**Virtualization VS Containerization**

* **Virtualization** (hardware-level virtualization):
  + It is a technology to run multiple same or different OSs on a single physical system which are completely isolated from each other to share hardware resources.
  + Run
* **Containerization** (OS-level virtualization):

**Image VS Container**

* **image** is a file-system bundle that contains all dependencies required to execute a process file in the file system, installed packages, available resources, running processes, and kernel modules.
* **container** is an instance of an image, with known lifecycle start, run, pause, stop, delete.

**Podman:**

* Podman is a tool for creating, deploying, and running applications easily.
* It allows us to package our applications with all the dependencies and distribute them as individual bundles.

**Root VS Rootless containers**

* **Root**: containers that run with root privileges on the host machine.
  + This meaning that the containers have access to all system resources and can perform any operation
* **Rootless**: containers that run without root privileges on the host machine.
  + This means that the containers are limited in their access to system resources, and that potentially dangerous operations can be prevented

**Mounting the host directory**

* Podman can mount host directories inside a running container.
* The host directory must be configured with ownership and permissions allowing access to the container.

**Steps to mount on the host:**

1. **Create a directory:**
2. **Apply the container\_file\_t context to the directory (and all subdirectories)**
3. **Provide a session to a user id to execute commands**
4. **Apply the SELinux container policy that you set up in the first step to the newly created directory**
5. **Mounting a volume**

**Public VS Private Registries:**

* **Public registries:** images that are publicly available to be downloaded.
* **Private registries:** Private registries give image creators the control about their images placement, distribution and usage.
* **Image Tag**: is a mechanism to support multiple releases (version) of the same image.

**How to save and load images?**

2 ways =>

1. Save container image as a **.tar** file
2. Push the container image to an image registry (quay.io).

**Docker**

* **Dockerfile:** contains a list of instructions that Docker will execute when you issue the docker build command, it used to create custom docker image

**How to create custom image [Midterm]:**

1. Create a working directory
2. Write the Dockerfile
3. Build the image with Podman

Text, application, chat or text message

Description automatically generated

**Dockerfile content:**

|  |  |
| --- | --- |
| **#** | Comment |
| **FROM** | Declares that the new container image extends ubi7/ubi:7.7 container base image. |
| **LABEL** | Is responsible for adding generic meta data, it’s a simple key-value pair. |
| **MAINTAINER** | Indicates the Author field |
| **RUN** | used to execute commands is /bin/sh. |
| **EXPOSE** | Indicates that the container listens on the specified network port at run time |
| **ENV** | Define environments variables |
| **ADD** | Copies file or folders from local or remote and adds them to container file system |
| **COPY** | copies files from the working directory and adds them to the container's file system. **(CAN’T COPY REMOTE OR URLs)** |
| **USER** | specifies the username or the UID |
| **ENTRYPOINT** | specifies the default command to execute when the image runs in a container |
| **CMD** | provides the default arguments for the ENTRYPOINT instruction |

**Source-to-image - S2I:**

* **S2I:** tool that makes it easy to build container images from application source code.
  + GitHub Repo 🡪 Inject Source Code into base container 🡪 produce new container

**S2I Image Creation:**

Two Major Steps:

1. **Build step:** Responsible for compiling source code, downloading library dependencies, and packaging the application as a container image.
2. **Deployment step:** Responsible for starting a pod and making the application available for OpenShift

**Kubernetes:**

is a portable, extensible, open-source platform for managing containerized workloads and services.

**RedHat OpenShift Container Platform (RHOCP):**

set of modular components and services built on top of Red Hat CoreOS and Kubernetes.

* RHOCP adds PaaS capabilities such as remote management, increased security, monitoring and auditing, application lifecycle management.

**Troubleshooting Volume Mount Errors**

* When redeploying an application that uses a persistent volume on a local file system, a pod might not be able to allocate a persistent volume claim even though the persistent volume indicates that the claim is released
* **Solution:** 
  + Delete the persistent volume 🡺 **oc delete pv <pv\_name>**
  + Recreate the persistent volume 🡺 **oc create -f <pv\_resource\_file>**

**Troubleshooting Permission Issues**

* **Solution:** 
  + relax the OpenShift project security with the command 🡺 **oc adm policy**

**Troubleshooting Invalid Parameters**

* **Solution:** 
  + Centralize shared parameters is to store them in ConfigMaps,
  + Injecting the same ConfigMap into different containers ensures that not only the same environment variables are available, but also the same values.

**Troubleshooting Obsolete Images**

* **Solution:** 
  + 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**.
  + Run the 🡺 **oc adm prune** command for an automated way to remove obsolete images and other resources.

**Forwarding Ports for Troubleshooting**

* Podman provides port forward 🡺 **podman run –name db -p 30306:3306 mysql**

**Solution:**

* Using OpenShift 🡺 **oc port-forward db 30306 3306**

**Graphical user interface, text, application, email

Description automatically generated**