

Lecture 5: Docker Workflows and Data Labeling

AC215

Pavlos Protopapas
SEAS/Harvard



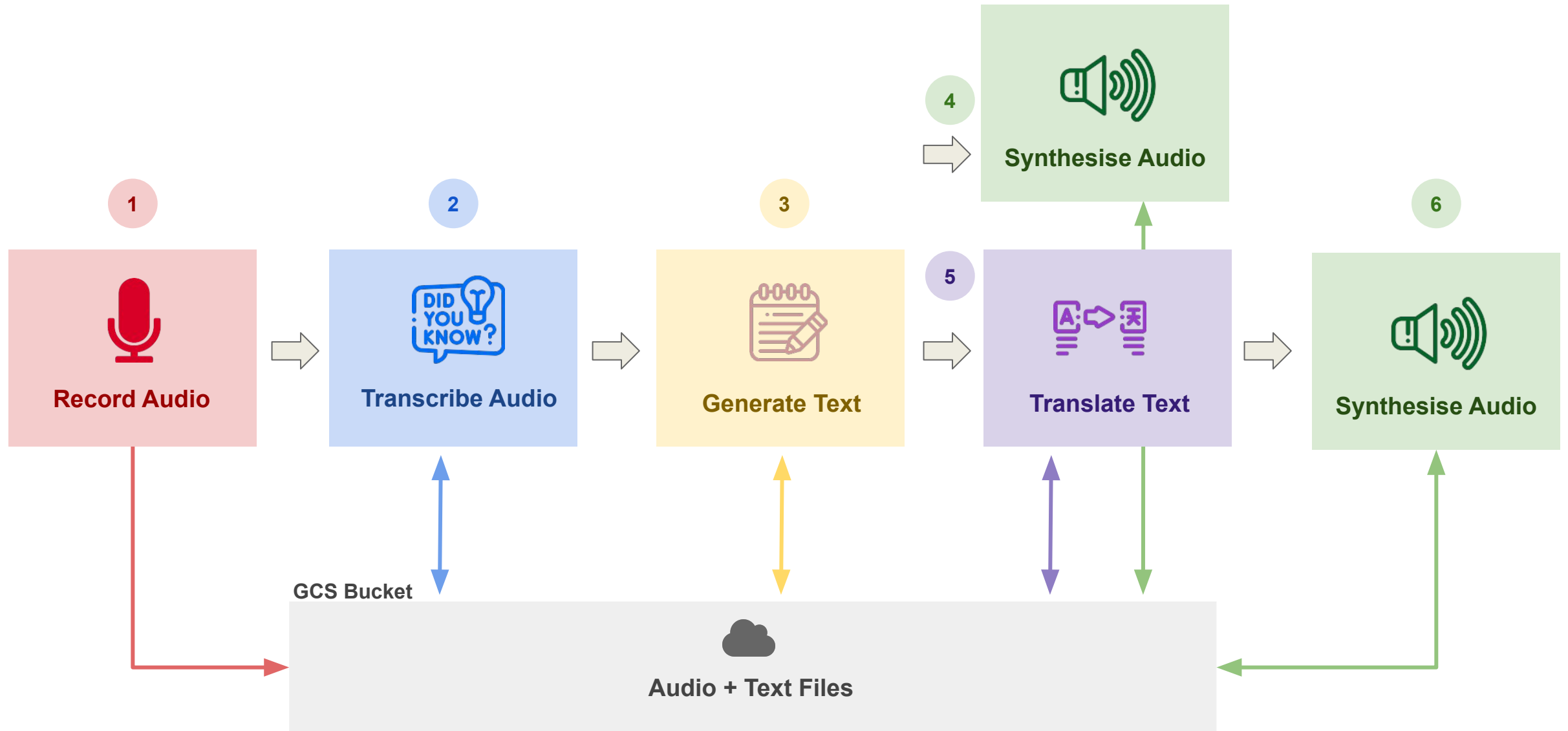
Outline

1. Recap: Review of Previous Material
2. Working with Containers Workflow
3. Data Pipelines
4. Data Labeling

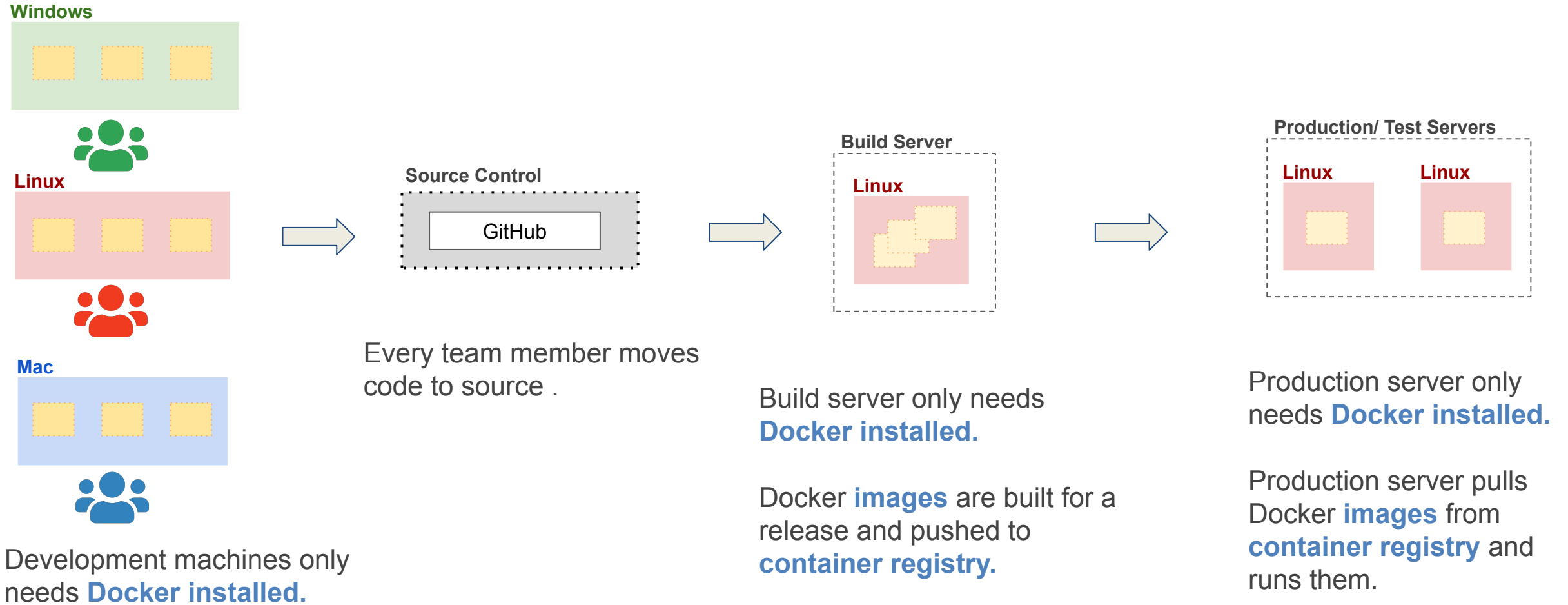
Outline

1. **Recap: Review of Previous Material**
2. Working with Containers Workflow
3. Data Pipelines
4. Data Labeling

Tutorial (T5) - Building the Mega Pipeline App



Software Development Workflow (with Docker)



Containers need to be setup only once.

Software Development Workflow (with Docker)

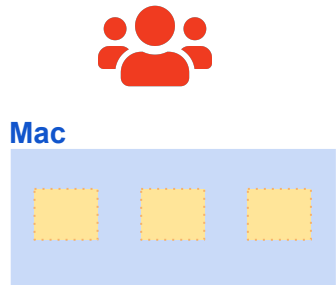
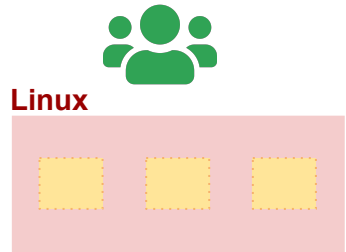
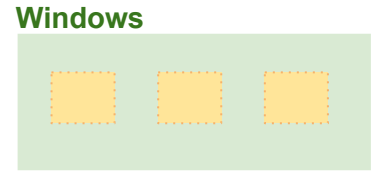
Who creates the Dockerfile, and where is it stored? Do we use pre-built images or does each developer build them? Who is in charge of managing this? Also, what's the process for handling the Pipfile and Pipfile.lock?

Development machines only needs **Docker installed**.

Containers need to be setup only once.

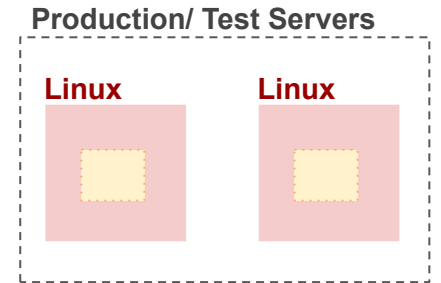
This seems like a lot.

Software Development Workflow (with Docker)



Development machines only needs **Docker installed**.

Containers need to be setup only once.



Production server only needs **Docker installed**.

Production server pulls Docker **images** from **container registry** and runs them.

Software Development Workflow (with Docker)

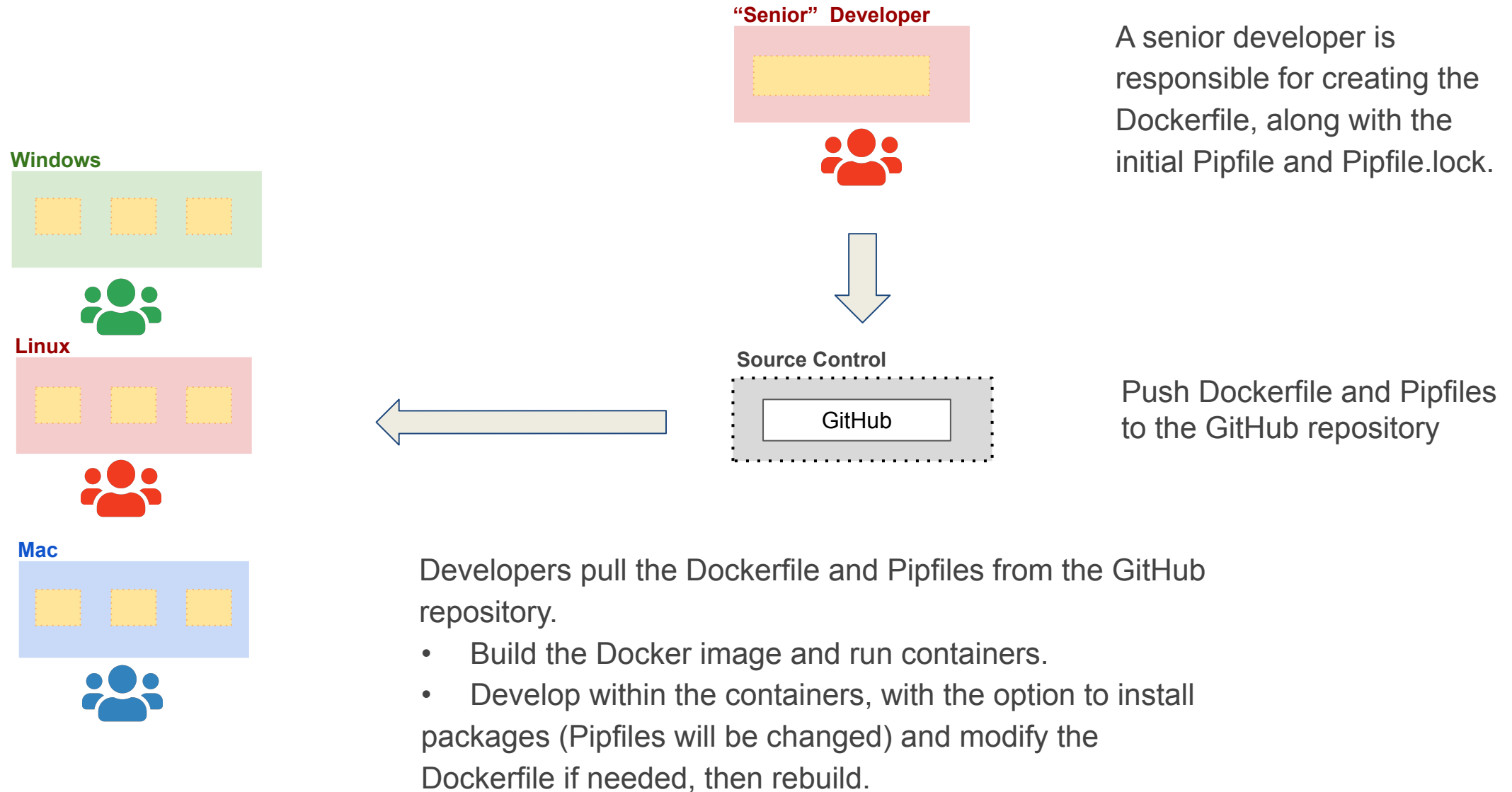
At this stage, creating multiple Dockerfiles along with their corresponding Pipfiles and secrets has become repetitive.

Is there any way to optimize the process?

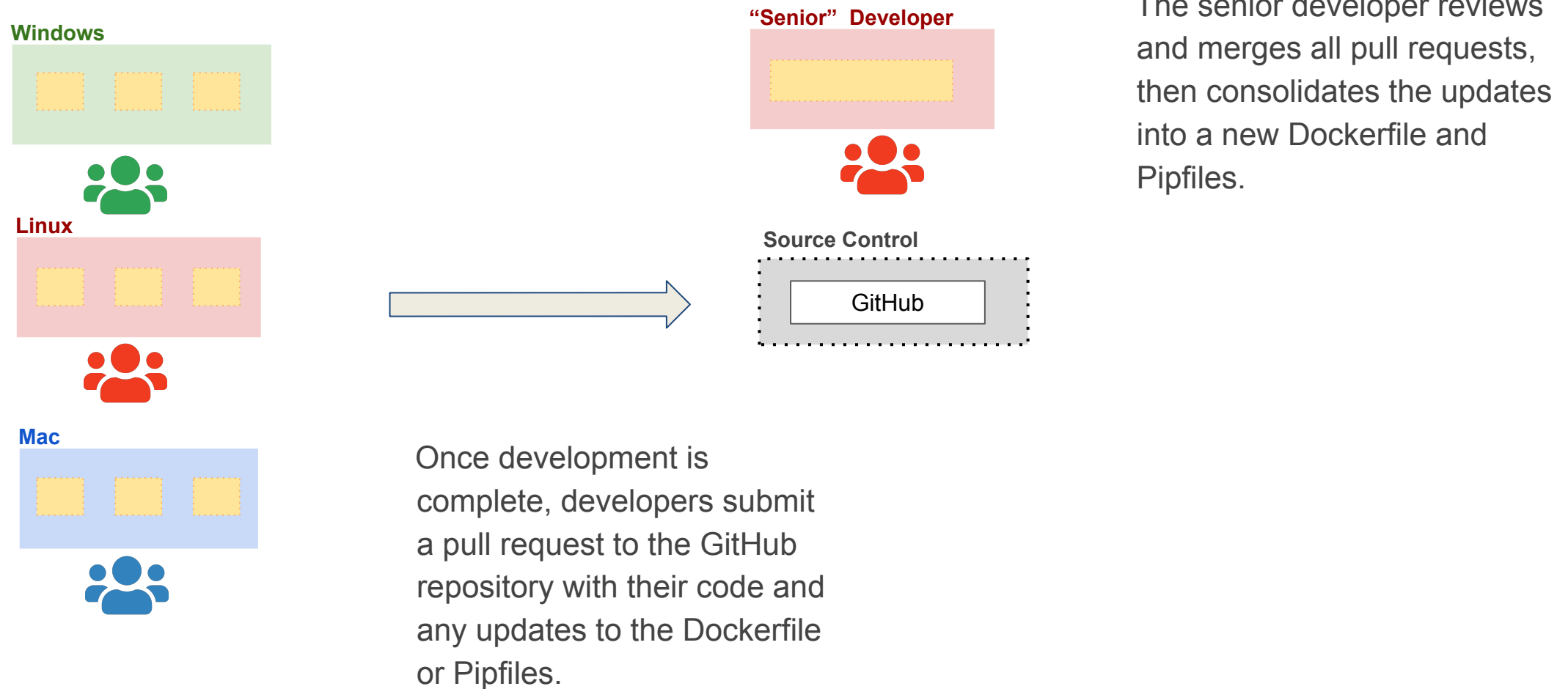
Outline

1. Recap: Review of Previous Material
2. **Working with Containers Workflow**
3. Data Pipelines
4. Data Labeling

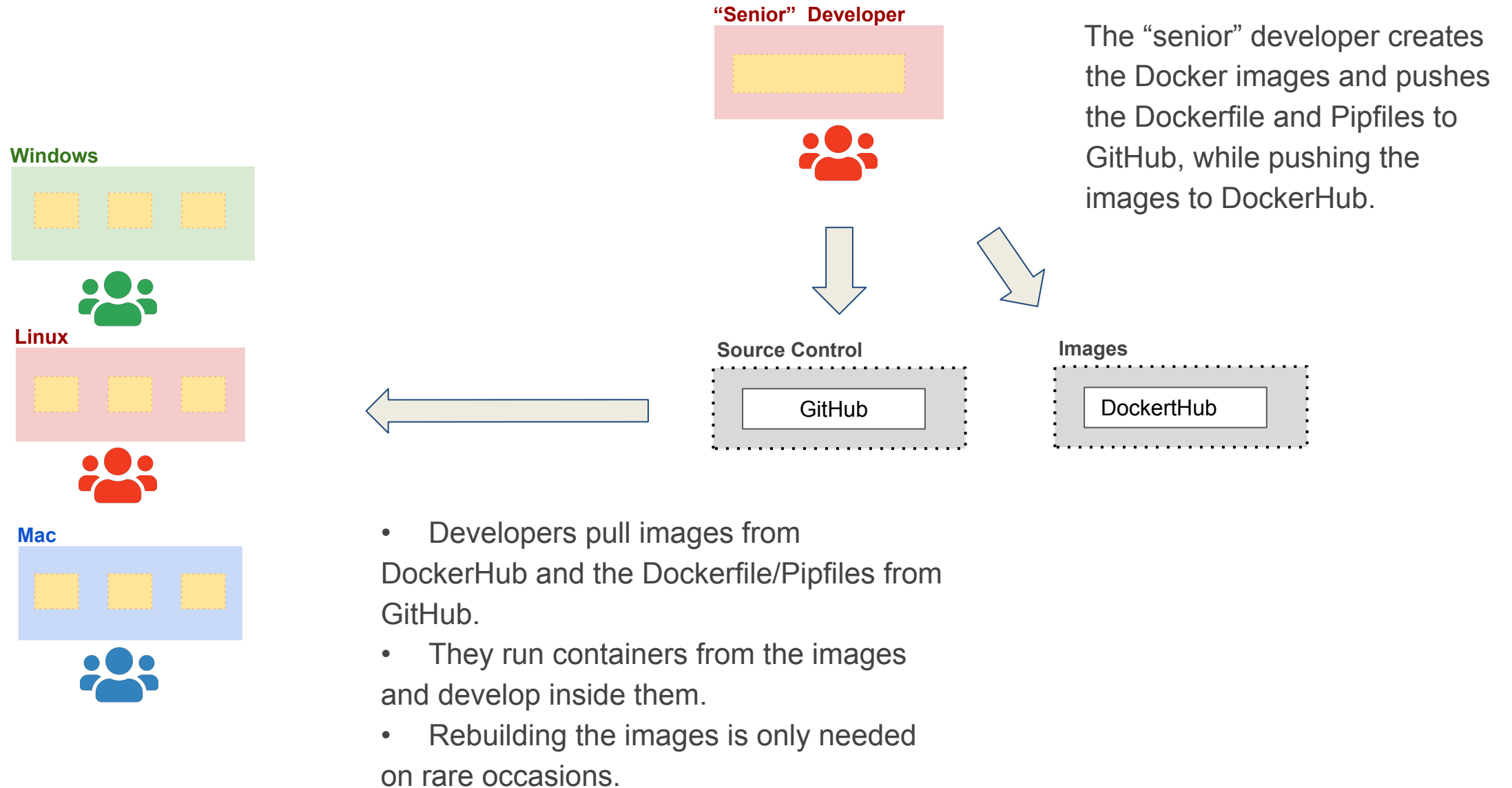
Workflow with Docker: **Scenario 1** (early stages of development)



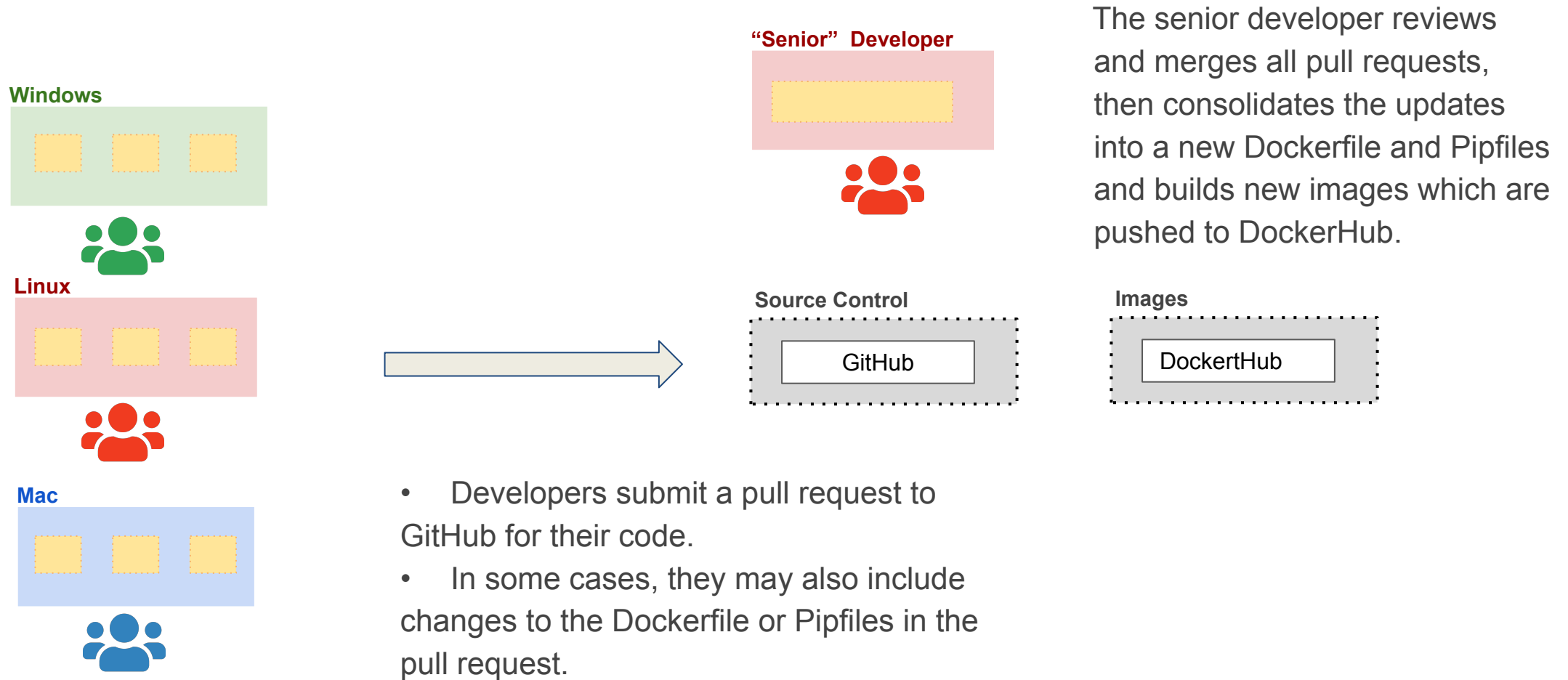
Workflow with Docker: **Scenario 1** (early stages of development)



Workflow with Docker: **Scenario 2** (later stages of development)



Workflow with Docker: **Scenario 2** (later stages of development)



Workflow with Docker: **A Flexible Approach**

Is there a “perfect” workflow?

No

So, how do we decide what to do?

Clear communication and rules are essential. Each team can have its own workflow, based on the project and team needs.



Tutorial (T5B) - Building the Mega Pipeline App with a structured workflow

In this tutorial we will build the [Mega Pipeline App](#) (again).

Unlike what we did in T5, this time we will follow a more structured workflow.

- The Dockerfiles and Pipfiles will be [provided](#); you won't need to create them.
- You can either [build](#) the images yourself or [run](#) them directly from [DockerHub](#).
- [Secrets](#) should be [stored](#) in a [folder outside](#) the app directories, which will not be part of the repository.
- A `docker-shell.sh` script is provided to handle all Docker-related tasks, including building, setting environments, and running containers.
- App: <https://ac215-mega-pipeline.dlops.io/>
- Instructions: <https://github.com/dlops-io/mega-pipeline/tree/flexible-workflow>



Outline

1. Recap: Review of Previous Material
2. Working with Containers Workflow
3. **Data Pipelines**
4. Data Labeling

Motivation

The 3 components for better Deep Learning



More Data



Better/Faster Models



Faster Hardware

Motivation

The 3 components for better Deep Learning



More Data

- Extraction
 - Transformation
 - Labeling
 - Versioning
 - Storage
-
- Processing
 - Input to Training



Better/Faster Models

- SOTA Models
- Transfer Learning
- Distillation
- Compression



Faster Hardware

- Scaling data processing
- GPU, TPU
- Multi GPU Server Training

Motivation

The 3 components for better Deep Learning



More Data

- **Extraction**
- **Transformation**
- **Labeling**
- **Versioning**
- **Storage**

- **Processing**
- **Input to Training**

The narrative of the data challenges

- When collecting cheese images or text, we might source them from web searches or user uploads, but the **quality and format** of these images or text can vary over time.
- Also images may not always be in the **correct format**, and we need to address issues with duplicates or poor-quality images.
- Additionally, cheese text needs to be **chunked** for Retrieval-Augmented Generation (RAG) applications and converted into a suitable format for fine-tuning large language models (LLMs).
- **Managing** this data involves labeling new images from both users and web searches, keeping track of different versions of cheese data, and ensuring that the images we acquire are of high quality.

Challenges

Extraction

- **Varied Sources/Formats:** Data comes in different shapes, sizes, and formats.
- **Timelines of Updates:** Data can change over time, affecting model performance.

Transformation

- **Labeling:** Manual annotation is often labor-intensive.
- **Versions:** Multiple versions can cause inconsistency.
- **Quality:** Poorly processed data can lead to poor models.

Management

- **Labeling:** Consistency and quality are paramount.
- **Versions:** Ensuring data traceability and reproducibility.
- **Quality:** Ensuring the data is clean, relevant, and well-documented.

Containerize Data Tasks

- **Benefits:** Consistent environment, easy to scale, and improves reproducibility.

Using Prebuilt Containers for Data Tasks

- **Benefits:** Saves time, ensures quality, and utilizes community-verified methods.

Manage Tasks Using Pipeline Management Tools

- **Examples:** Apache Airflow, Kubeflow Pipelines.
- **Benefits:** Streamlines data workflows, manages dependencies, and allows for easy monitoring.

Pipeline Management

- **Kubeflow** End-to-end orchestration of machine learning pipelines

Data Labeling [TODAY]

- **Label Studio**
 - Annotation of text, images, audio, and more.
 - Customizable templates, multi-format support.
 - Teams needing flexibility in data labeling tasks.

Data Versioning [NEXT LECTURE]

- **DVC (Data Version Control)**
 - Version control for datasets and machine learning models.
 - Git-like commands, storage optimization.
 - Teams that want to maintain version history of data and models.

Components (**artifacts**) of an AI Application

What are the components of an AI App?

- **Data:** The backbone of any AI application, needed for training and validation.
- **Model:** The trained AI algorithm
- **Source Code:** Includes the model implementation, front end, back end, and Dockerfiles
- **Container Images:** Encapsulated environments that ensure the application runs consistently across different systems.

How do we manage all of these?

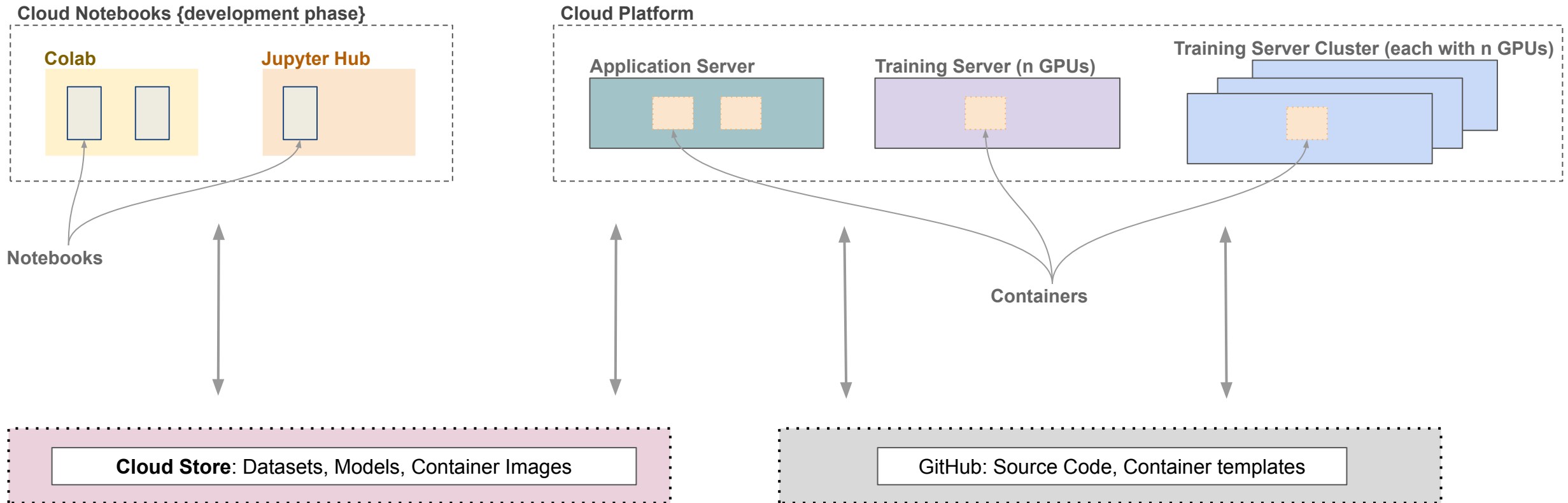
Components (**artifacts**) of an AI Application

Management Strategies

- **Data:** Implement storage, versioning, and backup.
- **Model:** Track versions and performance; update as needed.
- **Source Code:** Use version control and maintain documentation.
- **Containers:** Use orchestration tools for deployment and scaling.

What are Pipelines

Example components of an AI App:



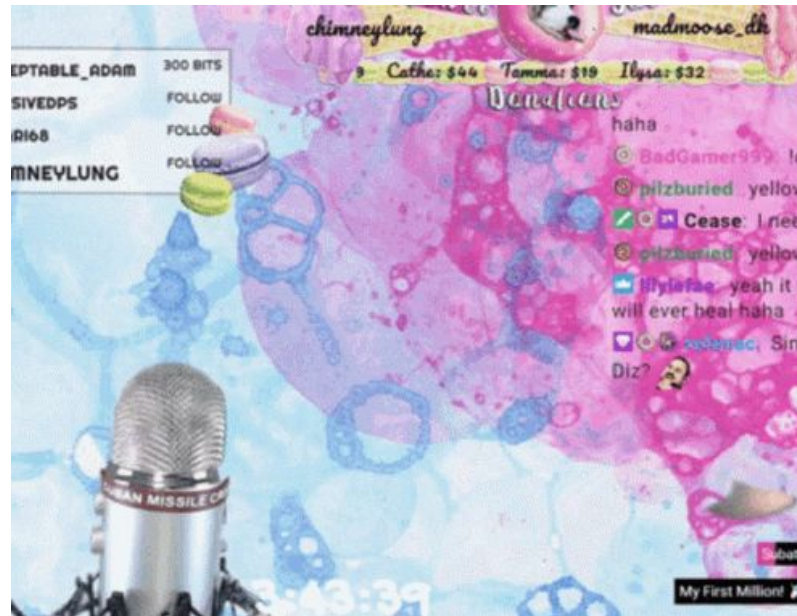
Streamline with Tools

- **Manual Methods:** Scripts and manual tasks can work, but they're often slow and prone to mistakes.
- **Automated Tools:** For better efficiency and fewer errors, use tools that simplify and automate these processes.



Streamline with Tools

- **Manual Methods:** Scripts and manual tasks can work, but they're often slow and prone to mistakes.
- **Automated Tools:** For better efficiency and fewer errors, use tools that simplify and automate these processes.



Wish List

We want a system with these features:

- Version control code, data, and models
- Easy access of data and models from external tools
- Automate data and model tasks

Pipelines

And a few more things like:

- Real-Time Monitoring of Models (data monitoring, future lecture)
- Auto-Scaling Resources (kubernetes, future lecture)
- Automated Testing Frameworks (Continuous Integration, future lecture)
- Easy Rollback and Rollforward Mechanisms (Continuous Deployment, future lecture)
- Built-in Security Measures (AIP, GCP secrets)

What are Data Pipelines

Various data tasks in a Machine/Deep Learning project:

- Extraction
- Transformation
- Pre-processing
- Train, validate, test split
- Pre-process step during model inference

Outline

1. Recap: Review of Previous Material
2. Working with Containers Workflow
3. Data Pipelines
4. **Data Labeling**

Tutorial (T6)

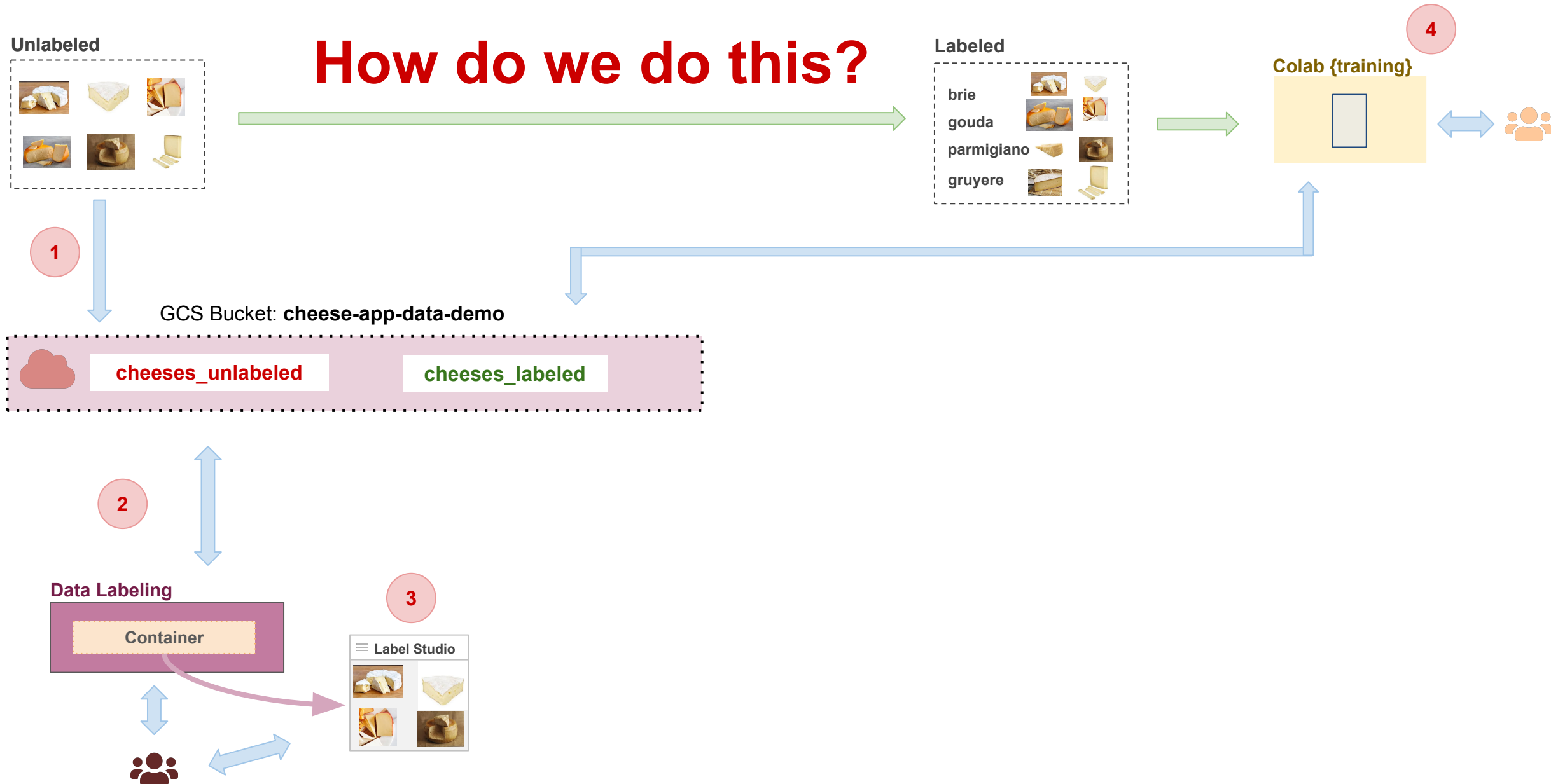
In this tutorial, we will learn how to perform labeling.

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)

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

- Docker Network
- Cloud Storage and Credentials
- Docker Compose

Tutorial (T6)

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 **inspect** **data-versioning-network** **> /dev/null**

display info about network

Name of the network

Redirects standard output
to this path that discards
the data sent to it.

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

Suppress the standard
error too

Run next command only if
previous command fails

Docker Network

In the following command:

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

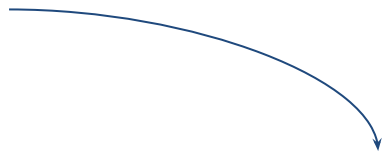
The final part creates the network if it does not exist.

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)

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.

Docker Compose

For this tutorial we will use [Docker Compose](#).

- A docker compose file is for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your containers.
- Then, with a single command, you **build** and **start** all the containers.

[PP: ADD HOW TO RUN DOCKER COMPOSE]

Run `docker-shell-CLI.sh` to enter a container where you can execute `cli.py`

Tutorial (T6)

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

```
version: "3.8"
```

```
# Define network that the various docker containers will share
```

```
networks:
```

```
  default:
```

```
    name: data-labeling-network
```

```
    external: true
```

List of containers to run

```
services:
```

```
  data-label-cli:
```

```
    image: data-label-cli
```

```
    container_name: data-label-cli
```

```
    volumes:
```

```
      - ../secrets:/secrets
```

```
      - ../data-labeling:/app
```

Volumes to mount to the container

```
    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"
```

Environment variables to set
inside container

```
    depends_on:
```

```
      - data-label-studio
```

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

```
...
```


docker-compose.yml continued

```
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"
```

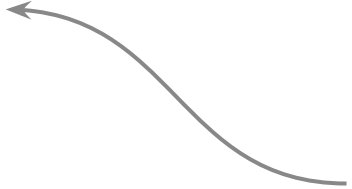
Port to expose from inside
container to the host outside



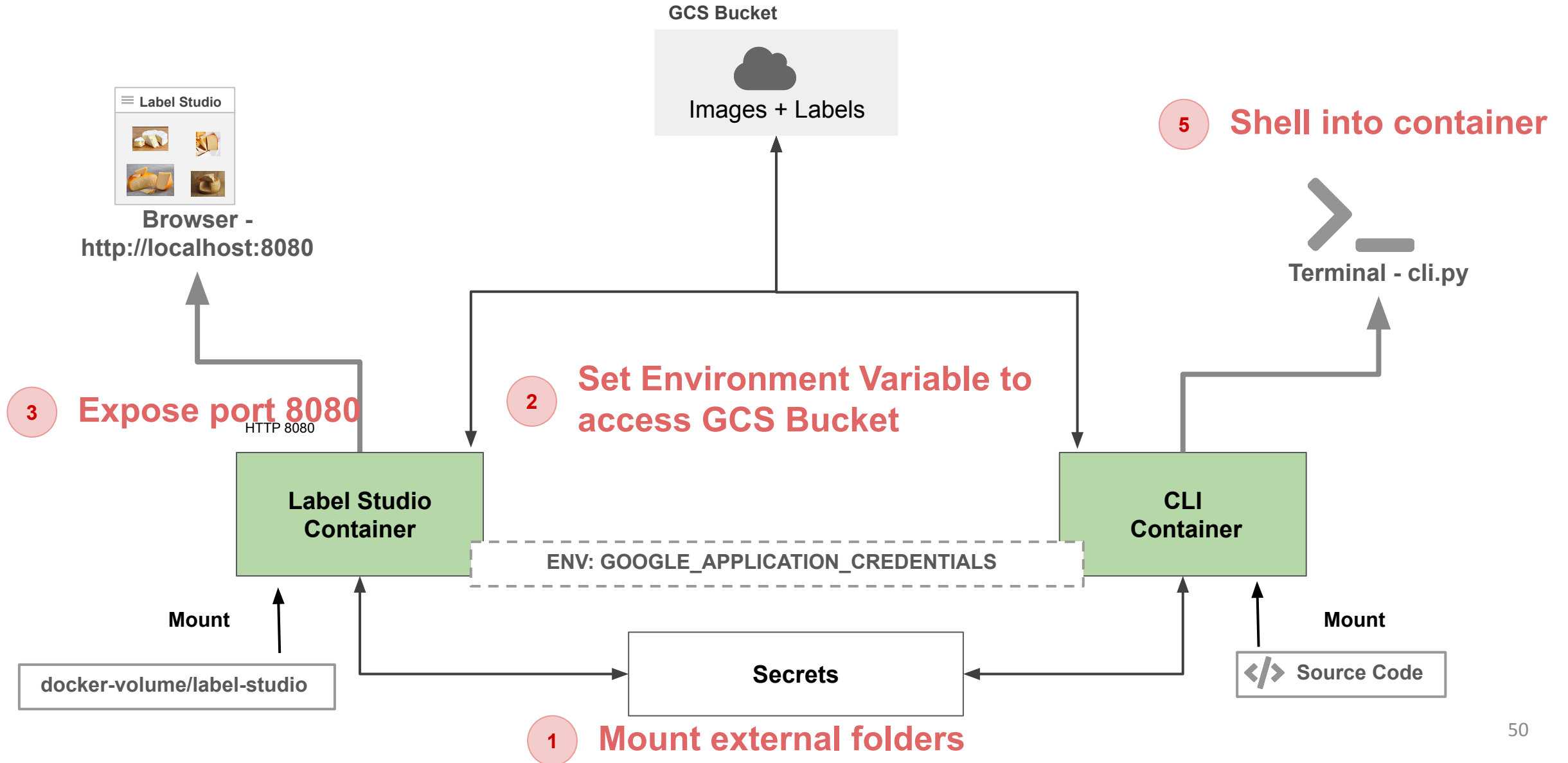
Volumes to mount to the container



Environment variables to set
inside container



Tutorial (T6): Label Studio + CLI



Tutorial (T6): Cheese App Data Pipeline

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

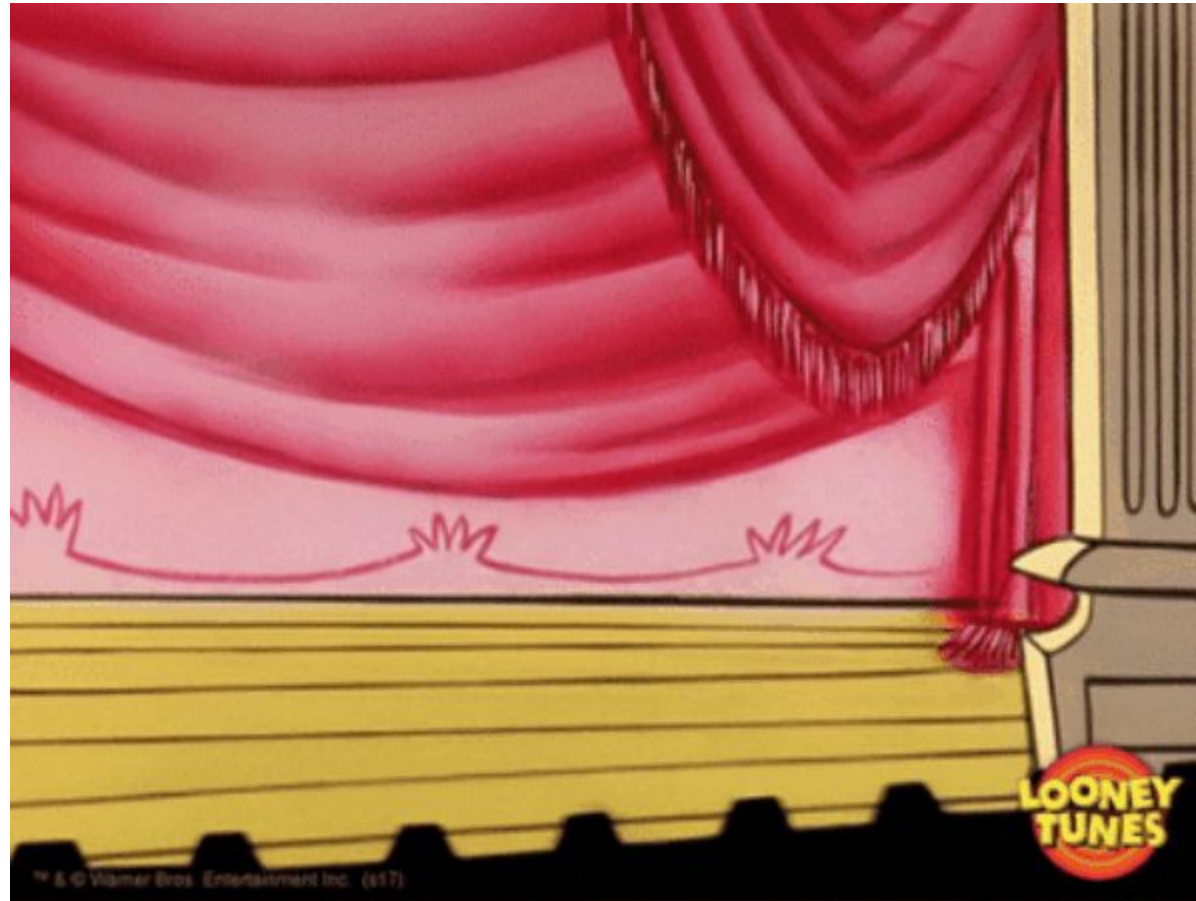
- Create a GCS bucket to store all data.
- Run Data Labeling Container.
- For detailed instructions, please refer to the following link
 - [Data Labeling](https://github.com/dlops-io/data-labeling). (<https://github.com/dlops-io/data-labeling>)



Logistics/Reminders

- Approx. 70% of class has project partners - if you have formed group make sure to update this [group info spreadsheet](#)
- We highly encourage you to find project partners based on your mutual interests or goals (rather than us assigning later on)
- Even if you don't have partners , you must submit Milestone 1 - due 09/20.
- Class video recordings are available on Canvas -> Panopto





THANK YOU