



Introduction to Containers for Data Science and Data Engineering



Open Data Science Conference West 2024

<https://github.com/mafudge/odsc24>

Michael Fudge

Professor of Practice

mafudge@syr.edu

Syracuse University, iSchool

<https://ischool.syr.edu>

Apphammer

<https://apphammer.co>



Part 1: Container Essentials

Important concepts for understanding containers and docker

Rationale

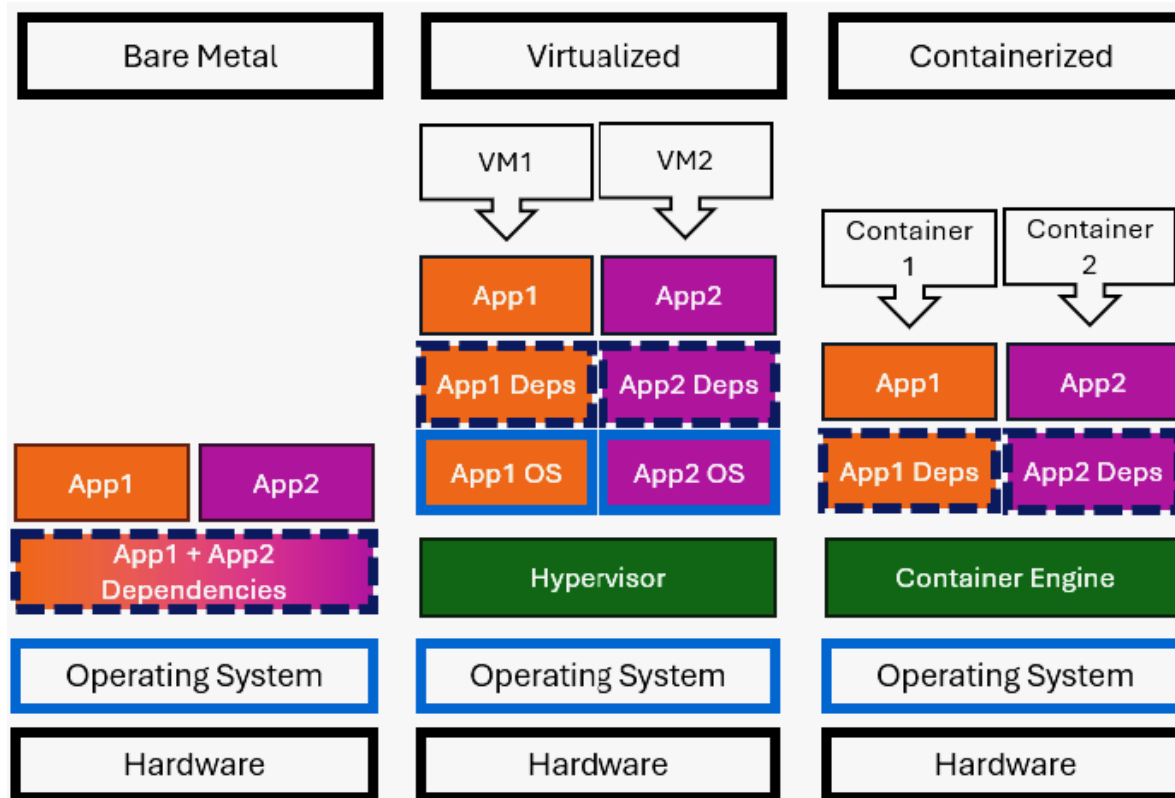
Have you ever:

- Worked on a project on more than one computer?
- Shared a project with a team?
- Said this: “but it works on my machine...”
- Spent hours trying to deploy your project live?
- If so, then containers are for YOU!

What is a container?

- A container is a form of virtualization.
- Containers virtualize the application and its dependencies.
- The virtualization occurs closer to the application than the hardware.

Containers, Virtualization, Bare Metal



Benefits of Containers

- **Reproducibility of work** - drastically minimizes “but it works on my machine”.
- Bundle the dependencies with your application.
- Can mimic complex environments with external systems.
- When combined with a git workflow, it can be team and cloud friendly.

Container Terminology

- **Image** application + dependencies at rest, consumes disk
- **Tag** the version / release name of the image
- **Container** running image: consumes disk + RAM/CPU/Network
- **Port** expose a TCP/UDP Port outside the container
- **Volume** persistent storage for a container

Our first container

- It always starts with a need...
- Hmm. I'd like to use: <https://jupyter.org/install>
- Rather than install all those dependencies on your computer, run it in a container!
- Portability! Reproducibility!

Dockerhub: finding images

- Dockerhub is an example of a container registry.
<https://hub.docker.com>
- A catalog of container images in the cloud!
- <https://hub.docker.com/search?q=jupyter%2Fminimal-notebook>
- Red Hat's Quay.io is another:
<https://quay.io/repository/jupyter/minimal-notebook>

Demo 1: Pulling an image



- Pulling images
 - From Dockerhub
 - From Another public Image repository
- Viewing downloaded images

Demo 1: Summary of Commands

What	Command	Example
Download image from Docker registry	<code>docker pull <image>:<tag></code>	<code>docker pull jupyter/minimal-notebook:lab-4.1.6</code>
Download image from Another public registry	<code>docker pull <reg>/<image>:<tag></code>	<code>docker pull quay.io/jupyter/minimal-notebook:lab-4.1.6</code>
View downloaded images + tags	<code>docker images</code>	<code>docker images</code>
Delete an image	<code>docker image rm <image>:<tag></code>	

Running an image as a container

- To run the image:
`docker run --name <containername> -d <image>:<tag>`
- `--name` give the container a name.
- `-d` run in the background, return to the console.

Demo 2: Containers!



- Run the container
- See running containers
- View logs of running container
- Stop a running container
- See all containers (running and stopped)
- Delete container definition

Demo 2: Summary of Commands

What	Command	Example
Run a container	<code>docker run --name <container> -d <name>:<tag></code>	<code>docker run --name jupyter -d jupyter/minimal-notebook:lab-4.1.6</code>
List running containers	<code>docker ps</code>	<code>docker ps</code>
View output of running container	<code>docker logs <name></code>	<code>docker logs jupyter</code>
Stop a running container	<code>docker stop <name></code>	<code>docker stop jupyter</code>
List all containers (running and stopped)	<code>docker ps -a</code>	<code>docker ps -a</code>
Delete a container	<code>docker rm <name></code>	<code>docker rm jupyter</code>

Publishing TCP/UDP Ports

- What good is running an application in a container if you cannot access it?
- When the container is created, we must publish the exposed the port:
 - p published-port:exposed-port or
 - p OUTSIDE:INSIDE
- When you publish a port, you can connect to exposed application services on the published port.
- For example, if 8888 is published, then <http://localhost:8888>

Demo 3: Ports



- Run the container, exposing 8888
- See the port is exposed
- Visit <http://localhost:8888>
- Find the key in the container output!
- Login to the application!

Demo 3: Summary of Commands

What	Command	Example
Run a container and expose ports	<code>docker run --name <container> -p <pub>:<exp> -d <name>:<tag></code>	<code>docker run --name jupyter -p 8888:8888 -d quay.io/jupyter/minimal- notebook:lab-4.1.6</code>

Need for volumes

- Containers are ephemeral. Only what is in the image is retained between subsequent runs.
- When you `docker rm` the changes are gone!
- How does one save changes? Volumes!
 - v `host:image` or
 - v `OUTSIDE:INSIDE`

Demo 4: Volumes



- In the running container save a file.
- Stop and remove the container
- Re-run the container
- Check back into the App... file is gone. Ephemeral.
- Stop and remove the container... again
- Check out that local work folder
- Mount that folder as a container volume
- Stop and remove the container... last time

Demo 4: Summary of Commands

What	Command	Example
Run a container and expose ports and a volume	<code>docker run --name <container> -p <pub>:<exp> -d <name>:<tag></code>	<code>docker run --name jupyter -p 8888:8888 -v \$PWD/work:/home/jovyan/work -d quay.io/jupyter/minimal-notebook:lab-4.1.6</code>
List running containers	<code>docker stop jupyter && docker rm jupyter</code>	<code>docker stop jupyter && docker rm jupyter</code>

Orchestration with Docker Compose

- Docker compose allows you to manage containers through a configuration file, in `yaml` format.
- Makes Starting and Stopping containers easier.
- Infrastructure as code:
 - Clear separate of Configuration –vs- Commands

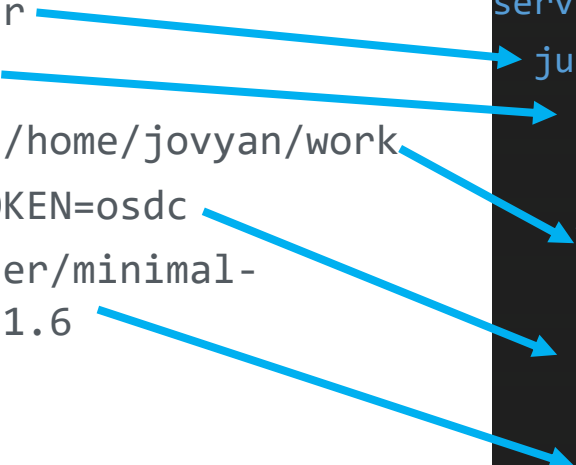
Docker vs Docker Compose

Command

```
$ docker run  
  --name jupyter  
  -p 8888:8888  
  -v $PWD/work:/home/jovyan/work  
  -e JUPYTER_TOKEN=osdc  
  quay.io/jupyter/minimal-  
  notebook:lab-4.1.6
```

Configuration file

```
version: "3"  
services:  
  jupyter:  
    ports:  
      - 8888:8888  
    volumes:  
      - ./work:/home/jovyan/work  
    environment:  
      - JUPYTER_TOKEN=osdc  
    image: quay.io/jupyter/minimal-  
    notebook:lab-4.1.6
```



The diagram illustrates the mapping between the Docker command and the Docker Compose configuration file. Blue arrows point from specific parts of the command to the corresponding sections in the configuration file:

- `--name jupyter` points to `jupyter:`
- `-p 8888:8888` points to `ports:`
- `-v $PWD/work:/home/jovyan/work` points to `volumes:`
- `-e JUPYTER_TOKEN=osdc` points to `environment:`
- `quay.io/jupyter/minimal-notebook:lab-4.1.6` points to `image:`

Demo 5: Docker Compose

- Start with docker compose
- View running containers
- Stop with docker compose



Demo 5: Summary of Commands

What	Command	Example
Build the configuration and start the container(s)	<code>docker compose -f <file> up -d</code>	<code>docker compose -f jupyter.yaml up -d</code>
List running containers from the configuration	<code>docker compose -f <file> ps</code>	<code>docker compose -d jupyter.yaml ps</code>
Stop the running containers and remove them	<code>docker compose -f <file> down</code>	<code>docker compose -d jupyter.yaml down</code>



Part 2: Development with Containers

Containerizing your development pipeline

Simple Python ETL Pipeline

- Simple data Pipeline: read file, transform it, write another file
- Open part2 folder in Visual Studio Code.
- Open a terminal in the part2 folder



Dockerfile: Building your own image

- `Dockerfile` explains how to build an image
- `FROM` is the base image from a public image repository
- `COPY` copies files into the image
- `RUN` executes a command in the image
- `ENTRYPOINT` defines the default command from `docker run` or `docker compose up`

Demo 6: Building A Custom Image



- Look at the Dockerfile. No data... just code!
- Look at the docker-compose.yaml File.
Data in volumes.
- Build the Image with docker compose
 - Note we don't need -f here.
- What is the image name?
- Run the application with docker compose

Demo 6: Summary of Commands

What	Command	Example
Build an image using the docker-compose file	<code>docker compose -f <file> build</code>	<code>docker compose build</code>
View images created by docker compose file	<code>docker compose -f <file> images</code>	<code>docker compose images</code>
View images with docker	<code>docker images</code>	<code>docker images</code>
Run the application	<code>docker compose -f <file> up</code>	<code>docker compose up</code>
Remove the container definitions	<code>docker-compose -f <file> down</code>	<code>docker compose down</code>

Useful, but not Realistic: Enter Dev Containers

- This is a useful way to run your working programs
- It is impracticable to implement a dev-loop this way:
write → run → debug...
- Enter the Visual Studio Code dev containers!
- `devcontainer.json` points to an image with code + debugger
- You are coding inside a running container!
- Current folder is mounted as a volume!

Demo 7: VS Code Dev Containers



- Open part2 in a dev container. It is building.
- Look at the `devcontainer.json`.
- Run code in container
- Set a breakpoint and debug!

Demo 7: Summary of Commands

What	Command
Open a Dev Container in VS Code	Open folder >< Remote window Reopen in container Dev Container builds
Run code in Dev Container	[F5]
Set a breakpoint	[F9]
Rebuild a dev container (if you add a new dependency, for example)	>< Remote Window Rebuild Container



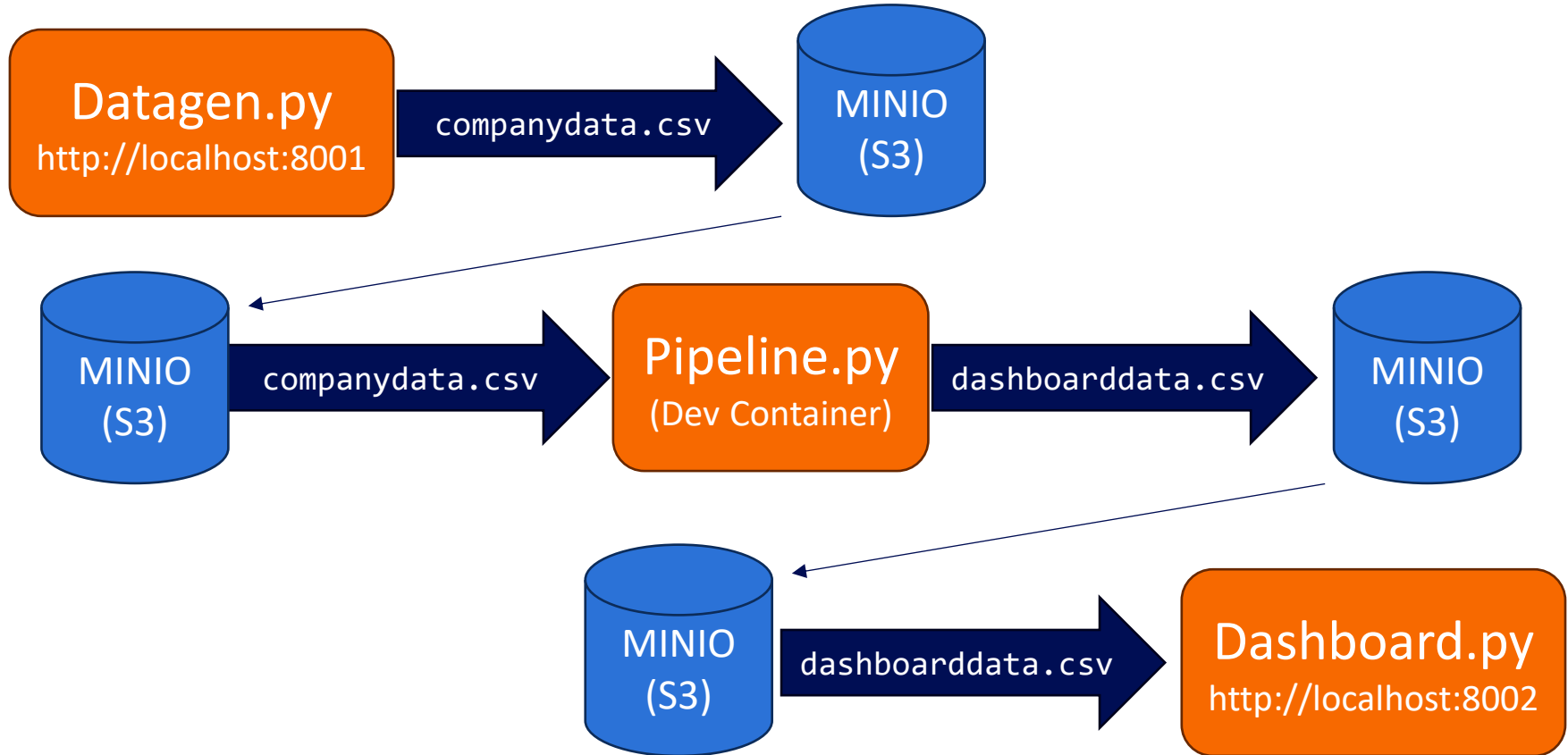
Part 3: Development with Containers and Other Container Dependencies

Use Docker Compose to orchestrate dependencies in your application

Overview of final pipeline

- Real world pipelines require many parts.
 - They read from/write to databases, for example.
 - Loosely coupled / No direct dependencies.
- Open `part3` folder in Visual Studio Code.
- This example uses `docker-compose.yml` with the dev container.
 - There are other containers used in this pipeline which demonstrate typical complexity.

Diagram of the pipeline



Demo 8: Complex Pipeline



- Open in Dev Container and build.
 - Look at all the running containers!
- Run `pipeline2.py`, Datagen <http://localhost:8001>, and Dashboard <http://localhost:8002>
- Watch the data update.
- Look at the `devcontainer.json`. Two docker-compose files.



Wrap-Up!

Summary

- Containers virtualize our applications,
 - bundling the dependencies with the app.
- Find images for your containers on repositories
 - docker hub and quay.io
- You can manage multiple containers with docker-compose and separate the configuration from the commands
- Write code inside containers with VS Code dev containers!
- Dev containers support docker compose for complex setups!

But Wait... There's more!

- How about a containerized Spark cluster With a Jupyter front-end and Minio storage?

<https://github.com/mafudge/docker-spark-cluster>

- Query your CSV/Excel files with SQL. Drop and go!

<https://github.com/mafudge/local-file-drill>

- Chat with your PDF file, 100% on device using llama2 (dev container example)

<https://github.com/mafudge/chat-pdf>

Cleaning up

- You might want to reclaim some disk space after the talk
- `docker system prune --all --volumes`

.....

Thank you For Attending!

Introduction to Containers for Data Science and Data Engineering



Open Data Science Conference
East 2024

<https://github.com/mafudge/odsc24>

Michael Fudge

Professor of Practice

mafudge@syr.edu

Syracuse University, iSchool

<https://ischool.syr.edu>