Advanced Labs in Software Engineering Summary

Chapter 1:

- Software applications typically depend on other libraries, configuration files, or services that are provided by the runtime environment. The traditional runtime environment for a software application is a physical host or virtual machine, and application dependencies are installed as part of the host.
- The **major drawback** to traditionally deployed software application is that the application's dependencies are entangled with the runtime environment. An application may break when any updates or patches are applied to the base operating system (OS).
- Furthermore, a traditionally deployed application must be stopped before updating the associated dependencies.
- A container is a set of one or more **processes** that are isolated from the rest of the system. Containers provide many of the same benefits as virtual machines, such as **security, storage, and network isolation**. Containers require far **fewer hardware resources** and are quick to start and terminate. The use of containers not only helps with the efficiency, elasticity, and reusability of the hosted applications, but also with application **portability**.
- The following are other major advantages to using containers:
    - Low hardware footprint
    - Environment isolation
    - Quick deployment
    - Multiple environment deployment
    - Reusability
- From the Linux kernel perspective, a container is a process with restrictions. However, instead of running a single binary file, a container runs an **image**. An image is a file-system bundle that contains all dependencies required to execute a process.
- An **image repository** is just a service - public or private - where images can be stored, searched, and retrieved.
- **Podman** can handle multiple containers while **docker** handle just one container.
- **Kubernetes** is an orchestration service that simplifies the deployment, management, and scaling of containerized applications. The smallest unit manageable in Kubernetes is a **pod**. A pod consists of one or more containers with its storage resources and IP address that represent a single application.
- Kubernetes offers the following features on top of a container infrastructure:
    - Service discovery and load balancing
    - Horizontal scaling
    - Self-healing
    - Automated rollout

    - Secrets and configuration management
    - Operators
- OpenShift adds the following features to a Kubernetes cluster:
    - Integrated developer workflow
    - Routes
    - Metrics and logging
    - Unified UI

Chapter 2:

- Container images can be found in image registries: services that allow users to search and retrieve container images. Podman users can use the **search** subcommand to find available images from remote or local registries. [*sudo podman search x*]
- Container images are named based on the following syntax:
- *registry_name/user_name/image_name:tag*
    - First **registry_name**, the name of the registry storing the image. It is usually the FQDN of the registry.
    - **user_name** stands for the user or organization the image belongs to.
    - The **image_name** should be unique in user namespace.
    - The **tag** identifies the image version. If the image name includes no image tag, latest is assumed.
- Podman stores images locally and you can list them with the **images** subcommand. [*sudo podman images*]
- The podman run command uses all parameters after the image name as the **entry point** command for the container.
- To start a container image as a background process, pass the **-d** option to the podman run command.
- Most Podman subcommands accept the **-l** flag (l for latest) as a replacement for the container id. Use the **--name** option to set the container name when running the container with Podman.
- Many Podman flags also have an alternative long form:
    - **-t** is equivalent to **--tty**, meaning a **pseudo-tty** (pseudo-terminal) is to be allocated for the container.
    - **-i** is the same as **--interactive**. When used, standard input is kept open into the container.
    - **-d,** or its long form **--detach**, means the container runs in the background (detached). Podman then prints the container id.
- Podman can inject environment variables into containers at startup by adding the **-e** flag to the run subcommand.
- Use the **-p 8080:80** option with sudo podman run command to forward the port.

Chapter 3:

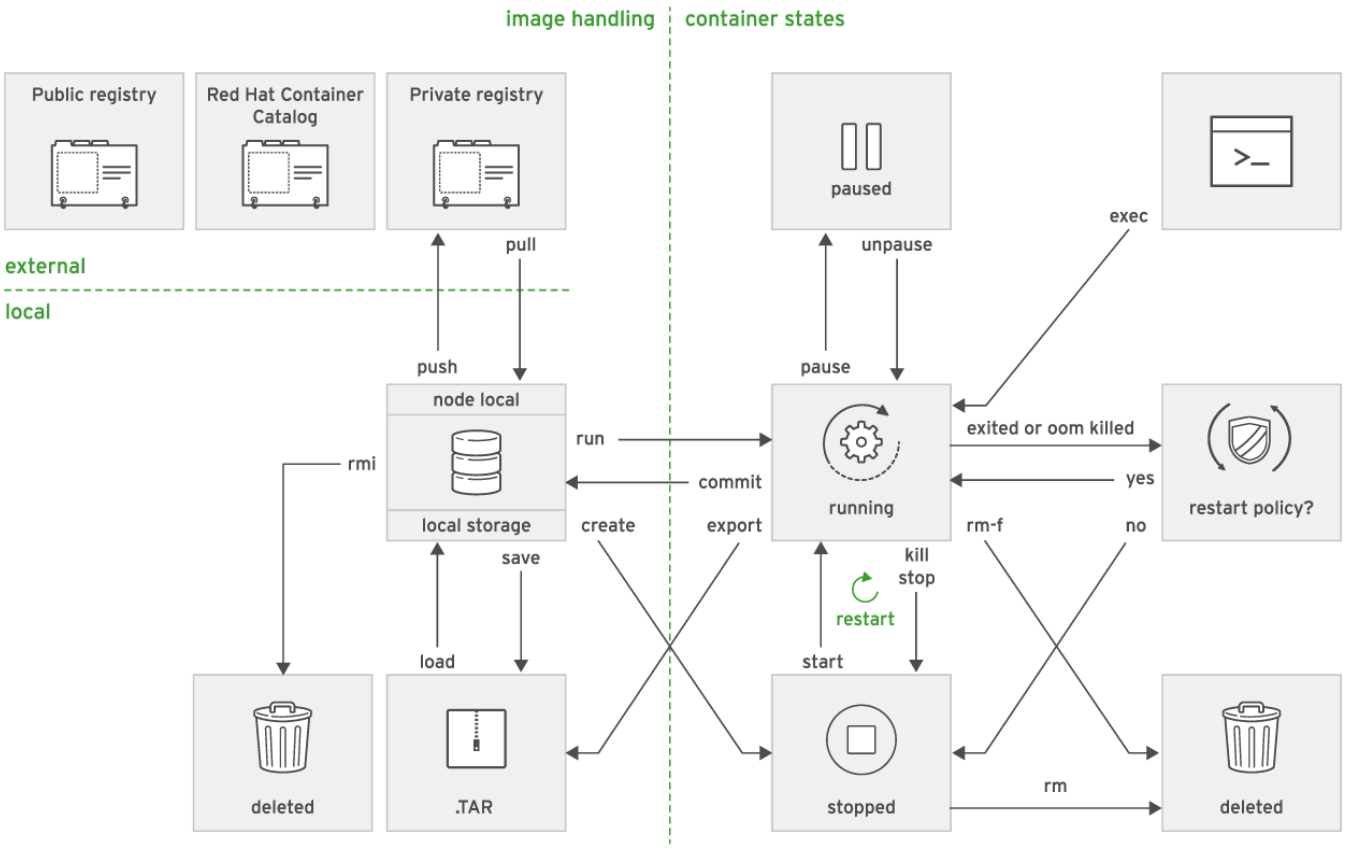
image handling
container states
Public registry
Red Hat Container Catalog
Private registry
paused
exec
pull
unpause
external
local
push
pause
node local
run
exited or oom killed
rmi
commit
yes
running
restart policy?
local storage
create
export
rm-f
no
save
kill
stop
restart
load
start
deleted
.TAR
stopped
rm
deleted

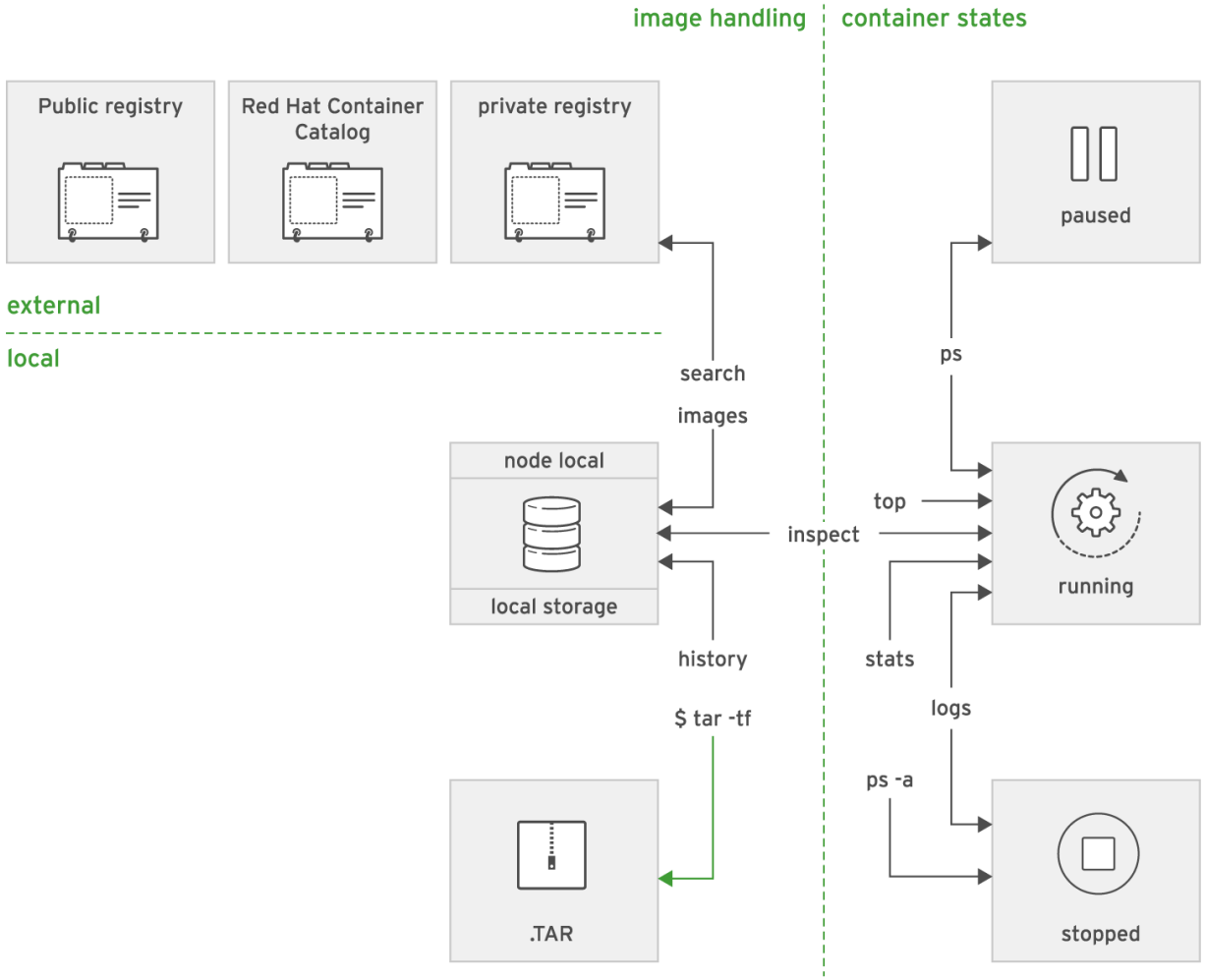
image handling
container states
Public registry
Red Hat Container Catalog
private registry
paused
external
local
ps
search
images
node local
top
inspect
running
local storage
history
stats
$ tar -tf
logs
ps -a
.TAR
stopped

- The **podman run** command creates a new container from an image and starts a process inside the new container. The **podman run** command, also, automatically generates a unique, random ID. It also generates a random container name. To define the container name explicitly, use the **--name** option when running a container.
- The **podman ps** command displays the container ID and names for all actively running containers.
- The **-d** option is responsible for running in detached mode. When using this option, Podman returns the container ID on the screen, allowing you to continue to run commands in the same terminal while the container runs in the background.
- The container image specifies the command to run to start the containerized process, known as the entry point.
- Some containers need to run as an interactive shell or process. This includes containers running processes that need user input (such as entering commands), and processes that generate output through standard output.
- The **-t** and **-i** options enable terminal redirection for interactive text-based programs. The **-t** option allocates a **pseudo-tty** (a terminal) and attaches it to the standard input of the container. The **-i** option keeps the container's standard input open, even if it was detached, so the main process can continue waiting for input.
- The **podman exec** command starts an additional process inside an already running container.
- Podman remembers the last container used in any command. Developers can skip writing this container's ID or name in later Podman commands by replacing the container id by the **-l** option.
- This command lists running containers **podman ps**.
- Podman does not discard stopped containers immediately. Podman preserves their local file systems and other states for facilitating postmortem analysis. Option **-a** lists all containers, including stopped ones.
- This command lists metadata about a running or stopped container **podman inspect.**
- This command stops a running container gracefully **podman stop**.
- **podman kill**: This command sends Unix signals to the main process in the container. If no signal is specified, it sends the *SIGKILL* signal, terminating the main process and the container.
- This command restarts a stopped container **podman restart**.
- The **podman restart** command creates a new container with the same container ID, reusing the stopped container state and file system.
- This command deletes a container and discards its state and file system **podman rm**.
- The **-f** option of the **rm** subcommand instructs Podman to remove the container even if not stopped. This option terminates the container forcefully and then removes it. Using **-f** option is equivalent to **podman kill** and **podman rm** commands together.

- You can delete all containers at the same time. Many podman subcommands accept the **-a** option. This option indicates using the subcommand on all available containers or images.
- Container storage is said to be ephemeral, meaning its contents are not preserved after the container is removed.
- Container images were characterized as immutable and layered, meaning that they are never changed, but rather composed of layers that add or override the contents of layers below.
- If the administrator needs to reclaim old container storage, the container can then be deleted using **podman rm** *container_id*. This command also deletes the container storage. The stopped container IDs can be found using **podman ps -a** command.
- To bind mount a host directory to a container, add the **-v** option to the **podman run** command, specifying the host directory path and the container storage path, separated by a colon (:).
- When Podman creates containers on the same host, it assigns each container a unique IP address and connects them all to the same software-defined network. These containers can communicate freely with each other by IP address.
- Each SDN is isolated, which prevents a container in one network from communicating with a container in a different network.
- A container is assigned an IP address from a pool of available addresses. When a container is destroyed, the container's address is released back to the pool of available addresses. Another problem is that the container software-defined network is only accessible from the container host.
- Define port forwarding rules to allow external access to a container service. Use the *-p [<IP address>:][<host port>:]<container port>* option with the **podman run** command to create an externally accessible container.
- To see the port assigned by Podman, use the **podman port <container name>** command.

Chapter 4:

- Image registries are services offering container images to download. They allow image creators and maintainers to store and distribute container images to public or private audiences.
- To configure registries for the podman command, you need to update the */etc/containers/registries.conf* file. Edit the *registries* entry in the *[registries.search]* section, adding an entry to the values list.
- The **podman search** command finds images by image name, username, or description from all the registries listed in the */etc/containers/registries.conf* configuration file.
  [student@workstation ~]$ **sudo podman search [OPTIONS] <term>**
- The following table shows some useful options available for the search subcommand:

Option	Description
--limit <number>	Limits the number of listed images per registry.
--filter <filter=value>	Filter output based on conditions provided. Supported filters are: • **stars=<number>**: Show only images with at least this number of stars. • **is-automated=<true|false>**: Show only images automatically built. • **is-official=<true|false>**: Show only images flagged as official.
--tls-verify <true|false>	Enables or disables HTTPS certificate validation for all used registries. true

- A remote registry exposes web services that provide an application programming interface (API) to the registry.
- The **podman login** command allows username and password authentication to a registry.
- To pull container images from a registry, use the podman pull command.
- The podman pull command uses the image name obtained from the search subcommand to pull an image from a registry. The pull subcommand allows adding the registry name to the image.

[student@workstation ~]$ sudo podman pull [OPTIONS] [REGISTRY[:PORT]/]NAME[:TAG]

- Podman provides an **images** subcommand to list all the container images stored locally.
- An image tag is a mechanism to support multiple releases of the same image.
- For example, a developer finishes testing a custom container in a machine and needs to transfer this container image to another host for another developer, or to a production server. There are two ways to do this:

    - Save the container image to a .tar file.
    - Publish (push) the container image to an image registry.
- Existing images from the Podman local storage can be saved to a .tar file using the **podman save** command.
- The general syntax of the save subcommand is as follows:

[student@workstation ~]$ sudo podman save [-o FILE_NAME] IMAGE_NAME[:TAG]

- To restore the container image, use the podman load command:

[student@workstation ~]$ sudo podman load [-i FILE_NAME]

- To delete an image from the local storage, run the podman rmi command:

[student@workstation ~]$ sudo podman rmi [OPTIONS] IMAGE [IMAGE...]

- To delete all images that are not used by any container, use the following command:

[student@workstation ~]$ sudo podman rmi -a

- The syntax for the podman commit command is as follows:

[student@workstation ~]$ sudo podman commit [OPTIONS] CONTAINER \

> [REPOSITORY[:PORT]/]IMAGE_NAME[:TAG]

Option	Description
--author ""	Identifies who created the container image.
--message ""	Includes a commit message to the registry.
--format	Selects the format of the image. Valid options are oci and docker.

- To find the ID of a running container in Podman, run the **podman ps** command.
- To identify which files were changed, created, or deleted since the container was started, use the diff subcommand.
- To tag an image, use the podman tag command:

[student@workstation ~]$ sudo podman tag [OPTIONS] IMAGE[:TAG] \

> [REGISTRYHOST/][USERNAME/]NAME[:TAG]

- A single image can have multiple tags assigned using the **podman tag** command. To remove them, use the **podman rmi** command.
- To push the image to the registry the syntax of the push subcommand is:

[student@workstation ~]$ sudo podman push [OPTIONS] IMAGE [DESTINATION]

Chapter 5:

- Dockerfiles are another option for creating container images, addressing these limitations. Dockerfiles are easy to share, version control, reuse, and extend.
- Dockerfiles also make it easy to extend one image, called a **child image**, from another image, called a **parent image**. A child image incorporates everything in the parent image and all changes and additions made to create it.
- It is not trivial to customize an application configuration to follow recommended practices for containers, and so starting from a proven parent image usually saves a lot of work.
- Two sources of container images to use either as parent images or for changing their Dockerfiles are **Docker Hub** and the **Red Hat Software Collections Library (RHSCL)**.
- All components have been rebuilt by Red Hat to avoid known **security vulnerabilities**.
- **Mission-critical** applications require trusted containers.
- **Source-to-Image (S2I)** provides an alternative to using Dockerfiles to create new container images and can be used either as a feature from OpenShift or as the standalone s2i utility. S2I allows developers to work using their usual tools, instead of learning Dockerfile syntax and using operating system commands such as yum, and usually creates slimmer images, with fewer layers.
- A Dockerfile is a mechanism to automate the building of container images. Building an image from a Dockerfile is a three-step process:
    - Create a working directory
    - Write the Dockerfile
    - Build the image with Podman
- Lines that begin with a hash, or pound, symbol (#) are comments.
- *INSTRUCTION* states for any Dockerfile instruction keyword. Instructions are not case-sensitive, but the convention is to make instructions all uppercase to improve visibility.
- The first non-comment instruction must be a FROM instruction to specify the base image.
- Each Dockerfile instruction runs in an independent container using an intermediate image built from every previous command. This means each instruction is independent from other instructions in the Dockerfile.
- The **FROM** instruction declares that the new container image extends another container base image.
- The **LABEL** is responsible for adding generic metadata to an image.
- **MAINTAINER** indicates the Author field of the generated container image's metadata.
- **RUN** executes commands in a new layer on top of the current image.
- **EXPOSE** indicates that the container listens on the specified network port at runtime.

- **ENV** is responsible for defining environment variables that are available in the container.
- **ADD** instruction copies files or folders from a local or remote source and adds them to the container's file system.
- **COPY** copies files from the working directory and adds them to the container's file system.
- **USER** specifies the username or the UID to use when running the container image for the **RUN**, **CMD**, and **ENTRYPOINT** instructions.
- **ENTRYPOINT** specifies the default command to execute when the image runs in a container.
- **CMD** provides the default arguments for the **ENTRYPOINT** instruction.
- Exec form is the preferred form.
- The Dockerfile should contain at most one **ENTRYPOINT** and one **CMD** instruction. If more than one of each is present, then only the last instruction takes effect. **CMD** can be present without specifying an **ENTRYPOINT**. In this case, the base image's **ENTRYPOINT** applies, or the default **ENTRYPOINT** if none is defined.
- If the source is a file system path, it must be inside the working directory.
- If the source is a compressed file, then the ADD instruction decompresses the file to the destination folder. The COPY instruction does not have this functionality.
- Red Hat recommends minimizing the number of layers. You can achieve the same objective while creating a single layer by using the && conjunction. The problem with this approach is that the readability of the Dockerfile decays. Use the \ escape code to insert line breaks and improve readability.

Chapter 6:

- One of the main advantages of using Kubernetes is that it uses several nodes to ensure the resiliency and scalability of its managed applications. Kubernetes forms a cluster of **node servers** that run containers and are centrally managed by a set of **master servers**. A server can act as both a **server** and a **node**, but those roles are usually segregated for increased stability.
- An OpenShift cluster is a Kubernetes cluster that can be managed the same way, but using the management tools provided by OpenShift, such as the command-line interface or the web console. This allows for more productive workflows and makes common tasks much easier.
- Kubernetes Resource Types:
    - Pods
        - Basic unit of work for Kubernetes.
    - Services
        - Define a single IP/port combination that provides access to a pool of pods.
    - Replication Controllers
        - A Kubernetes resource that defines how pods are replicated into different nodes.
    - Persistent Volumes
        - Define storage areas to be used by Kubernetes pods.
    - Persistent Volume Claims
        - Represent a request for storage by a pod.
    - ConfigMaps and Secrets
        - Contains a set of keys and values that can be used by other resources.
- OpenShift Resource Types:
    - Deployment config (dc)
        - Represents the set of containers included in a pod, and the deployment strategies to be used.
    - Build config (bc)
        - Defines a process to be executed in the OpenShift project. Used by the OpenShift Source-to-Image (S2I) feature to build a container image from application source code stored in a Git repository.
    - Routes
        - Represent a DNS host name recognized by the OpenShift router as an ingress point for applications and microservices.
- Services link more stable IP addresses from the SDN to the pods. If pods are restarted, replicated, or rescheduled to different nodes, services are updated, providing scalability and fault tolerance.
-

- Kubernetes provides a virtual network that allows pods from different workers to connect. But, Kubernetes provides no easy way for a pod to discover the IP addresses of other pods.
- Services are essential resources to any OpenShift application. They allow containers in one pod to open network connections to containers in another pod. A pod can be restarted for many reasons, and it gets a different internal IP address each time. Instead of a pod having to discover the IP address of another pod after each restart, a service provides a stable IP address for other pods to use, no matter what worker node runs the pod after each restart.
- The set of pods running behind a service is managed by a *DeploymentConfig* resource. A *DeploymentConfig* resource embeds a *ReplicationController* that manages how many pod copies (replicas) have to be created and creates new ones if any of them fail.
- An application typically finds a service IP address and port by using environment variables. For each service inside an OpenShift project, the following environment variables are automatically defined and injected into containers for all pods inside the same project:
    - SVC_NAME_SERVICE_HOST is the service **IP address**.
    - SVC_NAME_SERVICE_PORT is the service **TCP port**.
- The **oc new-app** command can be used with the -o json or -o yaml option to create a skeleton resource definition file in JSON or YAML format, respectively. This file can be customized and used to create an application using the **oc create -f <filename>** command or merged with other resource definition files to create a composite application.
- Use the **oc get** command to retrieve information about resources in the cluster. Generally, this command outputs only the most important characteristics of the resources and omits more detailed information.
- Use the **oc get** all command to retrieve a summary of the most important components of a cluster. This command iterates through the major resource types for the current project and prints out a summary of their information.
- If the summaries provided by **oc get** are insufficient, use the oc describe command to retrieve additional information.
- The **oc delete** command removes a resource from an OpenShift cluster. Note that a fundamental understanding of the OpenShift architecture is needed here, because deleting managed resources such as pods results in new instances of those resources being automatically created. When a project is deleted, it deletes all of the resources and applications contained within it.
- The **oc exec** command executes commands inside a container. You can use this command to run interactive and noninteractive batch commands as part of a script.

- The **oc new-app** command does not create a route resource when building a pod from container images, Dockerfiles, or application source code. After all, **oc new-app** does not know if the pod is intended to be accessible from outside the OpenShift instance or not.
- Another way to create a route is to use the **oc expose service** command, passing a service resource name as the input. The --name option can be used to control the name of the route resource.
- By default, routes created by **oc expose** generate DNS names of the form:

  ```
  route-name-project-name.default-domain
  ```

  Where:
  - **route-name** is the name assigned to the route. If no explicit name is set, OpenShift assigns the route the same name as the originating resource (for example, the service name).
  - **project-name** is the name of the project containing the resource.
  - **default-domain** is configured on the OpenShift master and corresponds to the wildcard DNS domain listed as a prerequisite for installing OpenShift.
- **Source-to-Image (S2I)** is a tool that makes it easy to build container images from application source code. This tool takes an application's source code from a Git repository, injects the source code into a base container based on the language and framework desired, and produces a new container image that runs the assembled application.
- S2I is the primary strategy used for building applications in OpenShift Container Platform. The main reasons for using source builds are:
  - User efficiency.
  - Patching.
  - Speed.
  - Ecosystem.
- Even though it is evident that the container image needs to be updated when application code changes, it may not be evident that the deployed pods also need to be updated should the builder image change.
- The BuildConfig pod is responsible for creating the images in OpenShift and pushing them to the internal container registry. Any source code or content update typically requires a new build to guarantee the image is updated.
- The DeploymentConfig pod is responsible for deploying pods to OpenShift. The outcome of a DeploymentConfig pod execution is the creation of pods with the images deployed in the internal container registry. Any existing running pod may be destroyed, depending on how the DeploymentConfig resource is set.
- The BuildConfig resource creates or updates a container image. The DeploymentConfig reacts to this new image or updated image event and creates pods from the container image.

Chapter 7:

- Podman uses Container Network Interface (CNI) to create a software-defined network (SDN) between all containers in the host. Unless stated otherwise, CNI assigns a new IP address to a container when it starts.
- Due to the dynamic nature of container IP addresses, applications cannot rely on either fixed IP addresses or fixed DNS host names to communicate with middleware services and other application services.
- Using environment variables allows you to share information between containers with Podman.
- Pods are attached to a Kubernetes namespace, which OpenShift calls a project. When a pod starts, Kubernetes automatically adds a set of environment variables for each service defined on the same namespace. These environment variables usually follow a convention:
    - Uppercase: All environment variables are set using uppercase names.
    - Snakecase: Any environment variable created by a service is usually composed of multiple words separated with an underscore (_).
    - Service name first: The first word for an environment variable created by a service is the service name.
    - Protocol type: Most network environment variables include the protocol type (TCP or UDP).
- OpenShift templates provide a way to simplify the creation of resources that an application requires. A template defines a set of related resources to be created together, as well as a set of application parameters. The attributes of template resources are typically defined in terms of the template parameters, such as a resource's name attribute.
- The OpenShift installer creates several templates by default in the openshift namespace. Run the **oc get templates** command with the -n openshift option to list these preinstalled templates
- Templates define a set of parameters, which are assigned values. OpenShift resources defined in the template can get their configuration values by referencing named parameters. Parameters in a template can have default values, but they are optional. Any default value can be replaced when processing the template.
- When you process a template, you generate a list of resources to create a new application. To process a template, use the **oc process** command.
- OpenShift Container Platform manages persistent storage as a set of pooled, cluster-wide resources. To add a storage resource to the cluster, an OpenShift administrator creates a PersistentVolume object that defines the necessary metadata for the storage resource.
- When an application requires storage, you create a PersistentVolumeClaim (PVC) object to request a dedicated storage resource from the cluster pool.

Chapter 8:

- The S2I image creation process is composed of two major steps:
    - **Build** step: Responsible for compiling source code, downloading library dependencies, and packaging the application as a container image. Furthermore, the build step pushes the image to the OpenShift registry for the deployment step. The **BuildConfig** (BC) OpenShift resources drive the build step.
    - **Deployment** step: Responsible for starting a pod and making the application available for OpenShift. This step executes after the build step, but only if the build step succeeded. The **DeploymentConfig** (DC) OpenShift resources drive the deployment step.
- For the S2I process, each application uses its own **BuildConfig** and **DeploymentConfig** objects, the name of which matches the application name.
- For example, to retrieve the logs from a build configuration, run the following command.

```
$ oc logs bc/<application-name>
```

- If a build fails, after finding and fixing the issues, run the following command to request a new build:

```
$ oc start-build <application-name>
```

- Deployment logs can be checked with the oc command:

```
$ oc logs dc/<application-name>
```

- Sometimes, the source code requires some customization that may not be available in containerized environments, such as database credentials, file system access, or message queue information. Those values usually take the form of internal environment variables.
    - Troubleshooting Permission Issues:
        - The developer runs into permission issues, such as access denied due to the wrong permissions, or incorrect environment permissions set by administrators.
        - This **oc adm policy** command enables OpenShift executing container processes with non-root users.
    - Troubleshooting Invalid Parameters:
        - A good practice to centralize shared parameters is to store them in **ConfigMaps**. Those **ConfigMaps** can be injected through the Deployment Config into containers as environment variables.
    - Troubleshooting Volume Mount Errors:
        - To resolve the issue, delete the persistent volume claim and then the persistent volume. Then recreate the persistent volume.
    - Troubleshooting Obsolete Images

        - If you push a new image to the registry with the same name and tag, you must remove the image from each node the pod is scheduled on with the command podman rmi.
- Podman provides port forwarding features by using the -p option along with the **run** subcommand. In this case, there is no distinction between network access for regular application access and for troubleshooting.
- Podman and OpenShift provide the ability to view logs in running containers and pods to facilitate troubleshooting. But neither of them is aware of application specific logs.
- Some developers consider Podman and OpenShift logs to be too low-level, making troubleshooting difficult. Fortunately, OpenShift provides a high-level logging and auditing facility called events.
- Podman and OpenShift provide the **exec** subcommand, allowing the creation of new processes inside a running container, with the standard output and input of these processes redirected to the user terminal.