Extremely Basic Containerization

With Podman and OpenSUSE MicroOS

June 2022 Update

Johnnie Odom
The School District of Escambia County (ECSD)
The Technology Transfer Partners (TTP)

Prelude

- Please see my containers overview from 2021
  (TTP Login Required)
  https://www.thettp.org/?session=a-field-guide-to-containers-in-july-2021
- 1. Theory (See Above)
  2. ???
  3. Millions of Orchestrated Containers Whizzing Around Kubernetes
- Today we take the first step in "???"
  - Doing at a basic, low level so that you understand the fundamentals.
  - Eventually you will use an orchestration solution and many more complex products.
- We will use "podman" throughout as our container platform.

Overall Steps To Running Containers With Podman

- Find a Use Case
- List Run Requirements From Use Case
- Set up a Container-Friendly OS
- Pick a Base Image
- Gather Other Static Files
- Determine Build Steps and Put Into Containerfile / Dockerfile
- Build Image
- Attempt Run From Image
- Cycle Of Debugging And Tweaking
- Run Multiple Containers From Image

Find A Use Case

- See “The 12 Factor App” from previous presentation for applications with best fit.
- Needs to be a relatively portable application that does not need an entire server to itself and does not require permanent on-disk data writes.
- My Choice: A simple Machine-Learning Client I built as a proof-of-concept for several ideas.
  - Uses an internal API to another server.
  - Uses Access Manager for Authentication
  - Therefore, sufficiently small and sufficiently real-world.

List Run Requirements From Use Case

- Think of *Every* Dependency Your App Has
  - OS-Level Libraries
  - Platform Ecosystem
  - Network
  - Configuration
  - Other Assumptions
- My Example App
  - Python 3
  - Flask and Other Libraries
  - Configuration File Includes Network Names
  - Writes Session Data
  - Access Manager Endpoints and Secrets

Set Up Container-Friendly OS

- Remember that all containers are Linux underneath.
  - More options every day both for local dev, in-house serving, and Cloud.
- OpenSUSE MicroOS is my pick.
  - A true embodiment of SUSE Principles
    - Even the bad ones.
  - Very lightweight
  - Has a Containerization pattern (includes podman)
  - Based on the idea of a transactional server

Pick A Base Image

- Remember: You are not booting an OS
  - Fundamental OS functions like process control handled by Container Host.
- Base Image provides OS functions except for lowest and then the tricker platform functionality.
  - In my example, SUSE Linux + Python 3 will be base.
- Creating an image from a customized system is harder than it should be.
- Most Base Images come from a well-known public repository.
  - Docker and Red Hat dominate searches here.
  - SUSE Options should be sufficient:
    https://registry.opensuse.org
    https://registry.suse.com
- Now we begin using podman
  To grab a base image:
  podman pull *URL*
  podman pull registry.opensuse.org/opensuse/httpd:latest
- MicroOS stores pulled images at:
  /var/lib/containers/storage/btrfs-images

Gather Other Static Files

- Need static files that are not part of base image but are part of our application / workload.
  - In my case, the application files for my program.
    - I put everything into a folder, eliminated any unnecessary files, and ensured that the program could run solely from the files needed.
- Think carefully about how your program runs and which parts are OS and system-dependent.
  - For example, Python venv does NOT copy all the files needed by Python and has fixed paths dependent on local OS configuration.
  - Remember also that Containers are not good at writing permanent files.

Determine Build Steps And Put Into Containerfile / Dockerfile

- A Containerfile is a line-by-line file containing the recipe / steps required to make the final set of files in the image from which your container will run.
  - By default still named “Dockerfile"
- Everything needs to be below the directory where you will run the build command.
- Layer on existing images using FROM <imageid>
- Copy files using COPY <relativepathbelowbuild> </absolute/path/in/container>
- RUN commands that will change the files in the container DURING build.
  - i.e. RUN pip -r subdir/requirements.txt
- ENV commands set environmental variables active during container run.
- CMD is a single shell command that will run when the container starts up.

Build Image

- Again, default filename for Containerfile is “Dockerfile”
- podman build <build directory>
  - “podman build ./“
- If all goes well, a new image is created with a new ID.
  - Can be seen with “podman image list”
- If mistakes were made then “podman image rm <imageid>”
  - And if error encountered then
    “podman image rm —force <imageid>”
- To see what is in an image then “podman image mount <imageid>”
  - Output will show path where the mount can be seen.

Attempt Run From Image

- First try running in foreground “podman run <imageid>”
  - Gives you feedback on errors, etc.
  - CTRL-C to stop.
- Networking:
  - Different depending on roomful and rootless
    - Out-of-scope for simple presentation and also dynamic.
  - “podman run -p <publicport>:<privateport>” will do a mapping between container and host.
- Per-Instance Configuration and Secrets should be done with environmental variables
  - “podman run -e “SOMEENVVAR=somevalue”

Cycle Of Debugging And Tweaking

- Try using containerized application, see it does not work.
- Stop Container
- Delete Image
- Make Changes
- Build (New) Image
- Start Container

Run Multiple Containers From Image

- Start each container in background with “podman run -dt <imageid>
- See all running containers with “podman ps” (shows container ids and image ids)
- See all running and registered-but-not-running (paused, etc.) containers with “podman ps —all”
- Stop a given container with “podman stop <containerid>” remembering to tweak config with environmental and run options.
- Final run command
  podman run -dt -p <targetport>:<sourceport> -e "LOCALVAR=somevalue" <imageid>