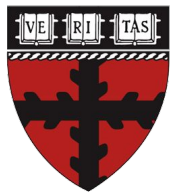
Lecture 6: Data Labeling
Data Versioning

AC215

Pavlos Protopapas
SEAS/Harvard



Outline

1. Data Labeling
2. Data Versioning

Outline

1. **Data Labeling**
2. Data Versioning

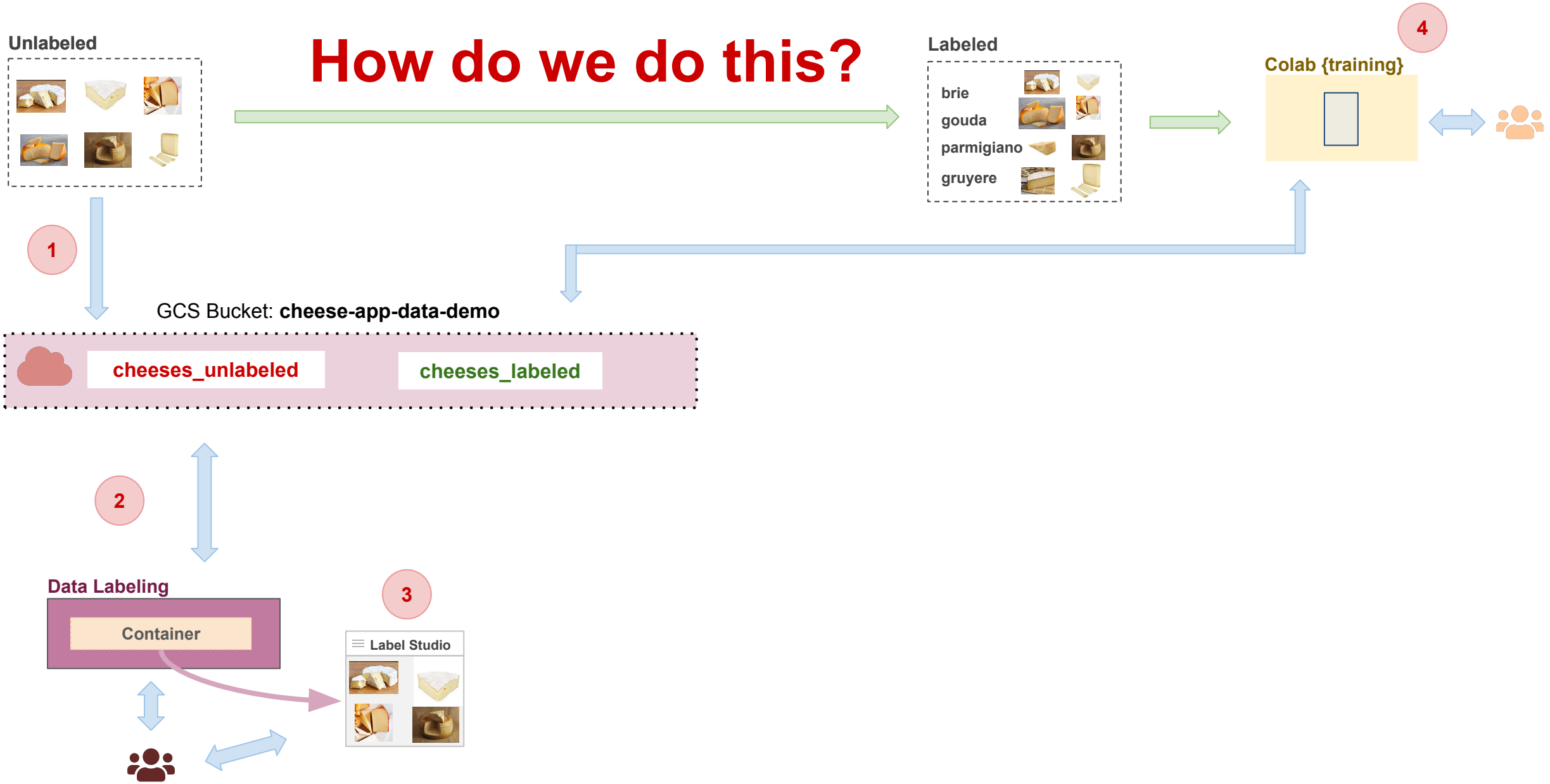
Tutorial (T6): Data Labeling

In this tutorial, we will learn how to perform labeling.

The task involves labeling images of cheeses such as Brie, Gruyère, Gouda, and Parmigiano.

We will begin with images scraped from the web and then use Label Studio to label them.

Cheese App Data Pipeline



Tutorial (T6): Data Labeling

To overcome some of the challenges of labeling, Label Studio allows us to streamline the process.

We want to avoid uploading our data to any system, so we will run it locally as a Docker container.

Tutorial (T6): Data Labeling

Before we proceed, we need to familiarize ourselves with some new concepts:

- Docker Network
- Cloud Storage and Credentials
- Docker Compose

Tutorial (T6): Data Labeling

Before we proceed, we need to familiarize ourselves with some new concepts:

- **Docker Network**
- Cloud Storage and Credentials
- Docker Compose

Docker Network

Docker networks allow different containers to communicate with each other in a controlled environment over a **virtual isolated local network**.

Each network acts like a private channel, ensuring that containers can talk to each other while staying separate from other containers that don’t need to interact.

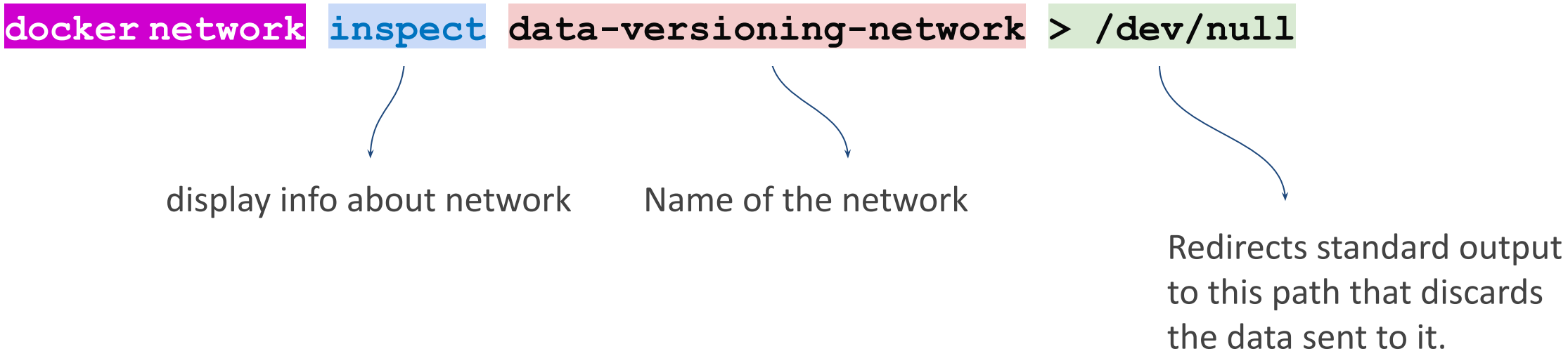
Typically, communication happen using predefined ports, such as localhost:8080.

Docker Network

In the following command:

```
docker network inspect data-versioning-network >/dev/null 2>&1
|| docker network create data-versioning-network
```

The first part checks if the network `data-versioning-network` exists. It sends the output to `/dev/null`, discarding it. Same as the error.

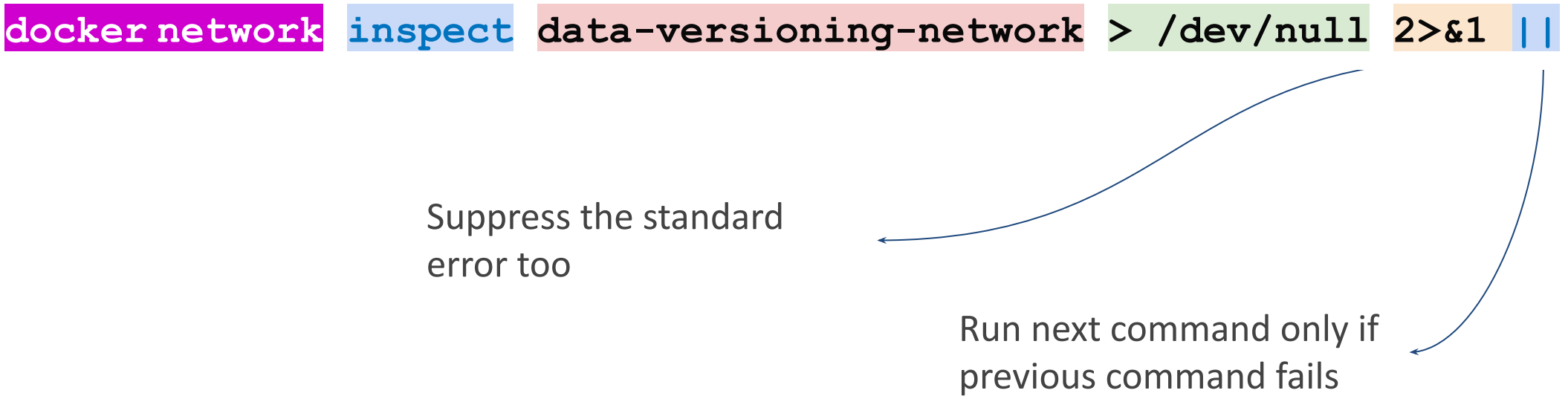


Docker Network

In the following command:

```
docker network inspect data-versioning-network >/dev/null 2>&1
|| docker network create data-versioning-network
```

The first part checks if the network `data-versioning-network` exists. It sends the output to `/dev/null`, discarding it. Same as the error.



Docker Network

In the following command:

```
docker network inspect data-versioning-network >/dev/null 2>&1
|| docker network create data-versioning-network
```

The final part creates the network if it does not exist.

```
docker network inspect data-versioning-network > /dev/null 2>&1 ||
docker network create data-versioning-network
```

creates the network

Desired name of the docker network

Tutorial (T6): Data Labeling

Before we proceed, we need to familiarize ourselves with some new concepts:

- Docker Network
- **Cloud Storage and Credentials**
- Docker Compose

Containers and Credentials

By now you are familiar with GCP Buckets. They allow to store information, without any VM or container attached to it.

To ensure privacy, by default they cannot be accessed from outside.
If we want to host Label Studio and use data from the container, we require the appropriate credentials.

Creating and setting up GCP Buckets

Buckets can be created programmatically or via the GUI.

For this tutorial:

- Go to https://console.cloud.google.com/storage/browser
- Create a bucket <*bucket_name*>
- Create a folder *cheeses_unlabeled* inside the bucket
- Create a folder *cheeses_labeled* inside the bucket

- Upload the images from your local folder into the folder *cheeses_unlabeled* inside the bucket
- Configure the credentials to allow Label Studio access to the data.

Containers and Credentials: Service Account

A service account is a special type of GCP account that represents a non-human user.

It is used by applications and virtual machines (VMs) to interact with Google Cloud services programmatically.

Unlike a regular user account, which is linked to an individual end-user, a service account belongs to an application or a service running on GCP.

Tutorial (T6): Data Labeling

Before we proceed, we need to familiarize ourselves with some new concepts:

- Docker Network
- Cloud Storage and Credentials
- **Docker Compose**

Docker Compose

For this tutorial we used shell scripts to automate the deployment of containers.

Docker Compose is the standard way to build and run sequences of containers that depend on each other.

They require a docker compose YAML file, for defining and running multi-container Docker applications.

With a single command, you **build** and **start** all the containers.

docker-compose.yml

```yaml
version: "3.8"

# Define network that the various docker containers will share
networks:
    default:
        name: data-labeling-network
        external: true

services:
    data-label-cli:
        image: data-label-cli
        container_name: data-label-cli
        volumes:
            - ../secrets:/secrets
            - ../data-labeling:/app
        environment:
            GOOGLE_APPLICATION_CREDENTIALS: /secrets/data-service-account.json
            GCP_PROJECT: "ac215-project"
            GCP_ZONE: "us-central1-a"
            GCS_BUCKET_NAME: "cheese-app-data-demo"
            LABEL_STUDIO_URL: "http://data-label-studio:8080"
        depends_on:
            - data-label-studio

    ...
```

List of containers to run

Volumes to mount to the container

Environment variables to set inside container

Specifies if this container depends on another container that needs to be started first

docker-compose.yml continued

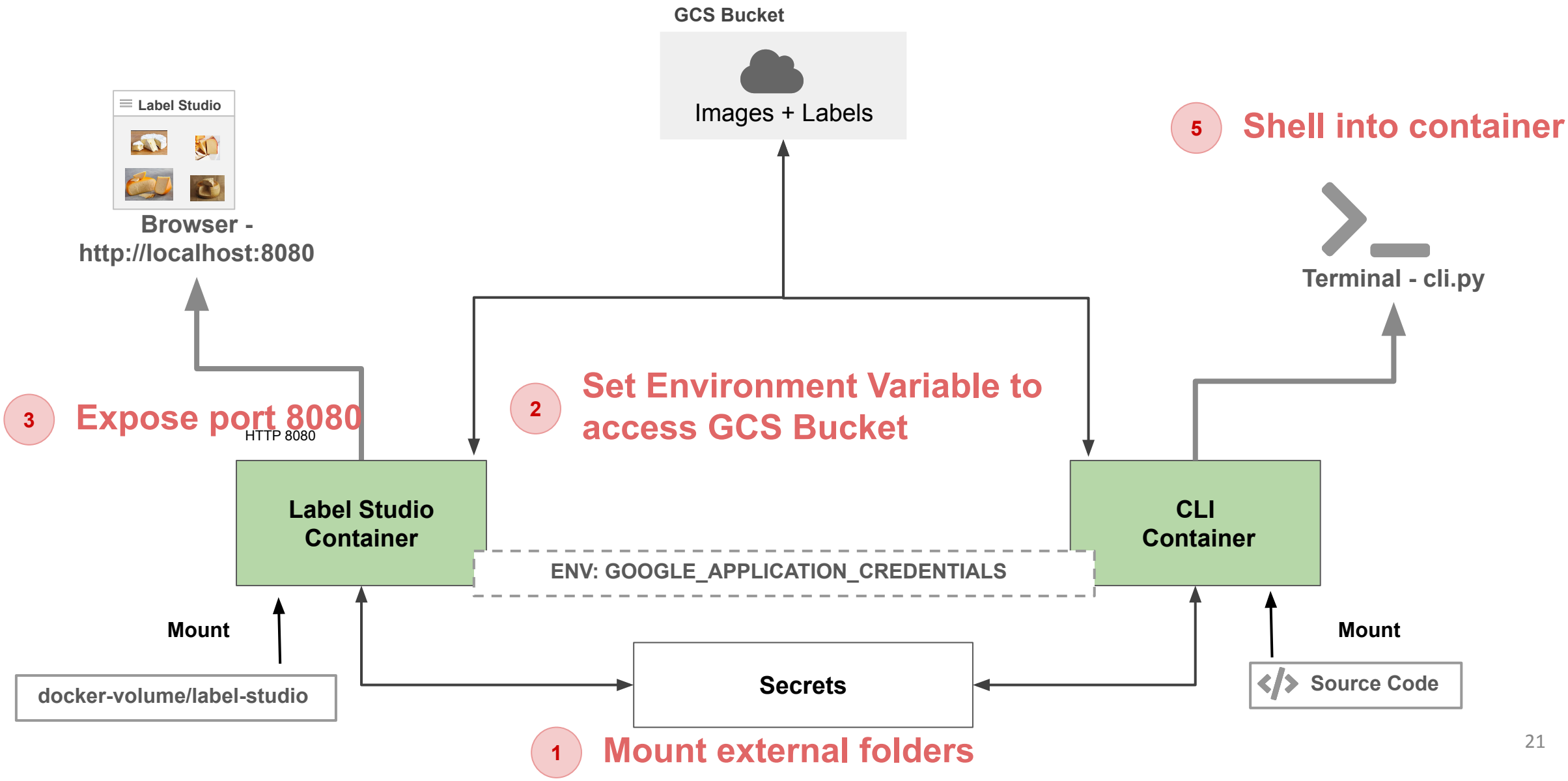
```yaml
data-label-studio:
    image: heartexlabs/label-studio:latest
    container_name: data-label-studio
    ports:
        - 8080:8080
    volumes:
        - ./docker-volumes/label-studio:/label-studio/data
        - ../secrets:/secrets
    environment:
        LABEL_STUDIO_DISABLE_SIGNUP_WITHOUT_LINK: "true"
        LABEL_STUDIO_USERNAME: "pavlos@seas.harvard.edu"
        LABEL_STUDIO_PASSWORD: "awesome"
        GOOGLE_APPLICATION_CREDENTIALS: /secrets/data-service-account.json
        GCP_PROJECT: "ac215-project"
        GCP_ZONE: "us-central1-a"
```

Port to expose from inside container to the host outside

Volumes to mount to the container

Environment variables to set inside container

Tutorial (T6): Data Labeling: Label Studio + CLI



Tutorial (T6): Data Labeling

Steps to create a Data Pipeline to use unlabeled images and create a processes to label the dataset:

- Create a GCS bucket to store all data.
- Run Data Labeling Container.
- For detailed instructions, please refer to the following link
  - Data Labeling. (https://github.com/dlops-io/data-labeling)

Outline

1. Data Labeling
2. **Data Versioning**

Why Data Versioning?

Keep Track:

- Monitor data changes and stay organized.

Protection:

- Backup data and restore earlier versions if needed.

Compliance:

- Meet regulations by tracking changes, making audits simple.

Collaboration:

- Let multiple users work together smoothly.

Efficiency:

- Save space by storing only the changes, not full copies.

Approaches to Data Versioning

Static Data: Only the queries are versioned since the underlying data remains unchanged over time.

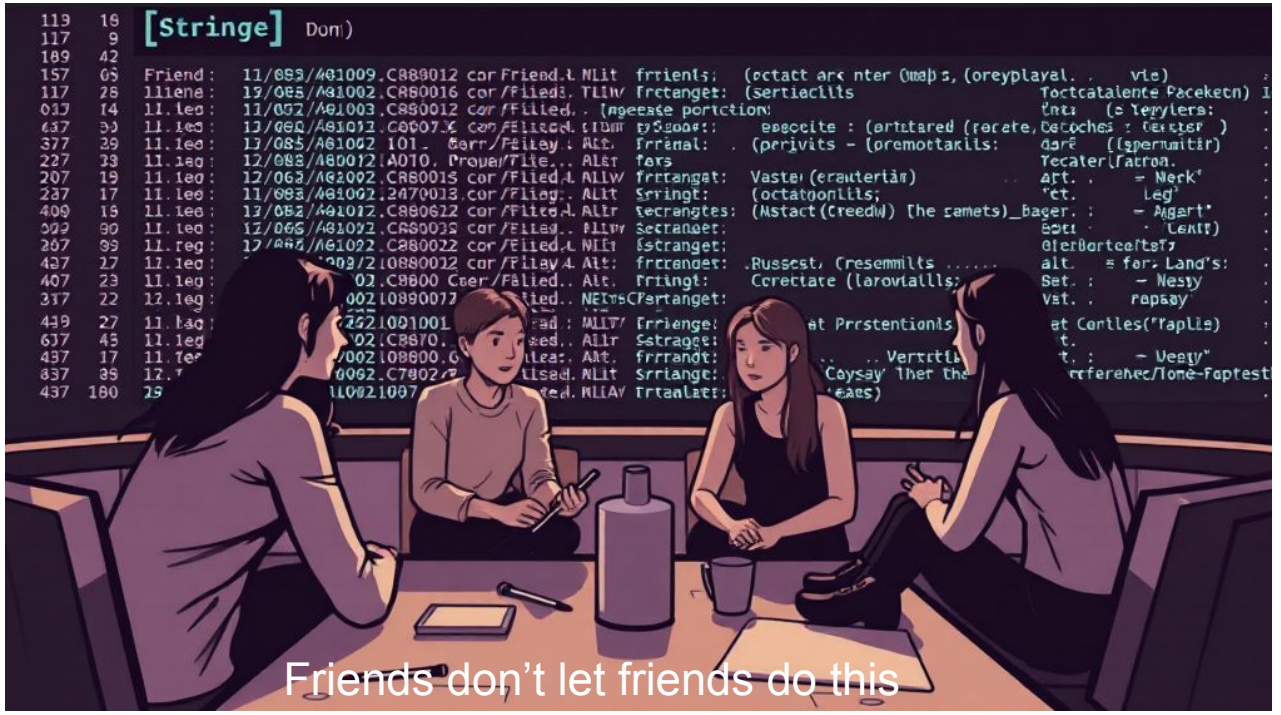
Dynamic Data: A full snapshot of the dataset is taken at specific points in time to capture changes.

All Other Cases: Versioning is based on tracking differences (deltas) between data states, enabling efficient storage and management of updates.

And … “wait for it”

Approaches to Data Versioning

```
(base) pavlosprotopapas@Pavloss-Mac-mini PINNS-TransferLearning-Paper % ls
ICML-subm-DM-v1.tex                         ICML-subm-final-final-v2.tex
ICML-subm-PP-v1.1-IB-PO-v2.tex              ICML-subm-final-final-v3-final.tex
ICML-subm-PP-v1.1-IB.tex                    ICML-subm-final-final-v3.tex
ICML-subm-PP-v1.1.tex                       ICML-subm-final-final.tex
ICML-subm-PP.tex                            ICML-subm-final-v2.tex
ICML-subm-final-final-v1.tex                ICML-subm-final.tex
(base) pavlosprotopapas@Pavloss-Mac-mini PINNS-TransferLearning-Paper %
```



Tools for Data Versioning Based on Differences

1. **DVC (Data Version Control)**
   - Git-like versioning for datasets and models.
   - Tracks changes in data and integrates with ML workflows.
2. **Delta Lake**
   - Adds version control and ACID transactions to data lakes.
   - Supports time travel for querying historical data.
3. **Pachyderm**
   - Version control for data pipelines.
   - Tracks every dataset change and supports rollback.
4. **Git LFS (Large File Storage)**
5. **Quilt**
6. **LakeFS**

Tools for Data Versioning Based on Differences:

1. **DVC (Data Version Control)**
   - Git-like versioning for datasets and models.
   - Tracks changes in data and integrates with ML workflows.
2. **Delta Lake**
   - Adds version control and ACID transactions to data lakes.
   - Supports time travel for querying historical data.
3. **Pachyderm**
   - Version control for data pipelines.
   - Tracks every dataset change and supports rollback.
4. **Git LFS (Large File Storage)**
5. **Quilt**
6. **LakeFS**

Tutorial (T7): Data Versioning

Steps to create a Data Pipeline to **version a dataset**:

- Run Data Versioning Container.
- Test data versions from Colab.
- For detailed instructions, please refer to the following link
  - Data Versioning (https://github.com/dlops-io/data-versioning).
  - Test Data Version Notebook (https://colab.research.google.com/drive/1RRQ1SIHq5IKK76R8LoQdi5LjCnND3jTq?authuser=1).

Tutorial (T7): Data Versioning

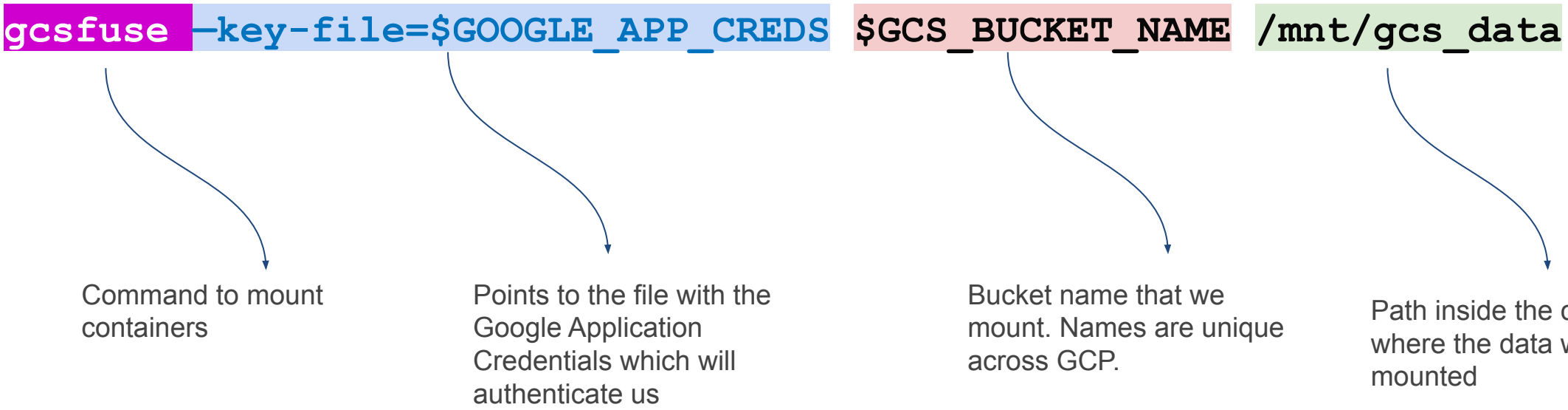
Before we proceed, we need to familiarize ourselves with some new concepts:

- Mounting disks from GCS to a container
- Entrypoint

Mounting disks from GCS to a container

In order to use the data on the container, we have to link them. This is done via gcsfuse.

```
gcsfuse --key-file=$GOOGLE_APPLICATION_CREDENTIALS
$GCS_BUCKET_NAME /mnt/gcs_data
```



Entrypoints

In Docker, entrypoints allow us to define a specific application or command that runs automatically when the container starts.
This enables us to automate tasks inside the container, or even make the `docker run` command behave like executing a standalone, containerized program.

Entrypoints

For example, an entrypoint can run a script

```
ENTRYPOINT ["/myapp/start.sh"]
```

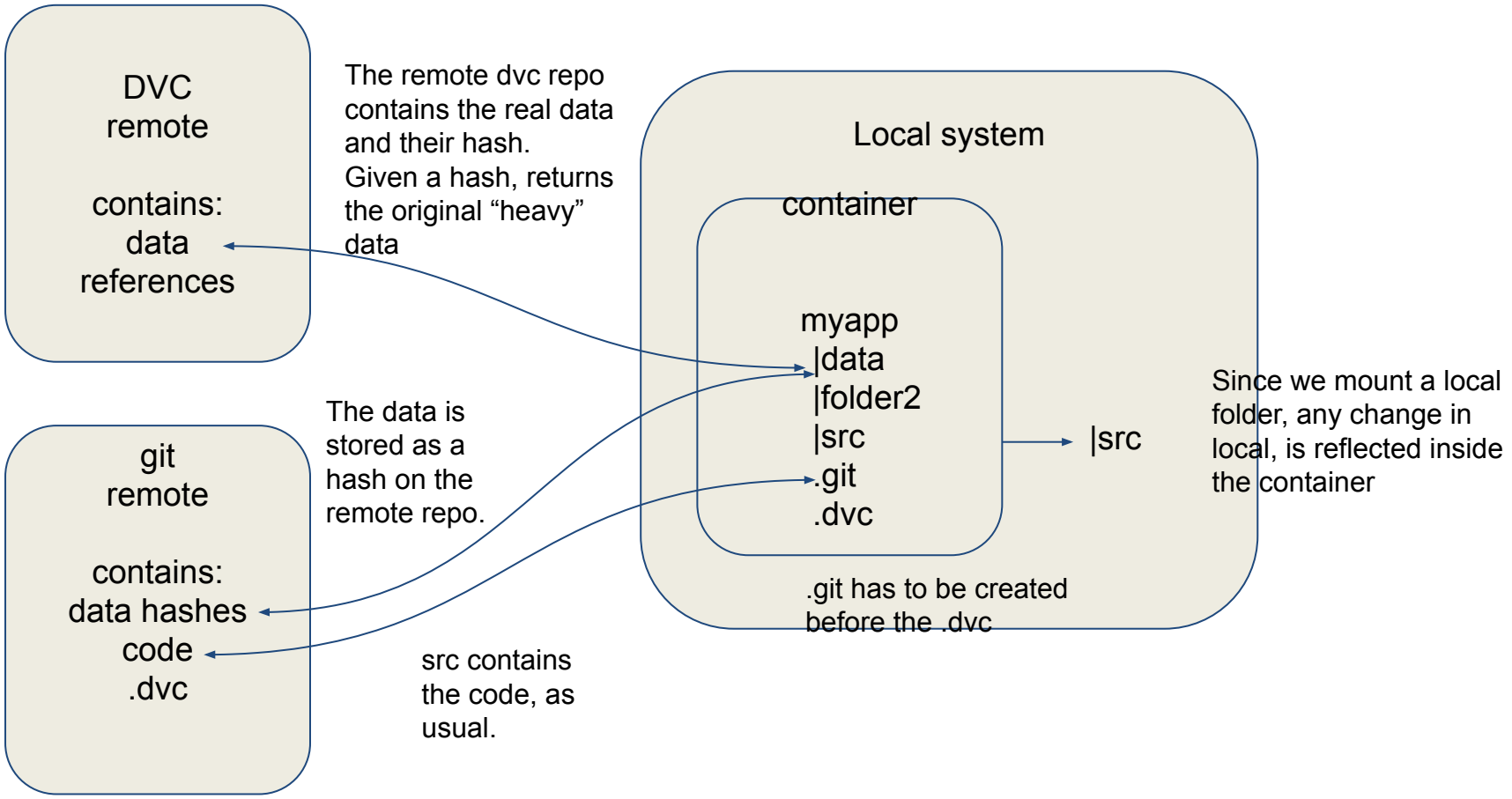
or a program with arguments

```
ENTRYPOINT ["python", "cli.py"]
```

In development containers,

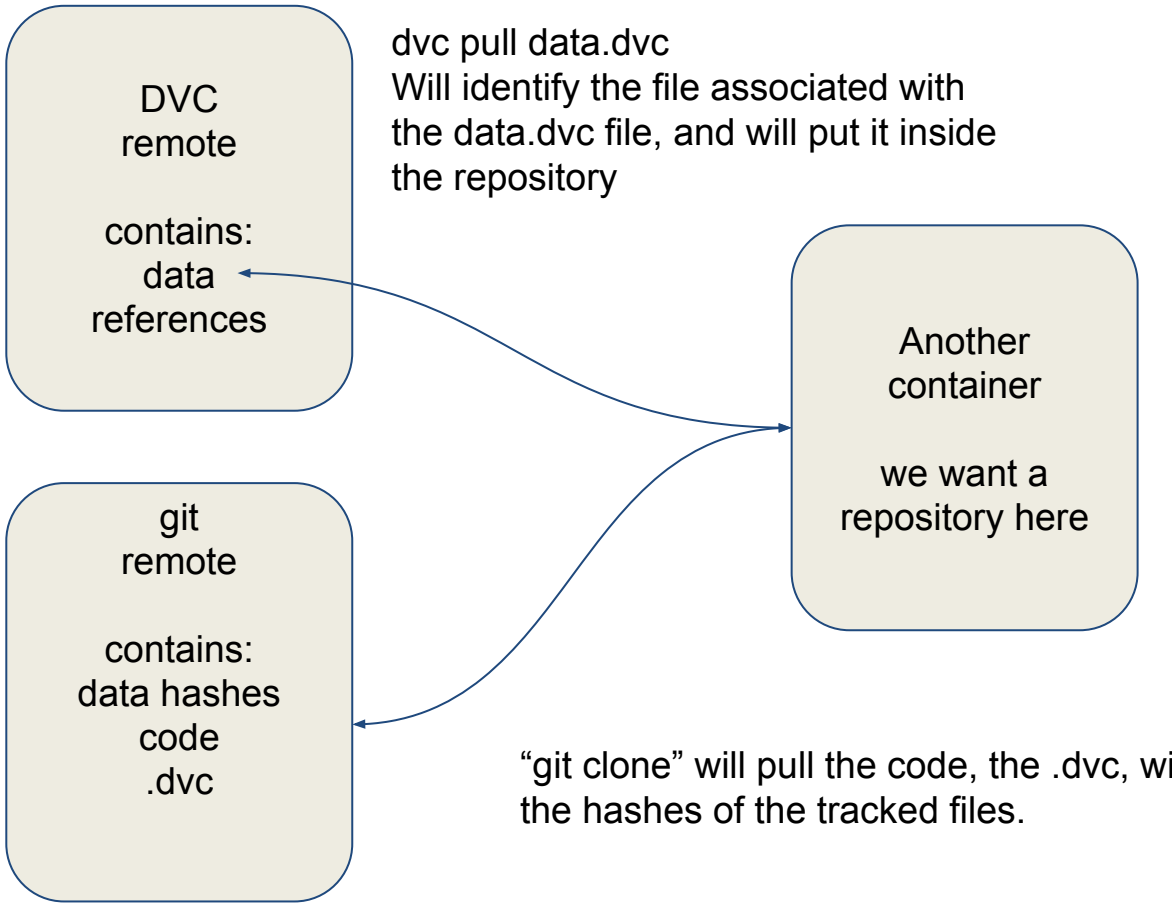
```
ENTRYPOINT ["/bin/bash"]
```

is used to open a shell terminal for interactive commands.



the dvc is contained inside the repo. Each dvc add, ignores the local “heavy” file (.gitignore). Only the hash is tracked.

The git and dvc remotes are different.
One is hosted on GitHub (for example) and the other on GCP buckets.



Logistics/Reminders

- Approx. 90% of class has project partners 🎉 - if you have formed group make sure to update this group info spreadsheet

- What makes a good project ?

THANK YOU