
    value from test-secret secret
  valueFrom:
    secretKeyRef:                                                 # similarly to configMapKeyRef
      name: test-secret
      key: admin_password
```

Another use case is the passing of data, such as TLS certificates, to an application by using the `--from-file=file` option. This exposes a sensitive file to an application. The pod definition can reference the secret, which creates the secret as files in a volume mounted on one or more of the application containers.

```
...output omitted...
      spec:
        containers:
...output omitted...
          volumeMounts:
          - mountPath: /conf/app/extra_ca_certs                   # mounts certs_volume to
              /conf/app/extra_ca_certs
            name: certs_volume
        volumes:
        - name: certs_volume
          secret:
            defaultMode: 420
            secretName: application-certificates                  # maps the volume to
                application_certificates secret
```

CONFIGURATION MAP

```
oc create configmap special-config --from-literal=serverAddress=172.20.30.40
```

Or in YAML:

```
vim special-config.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
```

```
data:
  serverAddress: 172.20.30.40
```

```
oc create -f special-config.yaml
```

Populate the APISERVER environment variable inside a pod definition from the above configuration map:

```
env:
- name: APISERVER
  valueFrom:
    configMapKeyRef:                                          # similarly to secretKeyRef
      name: special-config
      key: serverAddress
```

Encrypting Secrets in Etcd

1. On master nodes, define the experimental-encryption-provider-config in the `/etc/origin/master/master-config.yaml` file:

```
...output omitted...
kubernetesMasterConfig:
  apiServerArguments:
    experimental-encryption-provider-config:
    - /etc/origin/master/encryption-config.yaml
    storage-backend:
    - etcd3
...output omitted...
```

1. Create `encryption-config.yaml` file: `vim /etc/origin/master/encryption-config.yaml`

```
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets:
    providers:
    - aescbc:                                                 # algorithm used
      keys:
      - name: key1
        secret: hifzYhVwbOFYL7X+oGoEqwblf94sJFoVg7IOECcCk+Q=  # base64 encoded
    - identity: {}
```

```
head -c 32 /dev/urandom | base64                              # random secret used above as key1
master-restart api
master-restart controllers
# From now on, new secret object are encrypted:
etcdctl3 get /kubernetes.io/secrets/mytest/mysecret2 -w fields --cacert=/etc/etcd/ca.crt
    --key=/etc/etcd/peer.key --cert=/etc/etcd/peer.crt --endpoints 'https://172.25.250.10:2379' | grep Value
```

Preparing Secrets for Accessing Authenticated Registry

```
oc create secret docker-registry <secret_name> --docker-username=<user> --docker-password=<pass>
    --docker-server=<registry_full_host_name>
```

With Quay, you can create robot accounts (tokens) and grant them access to the repositories in an organization. Quay can generate a YAML Kubernetes resource file that you can also use with OpenShift (`oc create -f ~/Downloads/myorg-openshift-secret.yml`).

Configuring Project Service Account for Image PUSHING

The build process uses the OpenShift builder service account in your project. For the builder service account to automatically use that secret for authentication, link it to the secret:

```
oc secrets link serviceaccount/builder secret/<secret_name>
oc new-build GIT_project_URL --push-secret=<secret_name> --to-docker
    --to=<remote_registry>/<namespace>/<image> # S2I build process will now push to another registry
```

Configuring Project Service Account for Image PULLING

```
oc secrets link serviceaccount/default secret/<secret_name> --for=pull # link default SA in your project to
    the secret
```

Configuring OpenShift for Accepting Certificates Signed by a Private CA

Install CA's certificate on each node, under the `/etc/docker/certs.d/<registry_full_host_name>/` directory:

```
mkdir /etc/docker/certs.d/quay.apps.lab.example.com/
mv -v *.crt /etc/docker/certs.d/quay.apps.lab.example.com/
```

```
# v3.10 and later:
# Configure the OpenShift API Server to use the CA's certificate, on the master you do:
cat myca.crt >> /etc/origin/master/ca-bundle.crt                    # path to the
    /etc/origin/master/ca-bundle.crt file is defined by the additionalTrustedCA variable in
    /etc/origin/master/master-config.yaml
master-restart api

# v3.11.43 and later:
oc version                                                          # v3.11?
# in addition to the certificates in /etc/origin/master/ca-bundle.crt, the API server also trusts the
    certificates in the system /etc/pki/tls/certs/ca-bundle.trust.crt file:
mv -v *.crt /etc/pki/ca-trust/source/anchors/
update-ca-trust extract
```

LIMITING REGISTRIES, PROJECTS, AND IMAGES

```
cat /etc/containers/policy.json                                     # on all nodes, no need to restart
    when changed, default file accepts any image:
```

```
{
  "default": [
    {
      "type": "insecureAcceptAnything"
    }
  ],
  "transports":
    {
      "docker-daemon":
      {
        "": [{"type":"insecureAcceptAnything"}]
      }
    }
}
```

- system uses the default entry when no other rule matches
- `insecureAcceptAnything` = accept any image
- `reject` = refuse all images (usually set this requirement in the default entry and add specific rules to allow your images)
- `signedBy` = accept signed images (provide additional parameters such as the public GPG key to use to verify signatures)
- Using wildcards or partial names does NOT work!
- Under the `transports` section file groups registries by type: `docker` (Registry v2 API), `atomic` (OCR), `docker-daemon` (local daemon storage):

Another example:

```
{
    "default": [
        {
```

```
            "type": "reject"                                        <-- reject all images if no other
                rule matches
        }
    ],
    "transports": {
        "docker": {
            "registry.access.redhat.com": [                         <-- use any images from
                registr.access.redhat.com as long as they are GPG signed
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ],
            "quay.apps.lab.example.com/production": [               <-- use any image from the production
                organization in Quay
                {
                    "type": "insecureAcceptAnything"
                }
            ],
            "quay.apps.lab.example.com/staging/mypostgres-95:latest": [ <-- pull latest tag of the
                mypostgresl image in the staging of Quay
                {
                    "type": "insecureAcceptAnything"
                }
            ]
        },
        "atomic": {
            "172.30.13.172:5000": [                                 <-- pull any image from the local
                OpenShift Container Registry (OCR)
                {
                    "type": "insecureAcceptAnything"
                }
            ]
        },
        "docker-daemon": {
            "": [
                {
                    "type": "insecureAcceptAnything"
                }
            ]
        }
    }
}
```

Configuring Signature Transports

That above configuration (`"type"`, `"keyType"`, `"keyPath"`) is not enough; you also need to indicate the URL to the web server that stores the detached image signatures. To declare that URL, create a file under `/etc/containers/registries.d/` such as:

```
cat /etc/containers/registries.d/redhat.yaml
docker:
  registry.access.redhat.com:
    sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

For the OpenShift Container Registry, that you define with the 'atomic' transport type, you do not need to perform this extra configuration. The OCR has API extensions to store the signatures, and the atomic transport type consumes them.

USING DEPLOYMENT TRIGGERS

```
oc get dc                                                           # look at 'TRIGGERED BY' column
```

For example if it says `config,image(redis:latest)`, there are two types of triggers:

1. A configuration change trigger causes a new deployment to be created any time you update the deployment configuration object itself.

2. An image change trigger causes OpenShift to redeploy the application each time a new version of the redis:latest image is available in the registry

When you create an application with the `oc new-app` command, or from the web interface, OpenShift creates a deployment configuration object with the above two triggers already defined.

This is how it looks like in the YAML:

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
...output omitted...
  triggers:
  - type: ConfigChange
  - type: ImageChange
    imageChangeParams:
      automatic: true
      containerNames:
      - <container_name>
      from:
        kind: ImageStreamTag
        name: <image_stream_name>:latest
        namespace: <image_stream_project>
  template:
...output omitted...
```

```
# Use the oc set triggers command on an existing deployment configuration object:
oc set triggers dc/<dc_name> --from-image=<image_stream_project>/<image_stream_name>:latest -c
    <container_name>
# Use the oc import-image command to create an image stream object from an image in a remote repository
    (and poll regularly):
oc import-image <image_stream_name> --from=<registry_full_host_name>/<namespace>/<image_name> --confirm
    --scheduled=true
# Follow the triggering of new deployments (similar to 'Events' column in oc describe dc <dc_name>):
oc rollout history dc/<dc_name>
```

CUSTOMIZING OPENSHIFT PIPELINES

1. Clone a Git repository and execute a Maven build and installation:
   ```
   text   stage('Build App') {     steps {       git url: 'http://services.lab.example.com/bookstore'
         sh "mvn install -DskipTests=true"     }   }
   ```

2. Submit the code for analysis to a SonarQube instance:
   ```
   text   stage('Code Analysis') {     steps {       script {         sh "mvn sonar:sonar -Dsonar.host.url=http
   -DskipTests=true"       }     }   }
   ```

3. Input command asks the user for confirmation:
   ```
   text   stage('Promote to QA?') {     steps {       timeout(time:15, unit:'MINUTES') {         input
   message: "Promote to QA?", ok: "Promote"       }     }   }
   ```

4. Pipeline can also make calls to OpenShift - this rolls out the deployment of the application latest image in the `bookstore-qa` project:
   ```
   text   stage('Deploy QA') {     steps {       script {         openshift.withCluster() {           openshift
                openshift.selector('dc', 'bookstore').rollout().latest()           }         }       }
       }   }
   ```

LAB 5.1

```
# Use the Red Hat Quay Enterprise web interface to create the operator2 user account and set its password
    to redhat123. Also create the deploymentpolicy organization and a robot account named openshift with
    read access to the organization's repositories. In a later step, you configure OpenShift to use this
    account to authenticate with Quay. This way, you do not expose the credentials of a real user account.
```

```
podman login quay.apps.lab.example.com
# Push the local /home/student/DO425/labs/deployment-policy/v1/intranetphpv1.tar image to Quay, under the
    deploymentpolicy organization, use GPG key operator2@lab.example.com:
skopeo copy --sign-by operator2@lab.example.com
    docker-archive:/home/student/DO425/labs/deployment-policy/v1/intranetphpv1.tar
    docker://quay.apps.lab.example.com/deploymentpolicy/intranetphp
vim /etc/containers/policy.json                                          # limit registries
{
    "default": [
        {
            "type": "reject"
        }
    ],
    "transports": {
        "atomic": {
            "172.30.13.172:5000": [
                {
                    "type": "insecureAcceptAnything"
                }
            ]
        },
        "docker": {
            "quay.apps.lab.example.com/admin": [
                {
                    "type": "insecureAcceptAnything"
                }
            ],
            "quay.apps.lab.example.com/deploymentpolicy": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/containers/operator2.pub"
                }
            ],
            "registry.lab.example.com": [
                {
                    "type": "insecureAcceptAnything"
                }
            ]
        }
    }
}
scp /etc/containers/policy.json node1:/etc/containers/policy.json
scp /etc/containers/policy.json node2:/etc/containers/policy.json
gpg2 --armor --export operator2@lab.example.com > operator2.pub          # extract a copy of operator2's
    public key and distribute it everywhere
scp operator2.pub root@master:/etc/containers/                           # must match keyPath
scp operator2.pub root@node1:/etc/containers/                            # keyPath
scp operator2.pub root@node2:/etc/containers/                            # keyPath
vim quay.yaml
  docker:
    quay.apps.lab.example.com:                                           # URL of the host with detached sigs
      sigstore: http://signatures.lab.example.com
scp quay.yaml root@master:/etc/containers/registries.d/
scp quay.yaml root@node1:/etc/containers/registries.d/
scp quay.yaml root@node2:/etc/containers/registries.d/
oc login -u developer -p redhat https://master.lab.example.com
oc new-project deployment-policy
oc create -f /home/student/Downloads/deploymentpolicy-openshift-secret.yml # create the secret that
    contains the credentials to authenticate to Quay
oc secrets link serviceaccount/default secret/deploymentpolicy-openshift-pull-secret --for=pull # associate
    the secret to the default service account in the deployment-policy project
```

```
cat ~/DO425/labs/deployment-policy/intranetphp.yaml                    # file creates the intranetphp
    application, notice that the deployment uses your image in Quay (intranetphp:latest)
oc create -f ~/DO425/labs/deployment-policy/intranetphp.yaml
oc get pods
oc get routes
curl intranetphp-deployment-policy.apps.lab.example.com                # version 1
oc set triggers dc/intranetphp                                         # only the configuration change
    trigger is defined (TYPE: config)
# Create an image stream to monitor the deploymentpolicy/intranetphp image in Quay. Add the
    --scheduled=true option to instruct OpenShift to regularly poll Quay to detect new versions of the
    image:
oc import-image intranetphp --from=quay.apps.lab.example.com/deploymentpolicy/intranetphp --confirm
    --scheduled=true
oc set triggers dc/intranetphp --from-image=deployment-policy/intranetphp:latest -c intranetphp
oc set triggers dc/intranetphp                                         # image trigger is there now
podman login quay.apps.lab.example.com
skopeo copy --sign-by operator2@lab.example.com
    docker-archive:/home/student/DO425/labs/deployment-policy/v2/intranetphpv2.tar
    docker://quay.apps.lab.example.com/deploymentpolicy/intranetphp
oc rollout history dc/intranetphp                                      # new deploy automatically triggered,
    because of the new version of intranetphp
curl intranetphp-deployment-policy.apps.lab.example.com                # version 2
podman logout quay.apps.lab.example.com
oc delete project deployment-policy
```

SECURITY CONTEXT CONSTRAINTS (SCCs)

- SCCs define conditions that a pod must satisfy in order to be created
- Similar to policies, which enforce certain actions or prevent others from a service or a user (or service account)
- By default resources get `restricted` SCC (no root, mknod, setuid)
- Create your own SCC rather than modifying a predefined SCC

Control:

1. Privileged mode - running privileged containers should be avoided at all costs
2. Privileges escalation - on/off privileges escalation inside a container
3. Linux capabilities - Linux capabilities to and from your containers (e.g. KILL)
4. Seccomp profiles - allow or block certain system calls (e.g. CAP_CHOWN)
5. Volume types - permit or prevent certain volume types (e.g. emptyDir)
6. System control (Sysctl) settings - modify kernel parameters at runtime
7. Host resources - permit or prevent a pod for accessing the following host resources: IPC namespaces, host networks, host ports, and host PID namespaces
8. Read-only root file system - forces users to mount a volume if they need to store data
9. User and group IDs - restricting users to a certain set of ID or GIDs. Each project gets assigned its own range, as defined by a project annotation, such as `openshift.io/sa.scc.uid-range=1000190000/10000` and `openshift.io/sa.scc.supplemental-groups=1000190000/10000`. `/`=number of allowed values (e.g. 1000190000 up to 1000200000).
10. SELinux labels - define an SELinux label to the pods
11. File system groups - allows you to define supplemental groups for the user, which is usually required for accessing a block device

Introducing SCC Strategies

Categories:

1. Boolean (`Allow Privileged: true`)

2. Allowable set (`RequiredDropCapabilites: KILL,MKNOD,SETUID,SETGID`)

3. Controlled by a strategy, SCC Strategies:

   (a) `RunAsAny` - any ID defined in the pod definition (or image) is allowed (security issue), no SELinux labels

   (b) `MustRunAsRange` - the project or the pod must provide a range within an allowable set, lowest value is default

   - Run As User Strategy:
     `text  MustRunAsRange  UID: <none>  UID Range Min: 10000000  UID Range Max:10000100`

   (c) `MustRunAs` - project or the pod must provide a single value, for example SELinux context

Managing Supplemental Groups - shared storage example (e.g. NFS)

```
# You should not be running containers as root, only containers with an UID of 100001 or members of the
    group with the ID of 100099 are permitted to access the NFS storage:
oc describe demo-project
  ...output omitted...
  Annotations: openshift.io/sa.scc.supplemental-groups=1000190000/10000 # default range in OpenShift
  ...output omitted...
ls -lZ /opt/app/nfs -d                                                  # NFS server defines a directory
    accessible by the user ID 100001 and the group ID 100099
```

Annotation is used by OpenShift to determine the range for supplemental groups.

I. One way to allow access to the NFS share is to be explicit in the pod definition by defining a supplemental groups that all containers inherit. All containers that are created in the project are then members of the group 100099, which grants access to the volume, regardless of the container's user ID:

```
apiVersion: v1
kind: Pod
...output omitted...
spec:
  containers:
  - name: application
  securityContext:
    supplementalGroups: [ 100099 ]                                      # arra of supplemental group(s)
...output omitted...
```

```
oc rsh application-pod id
  uid=1000190000 gid=0(root) groups=0(root),100099,1000190000
# You must also ensure that the value is within the allowable range. This is enforced if the strategy for
    supplemental groups is set to MustRunAsRange. You must then ensure that the SCC applied to the pod
    allows this value.
```

II. Another solution is the creation of a custom SCC that defines a range for the group IDs, enforces the usage of a value inside the range, allows the GID 100099:

```
kind: SecurityContextConstraints
...output omitted...
priority: 10                                                            # larger number = higher priority
supplementalGroups:
  type: MustRunAs                                                       # strategy
  ranges:                                                               # allowed range
  - min: 100050
    max: 100100
...output omitted...
```

Managing File System Groups - block storage example (e.g. iSCSI)

Unlike shared storage, block storage is taken over by a pod, which means that the user and group IDs supplied in the pod definition are applied to the physical block device. If the pod uses a restricted SCC that defines a `fsGroup` with a strategy of `MustRunAs`, then the pod will fail to run. OpenShift doesn't allocate any GID to block storage, so if the pod definition doesn't explicitly set `fsGroup` and SCC uses RunAsAny, permission may still be denied! Define a file system group in the pod definition:

```
apiVersion: v1
  kind: Pod
...output omitted...
spec:
  containers:
  - name: application
  securityContext:
    fsGroup: [ 600 ]                                                    # filesystem group applied to the
        block storage
...output omitted...
```

Managing SELinux context with SCCs

Restricted SCC defines a strategy of `MustRunAs`, the project must define the options, such as user, role, type, and level otherwise pod will not be created

At creation time, OpenShift assigns a SELinux type to containers' main process, `container_runtime_t`

Define the values for the SELinux context in SCC and relationship with the project:

```
oc describe scc restricted
  ...output omitted..
  SELinux Context Strategy: MustRunAs
    User: <none>
    Role: <none>
    Type: <none>
    Level: <none>
oc describe project default
  ...output omitted...
  Annotations: openshift.io/sa.scc.mcs=s0:c1,c0
```

If the pod needs to access a volume, the same categories must be defined for the volume. Define the SELinux context for a pod:

```
apiVersion: v1
kind: Pod
spec:
  containers:
  ...output omitted...
  securityContext:
    seLinuxOptions:
      level: "s0:c123,c456"                                             # sensitivity 0, categories 123,456 =
          same must be on the volume
```

MANAGING SCCS

```
oc describe scc restricted
```

```
text   Name:                                restricted   Priority:                                <none>
  Access:     Users:                              <none>     Groups:                                 system:authentic
         # <-- members of SCC   Settings:     Allow Privileged:                   false     Default Add
Capabilities:           <none>     Required Drop Capabilities:         <none>                          #
<-- e.g. KILL,MKNOD,SETUID,SETGID     Allowed Capabilities:               <none>     Allowed Volume Types:
             awsElasticBlockStore,configMap,emptyDir,iscsi,nfs,persistentVolumeClaim,rbd,secret     Allow
Host Network:                 false                          # <-- access to host network     Allow Host
Ports:                   false     Allow Host PID:                      false     Allow Host IPC:
    Read Only Root Filesystem:          false     Run As User Strategy:               MustRunAsRange
<-- project using this SCC must define range for user IDs       UID:                                 <none>
      UID Range Min:                    <none>       UID Range Max:                     <none>     SELinux
Context Strategy:           MustRunAs       User:                               <none>       Role:
      Type:                             <none>       Level:                             <none>     FSGroup
```

```
Strategy:                   RunAsAny       Ranges:                              <none>     Supplemental Groups
Strategy:       RunAsAny       Ranges:                          <none>
```

Custom SCC: # only cluster admin

```
kind: SecurityContextConstraints
apiVersion: v1
metadata:
  name: custom-scc
allowHostNetwork: True                                          <-- access to the host namespaces and
    network stack
readOnlyRootFilesystem: true                                    <-- read-only file system
requiredDropCapabilities:                                       <-- two Linux capabilities to drop
- KILL
- SYS_CHROOT
seccompProfiles:
- unconfined                                                    <-- can be docker/default, unconfined
    or localhost/profile name (e.g. - localhost/my-restricted-profile) installed in node's SecComp profile
    root dir
runAsUser:
  type: RunAsAny                                                <-- mandatory strategy for the user
seLinuxContext:
  type: RunAsAny                                                <-- mandatory SELinux context
```

```
oc create -f custom-scc.yaml
oc describe scc custom-scc
# In order to use a custom seccomp profile, you need to enable the seccomp-profile-root directive in the
    kubeletArguments section of the nodes' configuration file /etc/origin/node/node-config.yml to point to
    a directory that contains seccomp profiles. The file must be present on all nodes + systemctl restart
    origin-node:
# kubeletArguments:
#   seccomp-profile-root:
#     - "/path/to/profile"
oc delete scc custom-scc
```

Managing Service Accounts for SCCs

Service accounts can be member of an SCC, similarly to users. This restricts all resources created by a service account to inherit the restrictions of the SCC. By default, pods run with the default service account, unless you specify a different service account. All authenticated users are automatically added to the system:authenticated group. As such, all authenticated users inherit the restricted SCC:

```
oc describe scc restricted | grep 'Groups:'                     # system:authenticated
```

If a container requires elevated privileges or special privileges, create a new service account and make it member of an existing SCC, or create your own SCC and make the service account member of that SCC. Every service account has an associated user name, so it can be added to any specific SCC.

Creating a custom service account and make it member of the anyuid SCC, which allows pods to use any UID:

```
oc create serviceaccount app-sa
oc adm policy add-scc-to-user anyuid -z app-sa                  # -z=service account in the current
    project
# If you need to add a SCC to the service account from another project (demo-project):
oc adm policy add-scc-to-user anyuid system:serviceaccount:demo-project:app-sa
# Update an existing deployment configuration or pod definition:
  ...output omitted...
  spec:
    containers:
      - image: registry-demo.lab.example.com/my-app:latest
        name: my-app
    spec:
      serviceAccountName: app-sa                                # <--
```

LAB 5.2

```
oc login -u admin -p redhat https://master.lab.example.com
oc get scc | grep restricted                                          # inspect existing SCCs
oc describe scc restricted                                            # shows members (users & groups)
oc get clusterrolebinding | awk 'NR == 1 || /system:authenticated/'   # list all cluster role bindings that
    apply for the system:authenticated group (this shows which capabilities authenticated user have)
  NAME                                           ROLE
  basic-users                                    /basic-user
  cluster-status-binding                         /cluster-status
  self-access-reviewers                          /self-access-reviewer
  self-provisioners                              /self-provisioner   # <-- allows to create projects
  servicecatalog-serviceclass-viewer-binding     /servicecatalog-serviceclass-viewer
  system:basic-user                              /system:basic-user
  system:build-strategy-docker-binding           /system:build-strategy-docker
  system:build-strategy-jenkinspipeline-binding  /system:build-strategy-jenkinspipeline
  system:build-strategy-source-binding           /system:build-strategy-source
  system:discovery                               /system:discovery
  system:oauth-token-deleters                    /system:oauth-token-deleter
  system:openshift:discovery                     /system:openshift:discovery
  system:scope-impersonation                     /system:scope-impersonation
  system:webhooks                                /system:webhook
oc login -u developer -p redhat https://master.lab.example.com
oc new-project deployment-scc
oc describe project deployment-scc                                    # verify that projects provide a
    default range 1000130000 up to 1000139999 (as openshift.io/sa.scc.uid-range annotation)
# To ensure that the UID allocated to containers are in the range, create a deployment configuration by
    using the file located at ~/DO425/labs/deployment-scc/deployment-scc.yaml:
oc create -f ~/DO425/labs/deployment-scc/deployment-scc.yaml          # creates 1 pod in deployment-scc
    project
oc get pods                                                           # deployment-scc-1-wkv5r
oc rsh deployment-scc-1-wkv5r id                                      # uid=1000130000 gid=0(root)
    groups=0(root),1000130000
# Create a service account and edit the deployment configuration to define the new service account and to
    force the UID to allocate:
oc create sa deployment-scc-sa                                        # create service account in project
oc get sa                                                             # shows deployment-scc-sa
oc login -u admin -p redhat
oc describe scc anyuid | grep 'Run As' -A3                            # RunAsAny
oc adm policy add-scc-to-user anyuid -z deployment-scc-sa
oc login -u developer -p redhat
oc edit dc/deployment-scc
  ...output omitted...
    spec:
      serviceAccount: deployment-scc-sa                               # use the SA we created
      containers:
      - image: registry.lab.example.com/rhscl/httpd-24-rhel7:latest
        securityContext:                                              # <--
          runAsUser: 600                                              # <-- ID of 600 for the pod's user
oc get pods                                                           # deployment-scc-2-d7hj5 running
oc rsh deployment-scc-2-d7hj5 id                                      # uid=600 gid=0(root) groups=0(root)
# User is no longer member of any supplemental group, as the pod overrides the project settings but does
    not define any supplemental group
cat ~/DO425/labs/deployment-scc/deployment-scc-host-mount.yaml        # mount a host volume
  ...output omitted...
  spec:
    volumes:
      - name: httpd
          hostPath:
        path: /opt/DO425/deployment-scc/
  ...output omitted...
ssh root@node1 ls /opt/DO425/deployment-scc                           # Hello.txt
```

```
oc create -f ~/DO425/labs/deployment-scc/deployment-scc-host-mount.yaml
oc get pods                                                    # deployment-scc-host-mount-1-deploy
    is running but NOT working!
oc get events                                                  # hostPath volumes are not allowed to
    be used
oc create sa deployment-scc-host-sa
oc login -u admin -p redhat
oc describe scc hostaccess                                     # "Allowed Volume Types: ...,
    hostPath, ..." default SCC that shows all the allowed plugins, also MustRunAsRange
oc adm policy add-scc-to-user hostaccess -z deployment-scc-host-sa    # attach this SCC to the
    deployment-scc-host-sa service account
# Another way of updating a deployment configuration is by using the oc patch verb:
oc login -u developer -p redhat
oc patch dc/deployment-scc-host-mount -p '{"spec":{"template":{"spec":{"serviceAccountName":
    "deployment-scc-host-sa"}}}}' # or 'oc edit'
oc get pods                                                    # deployment-scc-hot-mount-2-pvs55
oc rsh deployment-scc-host-mount-2-pvs55
  mount | grep www                                             # that is where it is mounted
  ls /var/www/html                                             # Hello.txt
  exit
# Delete the service accounts and the project:
oc login -u admin -p redhat
oc adm policy remove-scc-from-user anyuid -z deployment-scc-sa        # remove SCC from SA
oc adm policy remove-scc-from-user hostaccess -z deployment-scc-host-sa
oc delete project deployment-scc
```

FINAL LAB 5

```
TBA
```

6. Managing Secure Platform Orchestration

MANAGING APPLICATION HEALTH

1. Liveness Probe - is the pod healthy?

2. Readiness Probe - is the pod ready to serve requests?

MANAGING APPLICATION SCHEDULING

Scheduler filters the list of running nodes by the availability of node resources, such as host ports or memory

A common use for affinity rules is to schedule related pods to be close to each other for performance reasons.

A common use case for anti-affinity rules is to schedule related pods not too close to each other for high availability reasons.

Rules can be: mandatory (required) or best-effort (preferred)

Define 8 nodes, two regions, us-west and us-east, and a set of two zones in each region:

```
oc label node node1 region=us-west zone=power1a                # west
oc label node node2 region=us-west zone=power1a
oc label node node3 region=us-west zone=power2a
oc label node node4 region=us-west zone=power2a
oc label node node5 region=us-east zone=power1b                # east
oc label node node6 region=us-east zone=power1b
oc label node node7 region=us-east zone=power2b
oc label node node8 region=us-east zone=power2b
oc get nodes -L zone -L region                                 # inspect the labels assigned to nodes
```

Rule that requires the pod be placed on a node with a label whose key is `compute-CA-NorthSouth` and whose value is either `compute-CA-North` or `compute-CA-South`:

```
oc label node9 compute-CA-NorthSouth=compute-CA-North
```

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:                                                    # different from nodeSelector
    nodeAffinity:                                              # defines usage of affinity
      requiredDuringSchedulingIgnoredDuringExecution:          # it requires rule to enforce
          placement
        nodeSelectorTerms:
        - matchExpressions:                                    # must match this
          - key: compute-CA-NorthSouth
            operator: In                                       # In, NotIn, Exists, DoesNotExist,
                Lt, Gt
            values:
            - compute-CA-North                                 # first value that must match
            - compute-CA-South                                 # second value that must match to
                apply the rule
  containers:
  - name: with-node-affinity
    image: registry.access.redhat.com/openshift3/jenkins-2-rhel7
```

Node selector can be part of pod definition or deployment config (the below triggers new deployment):

```
oc patch dc dev-app --patch '{"spec":{"template":{"nodeSelector":{"env":"dev"}}}}' # configure the dev-app
    deployment configuration so that its pods only run on nodes that have the env=dev label
```

Same in YAML:

```
apiVersion: v1
kind: Pod
spec:
  nodeSelector:                                                # different from affinity
    env: dev
  containers:
    - image: registry.lab.example.com/rhscl/httpd-24-rhel7:latest
...output omitted...
```

Node maintenance/availability

```
oc adm manage-node --schedulable=false <node>                  # mark a node as unschedulable
oc adm drain <node> --delete-local-data --ignore-daemonsets    # destroys (evicts) all pods running
    on the node and instructs OpenShift to recreate the pods, --delete-local-data is for emptyDir pods
```

MANAGING RESOURCE USAGE

Define limits in a deployment configuration instead of a pod definition

1. Resource requests - indicate that a pod cannot run with less than the specified amount of resources
2. Resource limits - prevent a pod from using up all compute resources from a node

Managing Quotas

`ResourceQuota` resource object specifies hard resource usage limits for a project; all attributes of a quota are optional, meaning that any resource that is not restricted by a quota can be consumed without bounds (you can restrict e.g. pods, rc, svc, secrets, pvcs, CPU, memory, storage).

`ClusterResourceQuota` resource is created at the cluster level, uses `openshift.io/requester` annotation:

```
oc create clusterquota user-prod --project-annotation-selector openshift.io/requester=production --hard
    pods=12 --hard secrets=20 # create a cluster quota for all projects owned by the production user
oc create clusterquota env-qa --project-label-selector environment=qa --hard pods=10 --hard services=5 #
    create a cluster quota for all the projects with a label of environment=qa
oc delete clusterquota <quota>
```

ResourceQuota resource object that defines limits for CPU and memory:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: my-quota
spec:
  hard:
    limits.memory: "400Mi"                                    # by default quota attributes track
        resource request
    limits.cpu: "1200m"                                       # ...prefix the attribute with
        'limits.' to track resource limits
```

```
oc describe quota
  Name: my-quota
  Namespace:  dev-env                                         # If a quota that restricts the use
      of compute resources for a project is set, OpenShift refuses to create pods that do not specify
      resource requests or resource limits
  Resource      Used  Hard
  --------      ----  ----
  limits.cpu    400m  1200m                                   # millicores
  limits.memory 128Mi 400Mi                                   # mebibytes (1MiB=1,048,576b)
```

Managing Limit Ranges

`LimitRange` resource, also called a limit, defines the default, minimum, and maximum values for compute resource requests and limits (also storage - default, min, max capacity requested by image, is, pvc) for a single pod or for a single container defined inside the project. A resource request or limit for a pod is the sum of its containers.

To understand the difference between a limit range and a resource quota resource, consider that a limit range defines valid ranges and default values for a single pod, while a resource quota defines only maximum values for the sum of all pods in a project.

The following listing shows a limit range defined using YAML syntax:

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "limit-range"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "400m"                                           # maximum CPU allowed across all
            containers in a pod
        memory: "256Mi"
      min:
        cpu: "100m"
        memory: "64Mi"
    - type: "Container"
      max:
        cpu: "400m"                                           # maximum CPU allowed per container
        memory: "256Mi"                                       # maximum memory allowed per container
      min:
        cpu: "100m"
        memory: "64Mi"
      default:
        cpu: "200m"
        memory: "64Mi"
      defaultRequest:
        cpu: "200m"
        memory: "64Mi"
    - type: "PersistentVolumeClaim"
      min:
        storage: 2Gi
      max:
```

```
      storage: 50Gi                                                    # maximum capacity of the volume that
          can be requested by one claim
    - type: openshift.io/Image                                         # maximum size of an image that can
      be pushed to OCR
      max:
        storage: 1Gi
```

```
oc create -f limits.yml
oc get limits
  Name:       limit-range
  Namespace:  dev-project
  Type        Resource  Min   Max    Default Request  Default Limit
  ----        --------  ---   ---    ---------------  -------------
  Pod         cpu       100m  400m   -                -
  Pod         memory    64Mi  256Mi  -                -
  Container   cpu       100m  400m   200m             200m
  Container   memory    64Mi  256Mi  64Mi             64Mi
```

After creating a limit range in a project, all resource create requests are evaluated against each limit range resource in the project. If the new resource violates the minimum or maximum constraint enumerated by any limit, OpenShift rejects the resource. If the new resource does not set an explicit value, and the constraint supports a default value, then the default value is applied to the new resource as its usage value.

LAB 6.1

```
oc login -u admin -p redhat https://master.lab.example.com
oc get nodes -L region                                                 # region label has no value on any
    node (doesn't exist)
oc login -u developer -p redhat https://master.lab.example.com
oc new-project orchestration-scheduling
oc create -f ~/DO425/labs/orchestration-scheduling/dc-scale.yaml
oc get pods -o=custom-columns="name:metadata.name,node:.spec.nodeName" # orchestration-scheduling-1-g29wh
    on node2
oc scale dc orchestration-scheduling --replicas=6
oc get pods -o=custom-columns="name:metadata.name,node:.spec.nodeName" # pods are spread across the two
    application nodes 1 & 2
oc login -u admin -p redhat
oc label node node2.lab.example.com region=apps --overwrite=true
oc get nodes -L region                                                 # label 'apps' is applied to node2
oc login -u developer -p redhat
oc edit dc/orchestration-scheduling                                    # Update the spec section of the
    template group, and not the spec section of the metadata group
  ...output omitted...
      spec:
          nodeSelector:
            region: apps
          containers:
          - image: registry.lab.example.com/rhscl/httpd-24-rhel7:latest
  ...output omitted...
oc get pods -o=custom-columns="name:metadata.name,node:.spec.nodeName" # all pods should be scheduled on
    node2 only
oc login -u admin -p redhat
oc label node node1.lab.example.com region=apps --overwrite=true
oc get nodes -L region                                                 # 'apps' label is applied to node1
    and node2
oc adm manage-node --schedulable=false node2.lab.example.com           # mark node2 unschedulable
oc adm drain node2.lab.example.com --delete-local-data --ignore-daemonsets # drain all pods from node2
oc get pods -o=custom-columns="name:metadata.name,node:.spec.nodeName" # all the application pods are
    running on node1
oc adm manage-node --schedulable=true node2.lab.example.com            # node2 schedulable again
oc scale dc orchestration-scheduling --replicas=2                      # scale down to two
```

```
# Set resource quotas and limit ranges for your project, then verify that the project's pods consume
    resources from the project's quota:
cat ~/DO425/labs/orchestration-scheduling/quota.yaml
  apiVersion: v1
  kind: ResourceQuota
  metadata:
      name: orchestration-scheduling
  spec:
      hard:
          limits.memory: "400Mi"
          limits.cpu: "1200m"
oc create -f ~/DO425/labs/orchestration-scheduling/quota.yaml
oc describe quota                                                    # limits.memory used 128Mi, hard 400Mi
cat ~/DO425/labs/orchestration-scheduling/limit-range.yaml
  apiVersion: "v1"
  kind: "LimitRange"
  metadata:
    name: "orchestration-scheduling"
  spec:
    limits:
      - type: "Pod"
        max:
          cpu: "400m"
          memory: "256Mi"
        min:
          cpu: "100m"
          memory: "64Mi"
      - type: "Container"
        max:
          cpu: "400m"
          memory: "256Mi"
        min:
          cpu: "100m"
          memory: "64Mi"
        default:
          cpu: "200m"
          memory: "64Mi"
        defaultRequest:
          cpu: "200m"
          memory: "64Mi"
oc create -f ~/DO425/labs/orchestration-scheduling/limit-range.yaml
oc describe limits
  Name:       orchestration-scheduling
  Namespace:  orchestration-scheduling
  Type        Resource  Min   Max    Default Request  Default Limit
  ----        --------  ---   ---    ---------------  -------------
  Pod         cpu       100m  400m   -                -
  Pod         memory    64Mi  256Mi  -                -
  Container   cpu       100m  400m   200m             200m
  Container   memory    64Mi  256Mi  64Mi             64Mi
oc describe node node1.lab.example.com | grep -A 5 Allocated         # resources usage on node1
  Allocated resources:
    (Total limits may be over 100 percent, i.e., overcommitted.)
    Resource  Requests      Limits
    --------  --------      ------
    cpu       595m (29%)    525m (26%)
    memory    4916Mi (86%)  3864Mi (67%)
oc login -u developer -p redhat
oc get pods
oc describe pod orchestration-scheduling-2-49flz| grep -A2 Requests   # requests CPU 200m, memory 64Mi
oc scale dc orchestration-scheduling --replicas=3                    # memory = 3x 64Mi (192)
oc describe quota                                                    # limits.memory Used 192Mi, Hard 440Mi
```

```
oc scale dc orchestration-scheduling --replicas=7                    # memory = 7x 64Mi (448)
oc get pod                                                           # seventh pod is never created
    because the sum of the pod memory exceeds 400 MiB
oc describe dc orchestration-scheduling | grep Replicas              # Replicas 6 current/7 desired
oc get events | grep -i error                                        # exceeded quota
  requested: limits.cpu=200m,limits.memory=64Mi,
  used: limits.cpu=1200m,limits.memory=384Mi,
  limited: limits.cpu=1200m,limits.memory=400Mi
oc scale dc orchestration-scheduling --replicas=1
oc get pod                                                           # ensure only 1 pod running
oc describe quota                                                    # limits.memory 64Mi used, Hard 400Mi
oc set resources dc orchestration-scheduling --requests=cpu=100m     # Resource requests are not enforced
    by any quota
oc get pod                                                           # orchestration-scheduling-3-4594c
oc describe pod orchestration-scheduling-3-4594c | grep -A2 Requests # Requests cpu: 100m
oc describe quota                                                    # CPU request is not evaluated
    against the project's quota
oc set resources dc orchestration-scheduling --requests=memory=16Gi  # Invalid value: "16Gi": must be less
    than or equal to memory limit
cat ~/DO425/labs/orchestration-scheduling/dc-secret.yaml
  apiVersion: v1
  kind: List
  items:
  - apiVersion: apps.openshift.io/v1
    kind: DeploymentConfig
  ...output omitted...
    containers:
      - image: registry.lab.example.com/rhscl/mongodb-32-rhel7:latest
        name: mongodb
        env:
          - name: MONGODB_ADMIN_PASSWORD
            valueFrom:
              secretKeyRef:
                name: orchestration-secret
                key: admin_password
          - name: MONGODB_USER
            valueFrom:
              secretKeyRef:
                name: orchestration-secret
                key: username
          - name: MONGODB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: orchestration-secret
                key: password
  ...output omitted...
cat ~/DO425/labs/orchestration-scheduling/secret.yaml
  apiVersion: v1
  kind: Secret
  metadata:
    name: orchestration-secret
  type: Opaque
  stringData:
    admin_password: redhat123
    username: developer
    password: redhat
oc create -f ~/DO425/labs/orchestration-scheduling/secret.yaml
oc describe secret orchestration-secret                              # shows admin_password, username,
    password sizes
oc create -f ~/DO425/labs/orchestration-scheduling/dc-secret.yaml    # deploment config
oc get pods                                                          # orchestration-secrets-1-fs5n5
```

```
oc set env pod orchestration-secrets-1-fs5n5 --list                    # list environment variables that pod
    uses
oc rsh orchestration-secrets-1-fs5n5
  mongo -u $MONGODB_USER -p $MONGODB_PASSWORD $MONGODB_DATABASE        # connecting to orchestration-secrets
  exit
  mongo -u admin -p redhat123 admin --eval "db.version()"              # MongoDB shell version is displayed
  exit
oc delete project orchestration-scheduling
```

You can inspect the RESTful API calls:

```
oc create configmap special-config --from-literal=serverAddress=172.20.30.40 --loglevel=8 # 0 to 10
# Notice how the oc command sends the new resource details through a JSON object in the request body
```

LAB 6.2

```
oc login -u admin -p redhat https://master.lab.example.com
oc project tracking
oc get pods                                                            # bus-1-xtjlw
oc get svc                                                             # bus, ClusterIP 172.30.56.233, port
    8080/TCP
oc get route                                                           # none, not accessible from outside
    the cluster
curl http://bus.tracking.svc.cluster.local:8080/bustrack.php/api/v1/stations/ # could not resolve
ssh master curl http://bus.tracking.svc.cluster.local:8080/bustrack.php/api/v1/stations/ # returns "bus
    stations" JSON
firefox https://3scale-admin.apps.lab.example.com/
  1. Log in as admin/redhat
  2. Name: Bus Tracking
     Base URL: http://bus.tracking.svc.cluster.local:8080/bustrack.php
  3. GET method: /api/v1/stations
  4. Send request.
curl -k
    "https://api-3scale-apicast-staging.apps.lab.example.com:443/api/v1/stations/?user_key=a1b1dfeb9d7db608cf8477
# To protect the back-end application, limit the number of requests to five per minute. Hit API multiple
    times to confirm.
```

OPENSHIFT SECURITY MODEL

Infrastructure components, such as application nodes, use client certificates that OpenShift generates. Infrastructure components that run in containers use a token associated with their service account to connect to the API.

OpenShift Container Platform creates the PKI and generates the certificates at installation time. OpenShift uses an internal CA (`openshift-signer`) to generate and sign all the certificates that are listed in the master-config.yaml configuration file. This can be overriden in Ansible:

```
openshift_master_named_certificates=[{"certfile": "/path/to/certificate.crt", "keyfile":
    "/path/to/private-key.key", "cafile": "/path/to/ca.crt"}]
```

To override names:

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"certfile": "/path/to/certificate.crt", "keyfile":
    "/path/to/private-key.key", "names": ["public-master-host.com"], "cafile": "/path/to/ca.crt"}]
```

On the master server, there are many certificates and keys that are generated at installation time:

```
ls -l /etc/origin/master/*.{crt,key}
  additional_ca.crt                                                    # + .key
  ca-bundle.crt                                                        # + .key
  etcd.server.crt                                                      # + .key
  master.kubelet-client.crt                                            # + .key
  registry.crt                                                         # + .key
  . . .
```

LAB 6.3

```
ssh root@master
cd /etc/origin/master
vim master-config.yaml
  ...output omitted...
  proxyClientInfo:
    certFile: master.proxy-client.crt
    keyFile: master.proxy-client.key
  ...output omitted...
  oauthConfig:
  ...output omitted...
    masterPublicURL: https://master.lab.example.com:443
openssl x509 -in ca.crt -text -noout                                    # locate the issuer of default CA
  Issuer: CN=openshift-signer@1234567890
  ...output omitted...
  CA:TRUE
openssl x509 -in etcd.server.crt -text -noout                           # same openshift-signer, CA:FALSE
exit
ssh root@node1
cd /etc/origin/node
ls -l                                                                   # certificates/, client-ca.crt,
    node-config.yaml (grep -A1 -B2 certificates node-config.yaml)
openssl x509 -in client-ca.crt -text -noout                             # CA generates and signs all
    certificates for the nodes (CA: TRUE)
openssl x509 -in certificates/kubelet-client-current.pem -text -noout # CN common name, embeds the node
    name in the system:nodes organization (CN=system:node:node1.lab.example.com)
exit
oc login -u admin -p redhat https://master.lab.example.com
oc project openshift-console
oc get routes                                                           # reencrypt/Redirect
oc get service                                                          # 172.30.11.194, console
oc describe secret console-serving-cert                                 # secret type kubernetes.io/tls
    embeds service-signer.crt and service-signer.key
firefox https://console.apps.lab.example.com                            # openshift-signer is the CA, which
    signs the certificate that is valid for the *.apps.lab.example.com wildcard subdomain
oc get pods -o wide | awk '{print $7}'                                  # name of the nodes on which console
    pod is = master.lab.example.com
ssh root@master
ip r                                                                    # locate device that routes
    172.30.0.0/16 = tun0
tcpdump -ni tun0 -vvvs 1024 -l -A "tcp port 443 and src host 172.30.0.1"
  firefox https://console.apps.lab.example.com                          # log in as developer/redhat,
      everything is encrypted so no packets
  Ctrl+C
oc login -u developer -p redhat https://master.lab.example.com
oc new-project orchestration-certificates
oc new-app ~/DO425/labs/orchestration-certificates/app.yaml             # two httpd pods, each on different
    node
oc get pods -o wide                                                     # IP address of the pod running on
    node2 is 10.129.0.101
oc rsh httpd-node1 bash                                                 # at the same time, run tcpdump on
    node1: tcpdump -ni vxlan_sys_4789 -vvvs 1024 -l -A "src host 10.129.0.101" - you will see some activity
    on the terminal while you do the below - we're capturing traffic between two pods on different nodes
  exec 3<>/dev/tcp/10.129.0.101/8080                                    # the IP address of the second pod
echo -e "GET / HTTP/1.1\r\nhost: 10.129.0.101 \r\nConnection: close\r\n\r\n" >&3 # fetch the web server and
    print its content. node1's tcpdump will show raw packets
Ctrl+C
  exit
oc delete project orchestration-certificates
```

FINAL LAB 6

```
TBA
```

7. Providing Secure Network I/O

ISTIO

Sidecars sit alongside microservices and route requests to other proxies. These components form a mesh network.

- Definition of a `VirtualService`:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews-route
spec:
  hosts:
  - production.prod.svc.cluster.local
  http:
  - route:
    - destination:
        host: production-v2.prod.svc.cluster.local
        subset: v2
      weight: 25
    - destination:
        host: production-v1.prod.svc.cluster.local
        subset: v1
      weight: 75
```

- The `DestinationRule` associated with this virtual service:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: production-destination
spec:
  host: production.prod.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

MANAGING SECURE TRAFFIC IN OPENSHIFT CONTAINER PLATFORM

OpenShift can manage certificates by generating an X.509 certificate and a key pair as a secret in your application name spaces. Certificates are valid for the internal DNS name `service_name.namespace.svc`.

The following Red Hat Single Sign-on template shows how the service resource defines the annotation `service.alpha.openshift.io/servi` for generating the certificate with a value of `sso-x509-https-secret`.

The pod mounts the volume that contains this certificate in `/etc/x509/https`, as referenced by `secretName: sso-x509-https-secret` in the volumes section:

```
kind: Template
apiVersion: v1
...output omitted...
objects:
- kind: Service                                                    # <--
  spec:
  metadata:
    annotations:
      service.alpha.openshift.io/serving-cert-secret-name: sso-x509-https-secret # <--
- kind: DeploymentConfig                                          # <--
```

```
...output omitted...
  spec:
    template:
      spec:
        containers:
        - name: "${APPLICATION_NAME}"
          image: "${APPLICATION_NAME}"
...output omitted...
          volumeMounts:
          - name: sso-x509-https-volume
            mountPath: "/etc/x509/https"
            readOnly: true
          volumes:
          - name: sso-x509-https-volume
            secret:
              secretName: sso-x509-https-secret                    # <--
```

Service serving certificates allow a pod to mount the secret and use them accordingly. The certificate and key are in PEM format, and stored as `tls.crt` and `tls.key`.

OpenShift automatically replaces the certificate when it gets close to expiration.

Pods can use these security certificates by reading the CA bundle located at `/var/run/secrets/kubernetes.io/serviceaccount/service` which is automatically exposed inside pods.

If the service certificate generation fails, force certificate regeneration by removing the old secret, and clearing the following two annotations on the service:

```
service.alpha.openshift.io/serving-cert-generation-error-<error_number>
service.alpha.openshift.io/serving-cert-generation-error-num-<error_number>
```

The service serving certificates are generated on-demand, and thus are different from those used by OpenShift for node-to-node or node-to-master communication.

Managing Network Policies in OpenShift

3 SDNs:

1. `ovs-subnet` - flat network that spreads across all the cluster nodes and connects all the pods

2. `ovs-multitenant` - isolates each OpenShift project. By default, the pods in a project cannot access pods in other projects. The following command allows projectA to access pods and services in projectB, and vice versa: `oc adm pod-network join-projects --to=projectA projectB` - this give access to all pods & services in project.

3. `ovs-networkpolicy` - To use network policies, you need to switch from the default SDN provider to the `redhat/openshift-ovs-netw` provider. It allows you to create tailored policies between projects to make sure users can only access what they should (which conforms to the least privilege approach). By default, without any network policy resources defined, pods in a project can access any other pod.

Example: Both networks are separate projects - The following network policy, which applies to all pods in network-A, allows traffic from the pods in network-B whose label is `role="back-end"`, but blocks all other pods:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: network-A_policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: back-end
  - from:
    - namespaceSelector:
        matchLabels:
          name: network-B
```

Example: Both networks are separate projects - The following network policy, which applies to network-B, allows traffic from all the pods in network-A. This policy is less restrictive than the network-A policy, because it does not restrict traffic on any pods on the network-A project:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: network-B_policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector: {}
  - from:
    - namespaceSelector:
        matchLabels:
          name: network-A
```

The following excerpt shows how to allow external users to access an application whose labels match `product-catalog` over a TCP connection on port 8080:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: external-access
spec:
  podSelector:
    matchLabels:
      app: product-catalog
  ingress:
  - ports:
    - protocol: TCP
      port: 8080
```

```
ovs-ofctl dump-flows br0 -O OpenFlow13 --no-stats                    # flow rules on the Open vSwitch
    bridge on the node
```

The following network policy allows traffic coming from pods that match the `emails` label to access a database whose label is `db`:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-3306
spec:
  podSelector:
    matchLabels:
      app: db                                                    # destination
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: emails                                            # source
    ports:
    - protocol: TCP
      port: 3306
```

You can also define a default policy for your project. An empty pod selector means that this policy applies to all pods in this project. The following default policy blocks all traffic unless you define an explicit policy that overrides this default behavior:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: default-deny
spec:
  podSelector:
```

To manage network communication between two projects, assign a label to the project that needs access to another project:

```
oc label namespace front-end project=frontend_project                    # assigns the frontend_project label
    to the front-end project
```

The following network policy, which applies to a `back-end` project, allows any pods in the `front-end` project to access the pods labeled as `app=user-registration` through port 8080, in this `back-end` project:

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-8080-user-registration
spec:
  podSelector:
    matchLabels:
      app: user-registration
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: frontend_project
    ports:
    - protocol: TCP
      port: 8080
```

Node & Pod network

OpenShift configures each cluster node with an Open vSwitch bridge named `br0`. For each pod, OpenShift creates a `veth` device and connects one end to the `eth0` interface inside the pod and the other end to the `br0` bridge:

```
nsenter -t 13120 --net ip link show                                      # run as root on the node, 13120 is
    PID of a process running inside the container. 'eth0@if36:' (connects eth0 inside container to the 36th
    interface on the node) is one of the links in the output.
ip link show | grep ^36:                                                 # retrieve the other end of that veth
    device - e.g. 'veth4f3b73f@if3:'
yum install -y openvswitch                                               # not installed by default, good for
    inspecting the network with ovs-vsctl
ovs-vsctl show | grep veth4f3b73f                                        # shows the 'br0' and all the
    connected ports.
```

`tun0` interface on the node is an Open vSwitch port on the `br0` bridge, it is used for external cluster access

OpenShift uses the vxlan_sys_4789 interface on the node, or vxlan0 in br0, for building the cluster overlay network between nodes. Communications between pods on different nodes go through this interface.

CONTROLLING EGRESS TRAFFIC

By default, OpenShift allows egress traffic with no restrictions. You can control traffic with egress fw, routers, static IP. OpenShift allows traffic if no rule matches, checks rules in order.

1. Egress FWs
   This object allows the egress traffic to the 192.168.12.0/24 network, and to the db-srv.example.com and analytics.example.com systems. The last rule denies everything else. The rules only apply to the egress traffic and do not affect inter-pod communication:

```
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: myfirewall
spec:
  egress:
  - to:
      cidrSelector: 192.168.12.0/24
    type: Allow
  - to:
      dnsName: db-srv.example.com
    type: Allow
```

```
  - to:
      dnsName: analytics.example.com
    type: Allow
  - to:
      cidrSelector: 0.0.0.0/0
    type: Deny
```

```
oc create -f firewall.yaml -n myproject                                    # only one per project, you must be
    cluster admin to create it
```

1. Egress Routers
   3 modes:

   (a) Redirect - image `openshift3/ose-pod` (TCP and UDP)

   (b) HTTP proxy - image `openshift3/ose-egress-http-proxy` (HTTP and HTTPS)

   (c) DNS proxy - image `openshift3/ose-egress-dns-proxy` (TCP)

Can be used with any of the three SDN plugins, but underlying hosting platform may need to be reconfigured. Present a unique identifiable source IP address to the firewall and the external service.

Egress router is a particular pod running in your project with two interfaces (eth0, macvlan0). It acts as a proxy between your pods and the external service.

`macvlan0` interfaces are special devices that directly expose node interfaces to the container and has a MAC address seen by the underlying network.

```
nsenter -t 14171 --net ip addr show                                        # shows eth0@if22 and macvlan0@if2.
    IP address of the macvlan interface, 172.25.250.20 is not used by anything else and is the source IP
    address seen by the external firewall and service
ip link show | grep ^2:                                                    # "2: eth0: ..."
```

In front of each egress router, you need to create an OpenShift service object. You use that service host name inside your application to access the external service through the router.

a/ Example - redirect mode (can only be created by cluster admin & application may need reconfiguration to access external service through egress router):

```
apiVersion: v1
kind: Pod                                                                  # <--
metadata:
  name: egress-router-redirect
  labels:
    name: egress-router-redirect
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:                                                          # <-- starts before application
      container egress-router-wait, creates NAT rules
  - name: egress-router-setup
    image: registry.redhat.io/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE                                                  # source IP address
      value: 172.25.250.20/24
    - name: EGRESS_GATEWAY                                                 # IP of the gateway in the source
        subnet
      value: 172.25.250.254
    - name: EGRESS_DESTINATION                                             # IP address of the external service
      value: 23.21.165.246
    - name: EGRESS_ROUTER_MODE
      value: init
```

```
  containers:
  - name: egress-router-wait                                    # application container
    image: registry.redhat.io/openshift3/ose-pod

apiVersion: v1
kind: Service                                                   # <--
metadata:
  name: egress-router
spec:
  ports:
  - name: postgres
    port: 5432
  - name: pgrest
    port: 80
  type: ClusterIP
  selector:
    name: egress-router-redirect                                # label of the router pod
```

b/ Example - HTTP proxy mode:

```
apiVersion: v1
kind: Pod                                                       # <--
metadata:
  name: egress-router-http
  labels:
    name: egress-router-http
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router-setup
    image: registry.redhat.io/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 172.25.250.20/24
    - name: EGRESS_GATEWAY
      value: 172.25.250.254
    - name: EGRESS_ROUTER_MODE                                  # that is the mode
      value: http-proxy
  containers:
  - name: egress-router-proxy
    image: registry.redhat.io/openshift3/ose-egress-http-proxy  # container image name
    env:
    - name: EGRESS_HTTP_PROXY_DESTINATION                       # Squid rules:
      value: |
        !*.example.com                                          # except to example.com,
        !10.1.2.0/24                                            # or 10.1.2.0 network,
        *                                                       # forward requests to any external
            service

apiVersion: v1
kind: Service                                                   # <--
metadata:
  name: egress-router
spec:
  ports:
  - name: http-proxy
    port: 8080
  type: ClusterIP
  selector:
    name: egress-router-http                                    # label of the router pod
```

c/ Example - DNS proxy mode:

```
apiVersion: v1
kind: Pod                                                          # <--
metadata:
  name: egress-router-dns
  labels:
    name: egress-router-dns
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router-setup
    image: registry.redhat.io/openshift3/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 172.25.250.20/24
    - name: EGRESS_GATEWAY
      value: 172.25.250.254
    - name: EGRESS_ROUTER_MODE                                     # indicates the egress mode
      value: dns-proxy
  containers:
  - name: egress-router-proxy
    image: registry.redhat.io/openshift3/ose-egress-dns-proxy      # container image name
    env:
    - name: EGRESS_DNS_PROXY_DESTINATION                           # HAProxy dameon rules:
      value: |                                                     # source-port destination_host
          [destination-port]
        80  23.21.165.246                                         # forward request coming on port 80
            to 23.21.165.246
        8080 www.example.com 80                                    # forward request coming on 8080 to
            www.example.com on port 80
        2525 10.4.203.49 25

apiVersion: v1
kind: Service                                                      # <--
metadata:
  name: egress-router
spec:
  ports:
  - name: web1
    port: 80
  - name: web2
    port: 8080
  - name: mail
    port: 2525
  type: ClusterIP
  selector:
    name: egress-router-dns                                        # label of the router pod
```

1. Enabling Static IP Addresses for External Access
   You can define a static IP address at the project level, in the `NetNamespace` object. With such a configuration, all the egress traffic from the pods in the project originates from that IP address. OpenShift must use the `ovs-networkpolicy` SDN plug-in.

OpenShift automatically creates one `NetNamespace` object per project. First, associate IP with project and then node:

```
oc patch netnamespace myproject -p '{"egressIPs": ["172.25.250.19"]}' # associates the unused 172.25.250.19
    address to the myproject project
oc get netnamespace myproject
  NAME        NETID     EGRESS IPS
  myproject   13508931  [172.25.250.19]
```

```
oc patch hostsubnet node1 -p '{"egressIPs": ["172.25.250.19"]}'       # associates the 172.25.250.19
    address to node1 (you must indicate on which node OpenShift should assign the address)
oc get hostsubnet node1                                               # shows EGRESS IPS [172.25.250.19]
ssh root@node1 ip addr show dev eth0                                  # OpenShift declares the address as
    an alias on the external network interface of node1
```

When using the `oc patch` command to add a new address to a `HostSubnet` object that already has egress IP addresses defined for other projects, you must also specify those addresses in the `egressIPs` array:

```
oc patch hostsubnet node1 -p '{"egressIPs": ["172.25.250.19", "172.25.250.16", "172.25.250.15"]}' #
    otherwise new address replaces/overwrites all the existing ones
```

LAB 7.1

```
oc login -u developer -p redhat https://master.lab.example.com
oc new-project network-isolation
oc create -f ~/DO425/labs/network-isolation/logic.yaml
oc create -f ~/DO425/labs/network-isolation/presentation.yaml
oc get pods -o wide                                                   # logic-1-5nf56 on node2 &
    presentation-1-sg8cz on node1 running
oc rsh logic-1-5nf56 curl http://services.lab.example.com             # keep cURL running while you tail
    httpd log
ssh root@services.lab.example.com
tail -f /var/log/httpd/access_log                                     # you will see the curl command GET
    from 172.25.250.12 (node2)
oc rsh presentation-1-sg8cz curl http://services.lab.example.com      # run the same curl command from the
    second pod, it will be coming from 172.25.250.11 (node1)
vim ~/DO425/labs/network-isolation/router-redirect.yaml               # Deploy an egress router to access
    the web server on services.lab.example.com (172.25.250.13). Use the available 172.25.250.15/24 IP
  ...output omitted...
  env:                                                                # egress router in redirect mode
  - name: EGRESS_SOURCE
    value: 172.25.250.15/24
  - name: EGRESS_GATEWAY
    value: 172.25.250.254
  - name: EGRESS_DESTINATION
    value: 172.25.250.13                                              # =services.lab.example.com
  - name: EGRESS_ROUTER_MODE
    value: init
  ...output omitted...
oc login -u admin -p redhat                                           # egress router needs admin
oc create -f ~/DO425/labs/network-isolation/router-redirect.yaml      # router pod
oc create -f ~/DO425/labs/network-isolation/router-redirect-service.yaml # router service
oc get pods                                                           # egress-router running
oc rsh logic-1-5nf56 curl http://egress-router.network-isolation.svc.cluster.local # the running tail will
    show request coming from 172.25.250.15
oc describe pod egress-router                                         # config of egress-router shows it is
    running on node2.lab.example.com/172.25.250.12 and the Container ID of 'egress-router-wait' (not the
    init container!) is cri-o://6e56...af33
ssh root@node2.lab.example.com
runc state 6e56...af33 | grep pid                                     # "pid": 22938,
nsenter -t 22938 --net ip addr show                                   # eth0@if34 = 10.129.0.40/23 and
    macvlan0@if2 = 172.25.250.15/24 (if2 = mapped to the node's physical interface whose index is 2)
ip link show | grep ^2:                                               # 2: eth0:
nsenter -t 22938 --net iptables -t nat -nvL                           # inspect NAT rules in the container
  PREROUTING, ACCEPT, target = DNAT (replaces the destination address), in = eth0, destination to =
      172.25.250.13 (services.lab.example.com)
  POSTROUTING ACCEPT, target = SNAT (replaces the source address), out = macvlan0, destination to =
      172.25.250.15
exit
oc get pods                                                           # on workstation, shows:
    presentation-1-sg8cz, logic-1-5nf56, egress-router
```

```
oc describe pod presentation-1-sg8cz                                    # presentation Container ID
    cri-o://4cac...f7f0 running on node1.lab.example.com/172.25.250.11
ssh root@node1.lab.example.com
runc state 4cac...f7f0 | grep pid                                       # "pid": 13120,
nsenter -t 13120 --net ip link show                                     # eth0@if36 = eth0 interface in the
    container is a vEth device whose pair has the index 36 on the node
ip link show | grep ^36:                                                # 36: vetha4f3b73f@if3:
ovs-vsctl show                                                          # Bridge "br0" = "vetha4f3b73f";
    "tun0", "vxlan0" are also connected to "br0"
exit
oc login -u developer -p redhat                                         # you're still in network-isolation
    project
oc rsh presentation-1-sg8cz curl http://logic.network-isolation.svc.cluster.local:8080 # works
oc rsh logic-1-5nf56 curl http://presentation.network-isolation.svc.cluster.local:8080 # works
cat ~/DO425/labs/network-isolation/deny-all.yaml                        # blocks all communication between
    pods in the project
  kind: NetworkPolicy
  apiVersion: extensions/v1beta1
  metadata:
    name: default-deny
  spec:
    podSelector:                                                        # means all the pods in the project
oc create -f ~/DO425/labs/network-isolation/deny-all.yaml
oc rsh presentation-1-sg8cz curl http://logic.network-isolation.svc.cluster.local:8080 # fails
oc rsh logic-1-5nf56 curl http://presentation.network-isolation.svc.cluster.local:8080 # fails
cat ~/DO425/labs/network-isolation/allow-pres2logic.yaml                # allows pods with app=presentation
    to connect to app=logic on 8080
  kind: NetworkPolicy
  apiVersion: extensions/v1beta1
  metadata:
    name: allow-pres-to-logic
  spec:
    podSelector:
      matchLabels:
        app: logic                                                      # destination
    ingress:
    - from:
      - podSelector:
          matchLabels:
            app: presentation                                           # source
      ports:
      - protocol: TCP
        port: 8080
oc create -f ~/DO425/labs/network-isolation/allow-pres2logic.yaml
oc rsh presentation-1-sg8cz curl http://logic.network-isolation.svc.cluster.local:8080 # presentation pod
    can again access the logic pod
oc rsh logic-1-5nf56 curl http://presentation.network-isolation.svc.cluster.local:8080 # logic pod still
    cannot access the presentation pod
oc get pods -o wide                                                     # logic-1-5nf56 = 10.129.0.39
    (node2), presentation-1-sg8cz = 10.130.0.33 (node1)
ssh root@node1.lab.example.com
ovs-ofctl dump-flows br0 -O OpenFlow13 --no-stats | grep nw_src=10.130.0.33,nw_dst=10.129.0.39 #
    nw_src=10.130.0.33,nw_dst=10.129.0.82,tp_dst=8080 = rule that allows the 10.130.0.33 source IP address
    (nw_src) to access the 10.129.0.39 destination IP address (nw_dst)
exit
```

FINAL LAB 7

```
TBA
```

8. Providing Secure Storage I/O

CATEGORIZING STORAGE TYPES IN OPENSHIFT

1. Shared storage - GlusterFS, NFS, Ceph..

2. Block storage - EBS, GCE disk, iSCSI..

Accessing Files in a Shared Storage Type in OpenShift
If you need to access the same share from multiple pods, then you must configure each pod to use a default GID and define the group ownership of the share with a known GID:

```
chgrp test <directory>                                          # configure  dir with known GID
cat /etc/group | grep test                                      # test:x:1012:user
```

To enforce that your pod use a group, you must create a service account in each project. Each service account must be assigned to the same security constraint context (SCC) and it must restrict the limitations to a specific GID. Additionally, because the built-in SCC takes precedence over a custom one, you must set a higher priority in the custom SCC (`kind: SecurityContextConstraints`, `priority: XX`):

```
oc create serviceaccount <sccName> -n <projectName>                    # create the service account that
    configures the project with the custom SCC
oc adm policy add-scc-to-user <sccName> system:serviceaccount:<projectName>:<serviceAccount> # associate
    the service account with the SCC
```

Accessing Files in a Block Storage Type in OpenShift

Any OpenShift cluster can access block storage and even share the contents among pods in the same project. The first pod takes over ownership of the block storage, changing the GID and UID from that share. If any other pod running in the same project tries to access the same persistent volume bound to the block storage, the deployment fails due to lack of permissions.
To solve this problem, you must create a security constraint context that configures the `fsGroup` setting and allows any pod to access the same persistent volume.

LAB 8.1

```
ssh root@services
ls -lZ /exports                                                 # The secure-nfs directory is an NFS
    share that only has read and write permissions to users that belong to the secure-nfs group
grep secure-nfs /etc/group                                      # secure-nfs:x:1200:nfsnobody
showmount -e services.lab.example.com                           # /exports/secure-nfs *
oc login -u admin -p redhat https://master.lab.example.com
oc get pv pv0001                                                # pv0001, 5Gi, RWO
oc describe pv/pv0001                                           # NFS server:
    services.lab.example.com, path: /exports/secure-nfs
oc login -u developer -p redhat
oc new-project secure-nfs
oc new-app postgresql-persistent -p POSTGRESQL_VERSION=9.5 -p VOLUME_CAPACITY=1Gi -o yaml >
    ~/database-secure-nfs.yaml
oc create -f ~/database-secure-nfs.yaml
oc get pvc                                                      # postgresql bound to pv0001
oc get pods                                                     # CrashLoopBackOff error
oc logs postgresql-1-4ptk4                                      # cannot create directory error
oc delete all -l app=postgresql-persistent
vim /home/student/DO425/labs/storage-isolation/restricted-scc.yaml    # create an SCC to enable access
    using the group ID from the secure-nfs group
  # 1. Remove the fields named groups and users so this SCC is not pre-linked to anyone.
  # 2. Change the field named name to secure-nfs.
  # 3. Delete runtime information fields such as creationTimestamp.
  # 4. Also delete all annotations.
  # 5. Change the supplementalGroups field to use MustRunAs, instead of RunAsAny.
  # 6. Set the max value to 1300 and the min value to 1100.
  ...
  defaultAddCapabilities: null
  fsGroup:
    type: MustRunAs
  # delete this: groups:
  # delete this: - system:authenticated
```

```
  kind: SecurityContextConstraints
  metadata:
  # delete all  annotations
  # delete this:   creationTimestamp: null  name: secure-nfs
    name: secure-nfs
  priority: null
  ...
  supplementalGroups:
    type: MustRunAs
    ranges:
    - min: 1100
      max: 1300
      # delete this: users: []
  volumes:
  ...
oc login -u admin -p redhat
oc create -f /home/student/DO425/labs/storage-isolation/restricted-scc.yaml
oc create serviceaccount secure-nfs -n secure-nfs
oc adm policy add-scc-to-user secure-nfs -z secure-nfs -n secure-nfs  # create SA in secure-nfs project and
    link with new SCC
oc describe scc secure-nfs                                           # shows supplemental group range
    1100-1300
oc login -u developer -p redhat
vim ~/database-secure-nfs.yaml
  . . .
  kind: DeploymentConfig
  spec:
    . . .
    template:
      metadata:
      spec:
        . . .
        name: postgresql
        serviceAccountName: secure-nfs
        securityContext:
          supplementalGroups:
            - 1200                                                   # secure-nfs group
oc create -f ~/database-secure-nfs.yaml
oc get pods                                                          # now it works
```

FINAL LAB 8

```
TBA
```

9. Configuring Web Application Single Sign-on

1. Security Assertion Markup Language (SAML) 2.0
2. OpenID Connect
3. JWT

Describing the OpenID Connect Authorization Code Flow

1. The application redirects to the SSO server, which presents a login screen and validates the user's credentials.
2. On successful authentication, the SSO redirects back to the application providing a 'code'.
3. The application uses the code to request an access token from SSO server.
4. The SSO server returns an access token that the application uses to authorize end user's requests and to submit requests to other applications that are clients of the same SSO realm.

CONFIGURING KEYCLOAK ADAPTERS FOR SINGLE SIGN-ON

The core technology of Red Hat's SSO solution is the Keycloak open source project.

DESCRIBING SSO CLIENT ACCESS TYPES

1. 'client protocol' defines whether the application uses SAML 2.0 or OpenID Connect
2. 'access type' defines whether the application is required to authenticate itself or not
3. 'valid Redirect URIs' protects the SSO server from sending tokens to applications other than the ones that initiated an authentication request

LAB 9.1

```
# Access the SSO web console and create a realm for the Ntier application.
# Create the js and java clients in the java-js-realm realm and configure them for the Ntier application
    front end and back ends.
# Create a user for the Ntier application.
# Create and configure a project for the Ntier application:
oc login -u developer -p redhat https://master.lab.example.com
cat ~/DO425/labs/webapp/create-cm.sh                                    # Review the script that creates the
    Ntier application configuration map
~/DO425/labs/webapp/create-cm.sh
oc describe cm ntier-config                                             #
    https://sso-websso.apps.lab.example.com/auth
cat ~/DO425/labs/webapp/deploy-pgsql.sh                                 # Review the script that deploys the
    Ntier application database.
~/DO425/labs/webapp/deploy-pgsql.sh
oc get pod
cat ~/DO425/labs/webapp/deploy-eap.sh                                   # Review the script that deploys the
    Java EE back end of the Ntier application.
~/DO425/labs/webapp/deploy-eap.sh
oc get pod                                                              # now we have postgresql-1-d9nfr and
    eap-app-2-ftm2q
oc logs eap-app-2-ftm2q
curl -ik https://eap-app-webapp.apps.lab.example.com/jboss-api
~/DO425/labs/webapp/deploy-springboot.sh
oc get pod                                                              # + springboot-app-2-7ktcv
curl -ik https://springboot-app-webapp.apps.lab.example.com/springboot-api/status # Access the Spring Boot
    back end REST API.
~/DO425/labs/webapp/deploy-nodejs.sh                                    # Deploy the JavaScript front end of
    the Ntier application
oc get pod                                                              # + nodejs-app-2-vcfr7
firefox https://nodejs-app-webapp.apps.lab.example.com
```

FINAL COMPREHENSIVE LAB1 - SINGLE CONTAINER APP

```
TBA
```

FINAL COMPREHENSIVE LAB2 - MULTI-CONTAINER APPS

```
TBA
```

APPENDIX

To help create objects:

```
oc explain dc.spec.template.spec
oc new-app --docker-image=registry.lab.example.com/image -o yaml > resources.yml
cat resources.yml
```

Table of important files:

Path	Purpose
/etc/origin/master/master-config.yml	master config
/etc/origin/node/node-config.yml	node config
/etc/containers/registries.d/[REGISTRY].yml	where to store new signatures and retrieve existing ones
/etc/containers/policy.json	what registries are allowed
/etc/docker/certs.d/[URL]/*.crt	private CAs on each node
/etc/ipa/ca.crt	root CA of the IdM server
/etc/pki/ca-trust/source/anchors	automatically trusted CAs
/var/lib/atomic/sigstore	locally stored image signatures

Note: To generate beautiful PDF file, install `latex` *and* `pandoc`*:*

```
sudo yum install pandoc pandoc-citeproc texlive
```

And then use `pandoc` *v1.12.3.1 to output Github Markdown to the PDF:*

```
pandoc -f markdown_github -t latex -V geometry:margin=0.3in -o DO425.pdf DO425.md
```

For better result (pandoc text-wrap code blocks), you may want to try my listings-setup.tex:

```
pandoc -f markdown_github --listings -H listings-setup.tex -V geometry:margin=0.3in -o DO425.pdf DO425.md
```