

BIOS 259: The Art of Reproducible Science

A Hands-on Approach | A **Stanford** Biosciences Mini-course

Day 04: Containerization using Docker and Singularity

Aziz Khan <azizk@stanford.edu>

<https://github.com/asntech/bios259-w24/>



Course schedule

Date	Topic	Time	Location
02.26.2024 – Monday	Introduction to reproducibility and setting up	10:00-13:00	Alway Building M218A
02.28.2024 – Wednesday	Version Control (Git/GitHub)	10:00-13:00	M218A
03.01.2024 – Friday	Dependency management (Conda, Mamba, Bioconda)	10:00-13:00	M218A
03.04.2024 – Monday	Containerization (Docker, Singularity)	10:00-13:00	M218A
03.06.2024 – Wednesday	Workflows (Snakemake, Nextflow/nf-core)	10:00-13:00	M218A
03.08.2024 – Friday	Documentation (FAIR data and open code) and wrap up	10:00-13:00	Li Ka Shing LK208

M218A: <https://25live.collegenet.com/pro/stanfordson#/?home/location/1454/details>

LK208: <https://25live.collegenet.com/pro/stanfordson#/?home/location/1149/details>

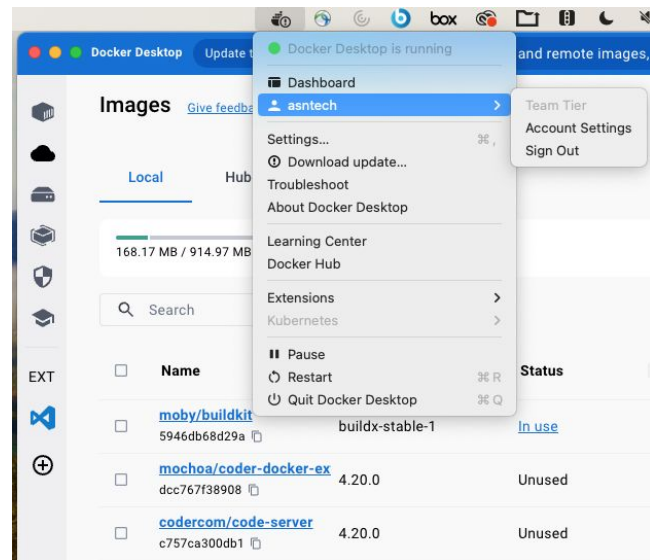
Office hours

Date	Instructor name	Time	Location
02.29.2024 – Thursday	Aziz	10:00-11:00	BMI 4022
03.05.2024 – Tuesday	Aziz	13:00-14:00	BMI 4022
03.07.2024 – Thursday	Aziz	10:00-11:00	BMI 4022

Pre-class software installations and account creation

Make sure you have:

- **Docker Desktop** - <https://www.docker.com/products/docker-desktop>
 - Make sure it is running
- **Docker Hub** - <https://hub.docker.com/signup>
 - If you haven't already, please create a Docker Hub account
 - Make sure you're logged in using Docker Desktop
- **Singularity** - <https://singularity.hpcng.org/>
 - You can also use Singularity on SCG or Sherlock



Learning goals

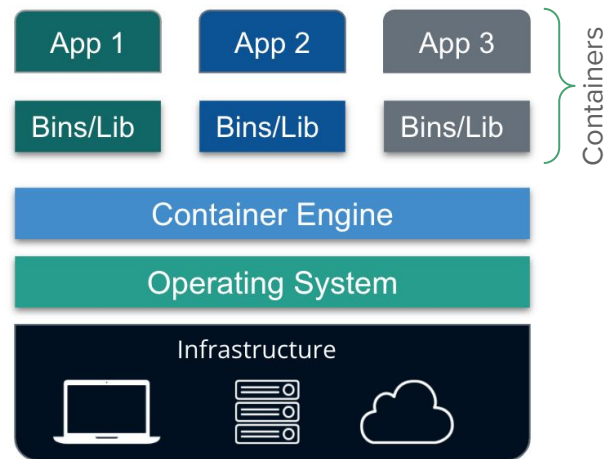
- Understand the concept of containerization and its benefits
- Create Docker containers from existing images and customize them according to their requirements
- Manage Docker containers, including starting, stopping, and deleting
- Export Docker images to Singularity and run on HPC system

What is a container?

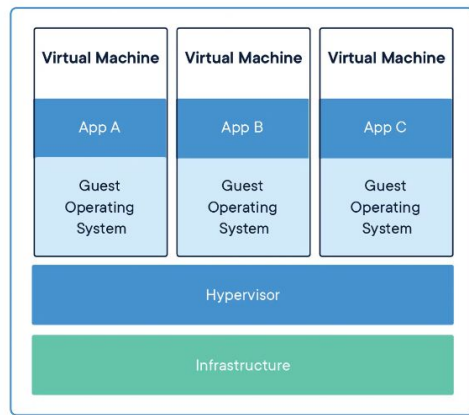
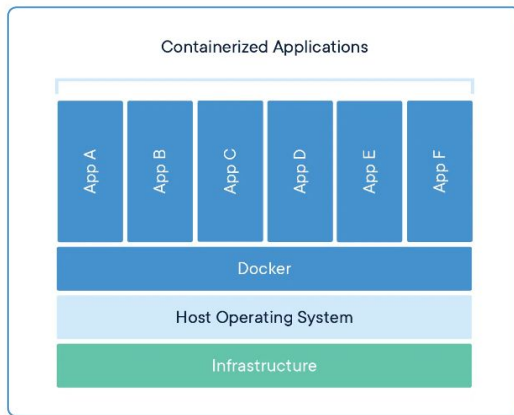
A container is an **isolated** environment that contains **all the** software **dependencies/versions** needed for your **code to run**.

Container has no knowledge of your operating system, or your local files.

It runs on the environment provided to you by Docker Desktop/Engine.



Containers vs. Virtual machines



<https://www.docker.com/resources/what-container/>

Containers

- Containers are lightweight, portable, and efficient
- Encapsulate an application and its dependencies, **sharing the host operating system** kernel
- Isolation is achieved through namespaces and control groups
- Immutable and built from layers, allowing for fast deployment and scaling
- **Examples:** Docker, Kubernetes, Singularity

Virtual machines (VMs)

- VMs are full-fledged virtualized environments and not efficient
- Each VM runs its **own operating system**, complete with its own kernel
- Isolation is achieved at the hardware level through hypervisors
- VMs require more resources and have slower startup times
- **Examples:** VMware, VirtualBox, Hyper-V

Why containers?

Containers can be **built** to bundle all the necessary ingredients (**data, code, environment**).

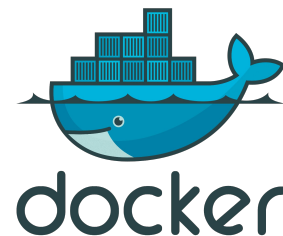
- **Lightweight:** Share the host OS kernel, reducing overhead
- **Portable:** Can run on any platform with Docker installed
- **Scalable:** Easily deploy and scale applications with Docker containers
- **Consistency:** Ensure consistent environments across development, testing, and production

Popular container implementations are **Docker** and **Singularity**

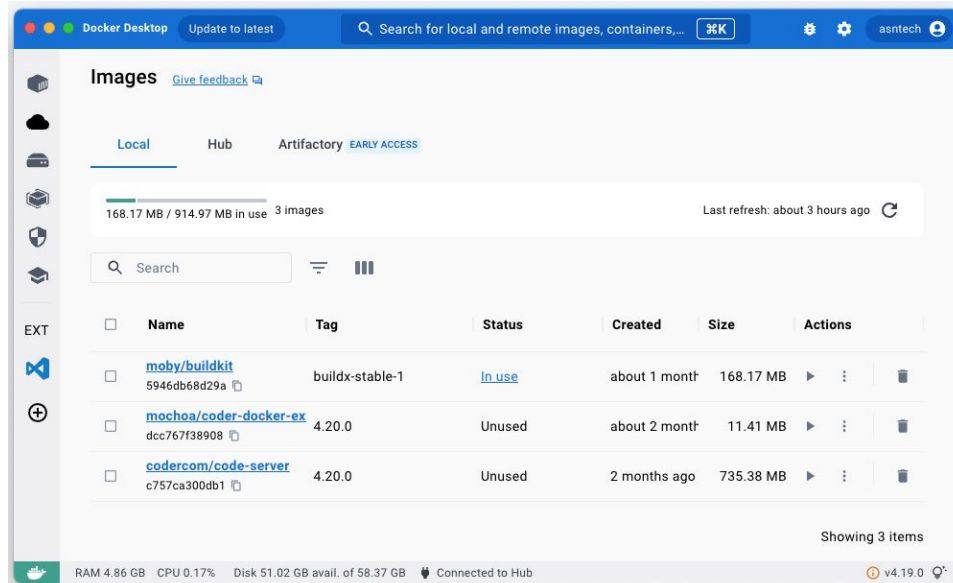


Docker

Docker is a platform (by Docker, Inc.) that can **build, share, and run containers anywhere** — without tedious environment configuration or management.

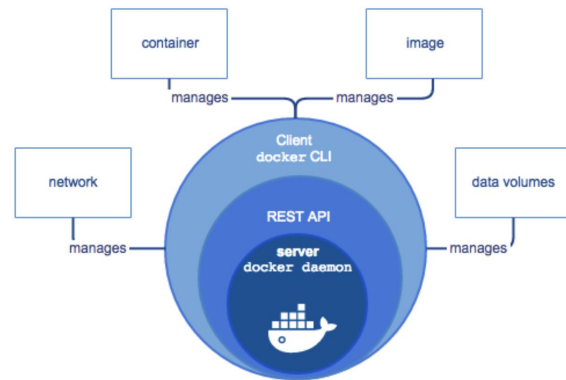


Docker can package our code or application and run it anywhere, on any machine with Docker

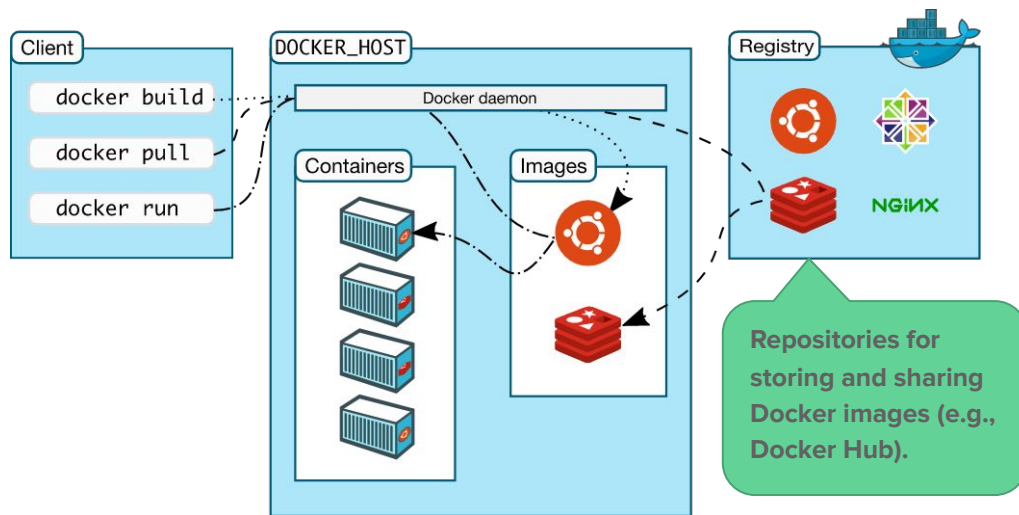


Docker architecture

- **Docker Engine:** The core component responsible for building, running, and managing containers



- **Images:** Immutable templates used to create containers.
- **Containers:** Runnable instances of Docker images.



<https://eglerean.github.io/reproducible-research/07-docker/>

Dockerfile vs. Image vs. Container

Dockerfile is a recipe which creates a container based on an image and applies small changes to it


Image is like a blueprint. It is immutable

Container is an instance of an image (process)

Dockerfile

- A text file that **contains instructions** for building a Docker image
- Commands to install dependencies, configure settings, and set up the container environment.

Dockerfile

 Copy code

```
# Use an official Python runtime as the base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Components of Dockerfile

FROM: Specifies the base image for the container

RUN: Executes commands inside the container during image build time

COPY/ADD: Copies files and directories from the host into the container

WORKDIR: Sets the working directory for subsequent instructions

EXPOSE: Exposes ports from the container to the host

ENV: Sets environment variables inside the container

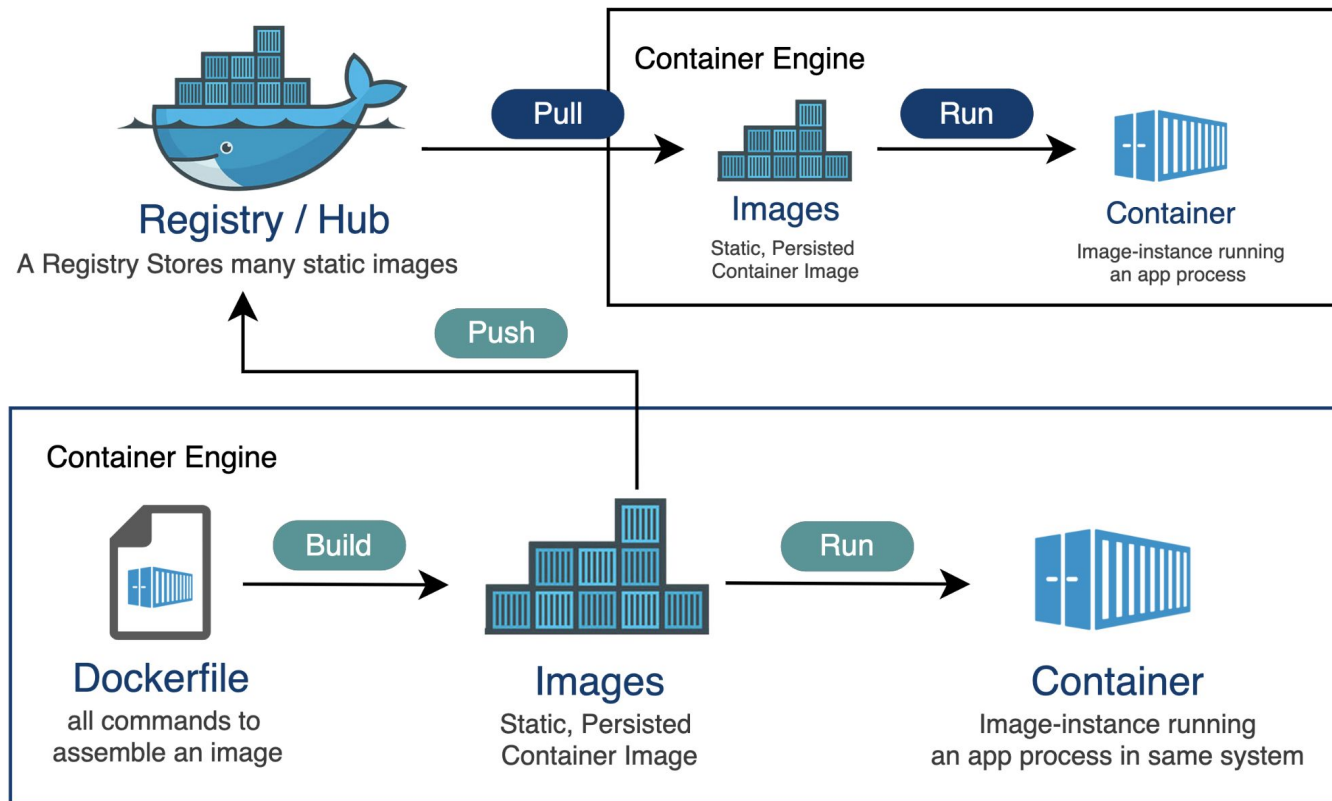
CMD/ENTRYPOINT: Specifies the command or script to run when the container starts

Dockerfile

 Copy code

```
FROM python:3.9-slim
RUN apt-get update && apt-get install -y \
    gcc \
    && rm -rf /var/lib/apt/lists/*
COPY . /app
WORKDIR /app
EXPOSE 80
CMD ["python", "app.py"]
```

The workflow



<https://community.sap.com/t5/technology-blogs-by-sap/use-private-registry-for-containerize-a-cap-application-part-1/ba-p/13541667>

Docker essential commands

\$ docker <command> --help

\$ docker run: Create/run a container from an image

\$ docker build: Create an image from Dockerfile

\$ docker push: Push an image to a registry

\$ docker pull: Pull an image from a registry

\$ docker stop: Stop a running container

\$ docker start: Start a stopped container

\$ docker ps: List running containers (-a shows all)

\$ docker rm: Remove a container (-f force delete)

\$ docker rmi: Remove an image

\$ docker rename: Rename a container

\$ docker image ls: List Docker images

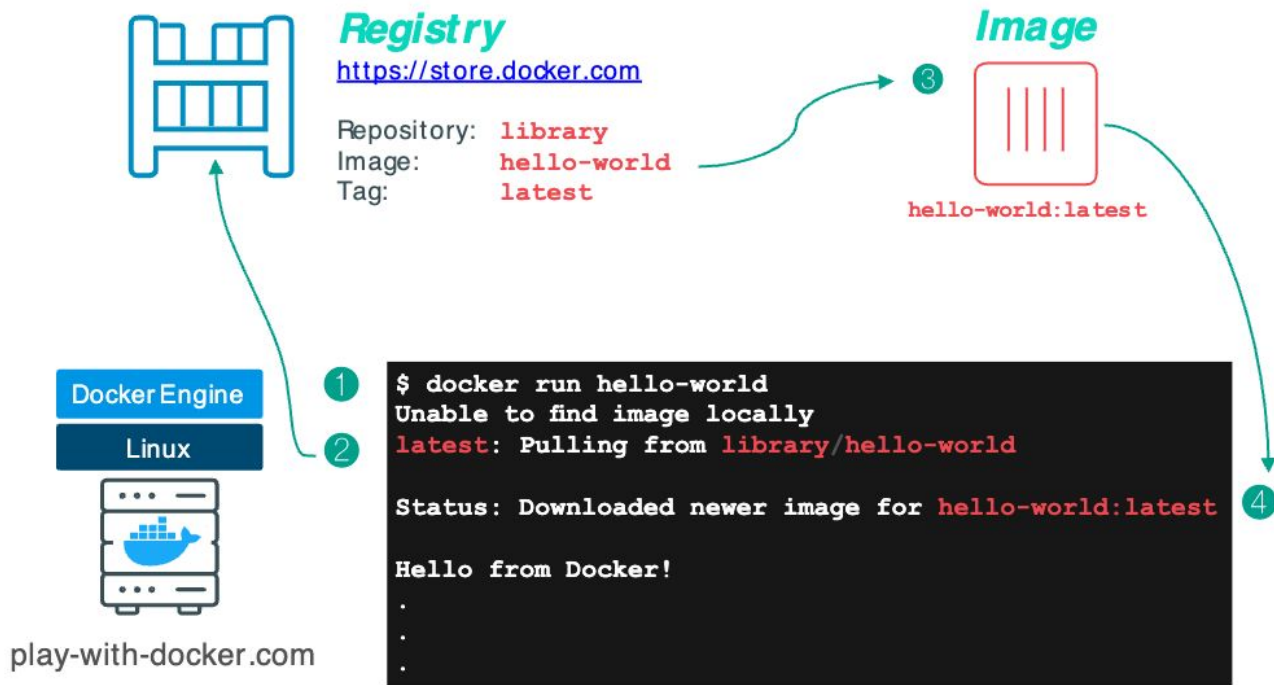
\$ docker container ls: List Docker containers

\$ docker search: Search Docker Hub for images

Let's run our first Container

Try this:

`$ docker run hello-world`



Build an image

```
$ docker build < >
```

```
$ docker build --tag bios259:w24 -f  
./Dockerfile
```

```
# Do not use cache when building  
--no-cache
```

```
#Set target platform for build  
--platform
```

Run a container from an image

```
$ docker run <image tag/id>
```

```
$ docker run bios259:w24
```

```
# additional arguments
```

```
# -it :run as interactive
```

```
# -d :run in the background
```

```
# -v :bind mount a volume
```

```
# -e :set environment variables
```

Push an image to Docker Hub

```
$ docker push <username>/<image>:<tag>
```

For example

```
$ docker push asntech/bios259:w24
```

slido



What is the correct order for Docker concepts (Dockerfile, Container, Image)?

① Click **Present with Slido** or install our [Chrome extension](#) to activate this poll while presenting.

Peer instruction exercise

What is the correct order for Docker concepts (Dockerfile, Container, Image)?

- A. Dockerfile > Container > Image
- B. Container > Dockerfile > Image
- C. Dockerfile > Image > Container
- D. Image > Container > Dockerfile

Hands-on exercise 1

30 minutes

Pair in two and let's create our first Docker image and share

Edit exercise1 files to build and share Docker image using an existing Dockerfile

Open the URL for details:

<https://github.com/asntech/bios259-w24/tree/main/04-containers/exercise1>

slido



What is the difference between the CMD and RUN commands in a Dockerfile?

① Click **Present with Slido** or install our [Chrome extension](#) to activate this poll while presenting.

Peer instruction

What is the difference between the **CMD** and **RUN** commands in a Dockerfile?

- A. There is no difference.
- B. CMD is executed when building an image while RUN is executed when containers are started.
- C. RUN is executed when building an image while CMD is executed when containers are started.
- D. There is no difference unless the container is started interactively.

Docker is not for shared systems

- Docker requires a **root/sudo** access
- Therefore it can not be made available on **shared scientific** and high-performance computing (**HPC**) systems

Singularity for scientific computing



- Singularity is an **open-source container platform** designed for scientific and high-performance computing (**HPC**) environments
- Provides compatibility with Docker images, allowing Docker containers to be converted into Singularity containers

Benefits of Singularity:

- Singularity containers can be **run without root privileges**
- **Compatibility with HPC** environments
- Seamless integration with existing Docker workflows/registries

Exporting Docker to Singularity

Singularity build command to convert a Docker image to a Singularity image.

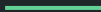
```
$ singularity build  
<singularity_image>.sif  
docker://<docker_image>
```

Run Singularity Container

Singularity build command to convert a Docker image to a Singularity image.

```
$ singularity run <singularity_image>.sif
```

Additional options can be specified when running Singularity containers, such as mounting directories, setting environment variables, or specifying run-time flags.



Register singularity

https://github.com/papaemmelab/register_apps

```
$ git clone  
https://github.com/papaemmelab/register_apps.git
```

```
$ cd register_apps && python setup.py install
```

```
$ register_singularity \  
  --target bios259 \  
  --command bios259 \  
  --image_url docker://asntech/bios259:w24 \  
  --bindir /example/bin \  
  --optdir /example/opt \  
  --volumes /oak /oak
```

Hands-on exercise 2

60 minutes

Build and share a Docker image
and run it as Singularity container

Keep the same pairs

Use the pre-build image by your
peer in exercise 1 as the starting
point

Open the URL for details:

<https://github.com/asntech/bios259-w24/tree/main/04-containers/exercise2>

Pros and cons of containers

- Allow for **seamlessly** moving workflows across different platforms.
- Much more **lightweight** than virtual machines.
- Works on **any machine** - not only yours.
- For software with many **dependencies** a best way to preserve the computational experiment for future reproducibility.

- Containers can have **security vulnerabilities** which can be exploited.
- Can be used to hide away software installation problems and thereby discourage good software development practices.
- It may not be clear whether to record the environment in the image part or the recipe part.

Clicker question

You're tasked with deploying a computational workflow on Sherlock HPC. Which of the following statements accurately describes this scenario?

- A) Singularity seamlessly converts Docker images to Singularity images, ensuring compatibility with HPC systems.
- B) Singularity provides a GUI for managing workflows, a feature absent in Docker.
- C) Singularity containers offer better performance than Docker in HPC environments.
- D) Singularity enables Docker images to be converted, ensuring compatibility and offering rootless operation.
- E) Transitioning requires manual reconfiguration of Docker containers for HPC deployment.

Clicker question

You're tasked with deploying a computational workflow on Sherlock HPC. Which of the following statements accurately describes this scenario?

- A) Singularity seamlessly converts Docker images to Singularity images, ensuring compatibility with HPC systems.
- B) Singularity provides a GUI for managing workflows, a feature absent in Docker.
- C) Singularity containers offer better performance than Docker in HPC environments.
- D) Singularity enables Docker images to be converted, ensuring compatibility and offering rootless operation.**
- E) Transitioning requires manual reconfiguration of Docker containers for HPC deployment.

Continuous Integration/Continuous Deployment (CI/CD) with Github workflows

The screenshot shows the GitHub Actions interface for a repository named 'asntech / bios259-w24'. The top navigation bar includes links for Code, Issues, Pull requests, Actions (which is highlighted), Projects, Wiki, Security, Insights, and Settings. Below the navigation bar, the heading 'Choose a workflow' is displayed, followed by a brief explanation of CI/CD and a link to 'set up a workflow yourself'. A search bar for workflows is present. The 'Suggested for this repository' section features three workflow cards: 'Docker image' (by Docker), 'Python Package using Anaconda' (by Python), and 'Publish Python Package' (by Python). Each card includes a description, a 'Configure' button, and a status indicator. Below this, the 'Continuous integration' section displays four more workflow cards: 'Publish Docker Container' (by Docker), 'SLSA Generic generator' (by Open Source Security Foundation), 'Python package' (by Python), and 'Laravel' (by PHP). Each card also includes a description, a 'Configure' button, and a status indicator. A 'View all' link is located at the end of the 'Continuous integration' section.

asntech / bios259-w24

Type to search

Code Issues Pull requests **Actions** Projects Wiki Security Insights Settings

Choose a workflow

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself](#) →

Search workflows

Suggested for this repository

Docker image

By GitHub Actions

Build a Docker image to deploy, run, or push to a registry.

[Configure](#)

Dockerfile

Python Package using Anaconda

By GitHub Actions

Create and test a Python package on multiple Python versions using Anaconda for package management.

[Configure](#)

Python

Publish Python Package

By GitHub Actions

Publish a Python Package to PyPI on release.

[Configure](#)

Python

Continuous integration

Publish Docker Container

By GitHub Actions

Build, test and push Docker image to GitHub Packages.

[Configure](#)

Dockerfile

SLSA Generic generator

By Open Source Security Foundation (OpenSSF)

Generate SLSA3 provenance for your existing release workflows

[Configure](#)

Go

Python package

By GitHub Actions

Create and test a Python package on multiple Python versions.

[Configure](#)

Python

Laravel

By GitHub Actions

Test a Laravel project.

[Configure](#)

PHP

[View all](#)

asntech / bios259-w24

<> Code Issues Pull requests **Actions** Projects Wiki Security Insights

Actions

New workflow

All workflows

Docker Exercise 1 Image CI

Management

Caches

Runners

All workflows

Showing runs from all workflows

Filter workflow runs

1 workflow run

Event	Status	Branch	Actor
✓	Update docker-image.yml	main	asntech
Docker Exercise 1 Image CI #6: Commit c943250 pushed by asntech			13 minutes ago ...

bios259-w24 / .github / workflows / docker-image.yml

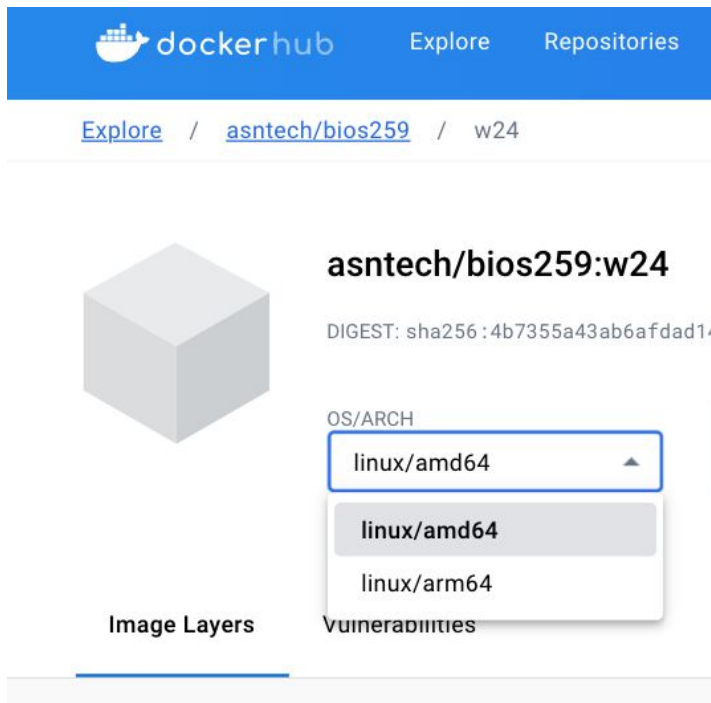
asntech Update docker-image.yml ✓ c943250 · 16 minutes ago History

Code Blame 22 lines (17 loc) · 427 Bytes

Raw Copy Download Edit

```
1 name: Docker Exercise 1 Image CI
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
9 jobs:
10
11   build:
12
13     runs-on: ubuntu-latest
14
15     steps:
16     - uses: actions/checkout@v3
17     - name: Build the Docker image
18       run: |
19         # Navigate to the desired directory
20         cd ../04-containers/exercise1/
21         # Build the Docker
22         docker build . --file Dockerfile --tag bios259:$(date +%s)
```

Build multi-platform images



```
$ docker buildx create --use
```

```
$ docker buildx ls
```

```
$ docker buildx build --platform  
linux/amd64,linux/arm64 --push --tag  
asntech/bios259:w24 exercise1
```

<https://hub.docker.com/r/asntech/bios259/tags>

Long-term archive your docker image

```
$ docker image save <>
```

> Upload to Zenodo

```
$ docker load -i <name>
```



```
# Make sure you've the docker locally
```

```
$ docker image ls
```

```
$ docker image save asntech/bios259:w24  
-o bios259-w24.tar
```

```
$ gzip bios259-w24.tar
```

```
# Load the container
```

```
$ docker load -i bios259-w24.tar.gz
```

docker-compose: multi-container applications

- Docker Compose is a tool for defining and running multi-container Docker applications.
- Simplifies the process of defining and managing complex multi-container Docker applications.
- Streamlines development, testing, and deployment workflows.

Key Features:

- Service Definitions
- Dependency Management
- Environment Configuration
- Scalability

docker-compose.yml

yml

Copy code

```
version: '3.8'

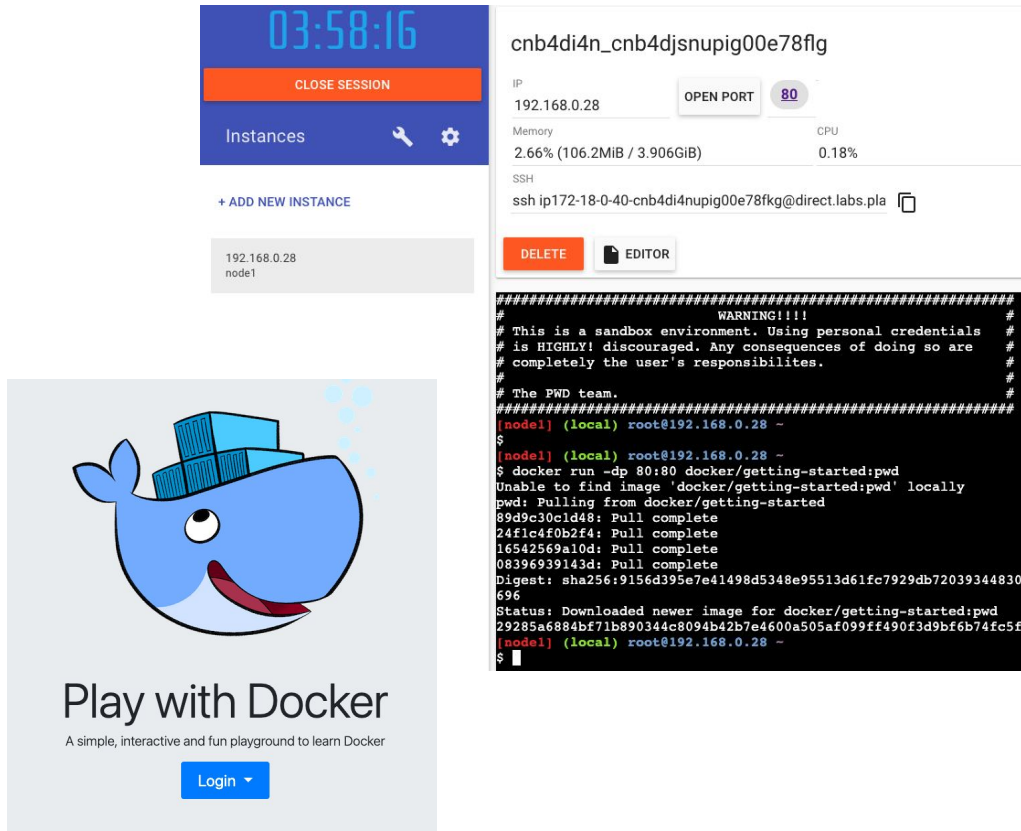
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
  database:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: password
```

```
$ docker-compose -f docker-compose.yml up
```

More - <https://docs.docker.com/compose/>

Play with Docker in the cloud - homework

1. Log into <https://labs.play-with-docker.com/> to access your PWD terminal
2. Click Add New Instance
3. Type the following command in your PWD terminal: *docker run -dp 80:80 docker/getting-started:pwd*
4. Wait for it to start the container and click the **Open Port 80** button
5. Have fun!

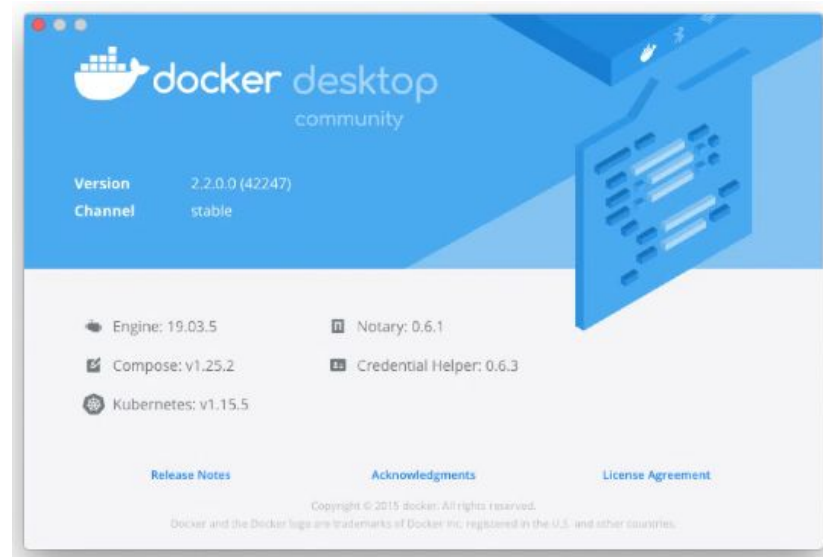


The screenshot shows the Play with Docker interface. At the top, there's a blue header with a timer '03:58:16' and a 'CLOSE SESSION' button. Below it, there's a section for 'Instances' with a '+ ADD NEW INSTANCE' button. A new instance named 'node1' with IP '192.168.0.28' is listed. To the right, there's a detailed view of the instance 'cnb4di4n_cnb4djsnupig00e78flg'. It shows the IP '192.168.0.28', memory usage '2.66% (106.2MiB / 3.906GiB)', and CPU usage '0.18%'. There's an 'OPEN PORT 80' button. Below this, there's a terminal window showing the command `docker run -dp 80:80 docker/getting-started:pwd` being executed. The terminal output shows a warning about the sandbox environment, the Docker team's message, and the successful pull and execution of the container. The container is named 'node1' and is running on the host '192.168.0.28'.

More advanced training material: <https://training.play-with-docker.com/>

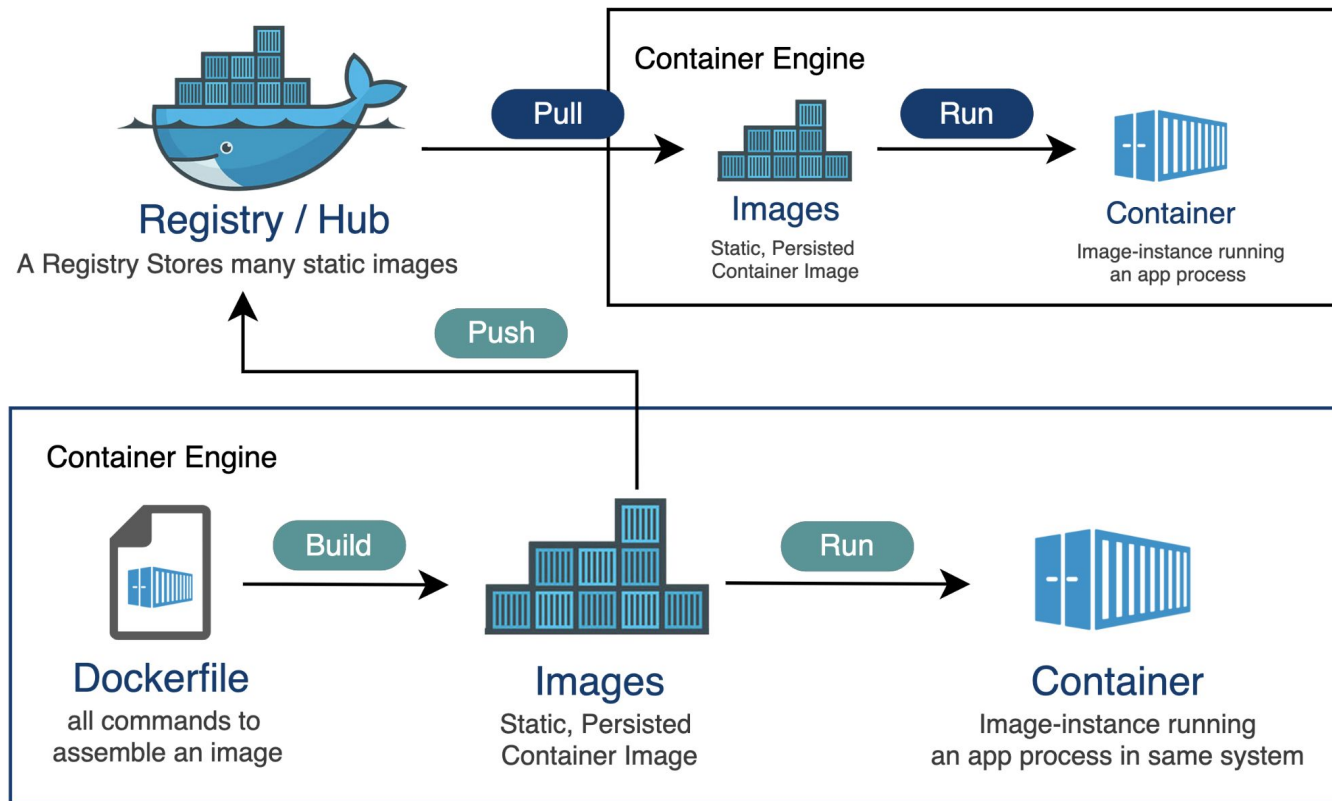
Play with Docker on your desktop - homework

1. Open Docker Desktop. (Download if you don't have it - <https://www.docker.com/products/docker-desktop>).
2. Type the following command in your PWD terminal: *docker run -dp 80:80 docker/getting-started*
3. Open your browser to <http://localhost>
4. Have fun!
5. docker container ls
6. docker stop <container_id>



More advanced training material: <https://training.play-with-docker.com/>

Summary



<https://community.sap.com/t5/technology-blogs-by-sap/use-private-registry-for-containerize-a-cap-application-part-1/ba-p/13541667>

Resources

- <https://docs.docker.com/guides/get-started/>
- <https://carpentries-incubator.github.io/docker-introduction/index.html>
- <https://carpentries-incubator.github.io/docker-introduction/index.html>
- <https://training.play-with-docker.com/>
- <https://docs.docker.com/compose/>