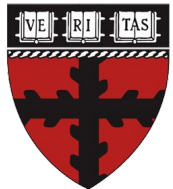


# Lecture 5: Docker Workflows

AC215

Pavlos Protopapas  
SEAS/Harvard



## 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 fill the form so we know you are active.
- Class video recordings are available on Canvas -> Panopto

# Outline

---

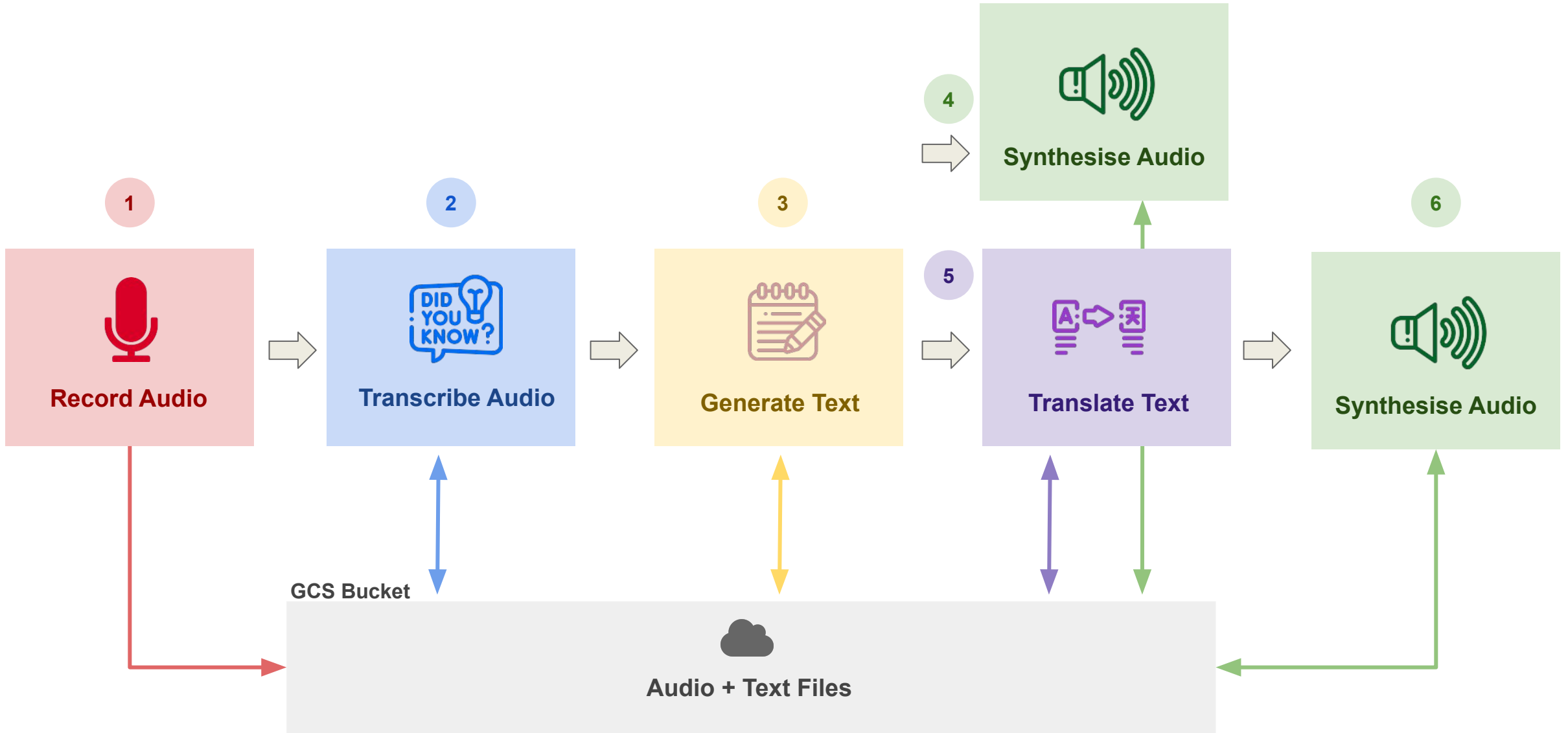
1. Recap: Review of Previous Material
2. Working with Containers Workflow
3. Data Pipelines
4. Data Labeling

# Outline

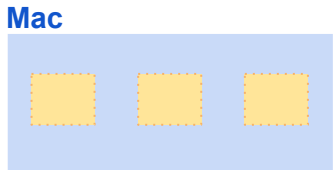
---

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

# Tutorial (T5) - Building the Mega Pipeline App

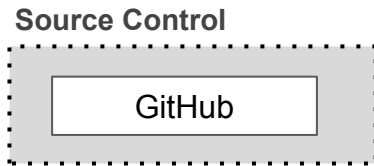


# Software Development Workflow (with Docker)

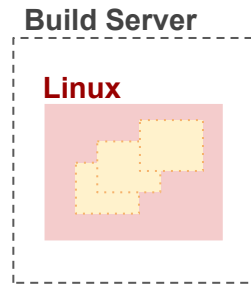


Development machines only needs **Docker installed**.

**Containers** need to be setup only once.

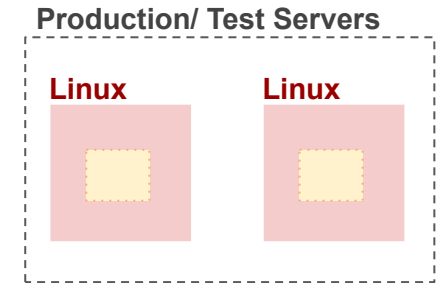


Every team member moves code to source .



Build server only needs **Docker installed**.

Docker **images** are built for a release and pushed to **container registry**.



Production server only needs **Docker installed**.

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

# Software Development Workflow (with Docker)

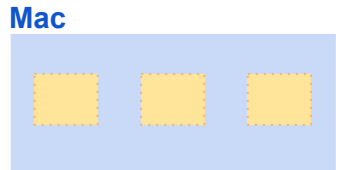
Who creates the Dockerfile, and where is it stored? Do we use pre-built images or does each developer build them? Who is in charge of managing this? Also, what's the process for handling the Pipfile and Pipfile.lock?

Development machines only needs **Docker installed**.

**Containers** need to be setup only once.

This seems like a lot.

# Software Development Workflow (with Docker)

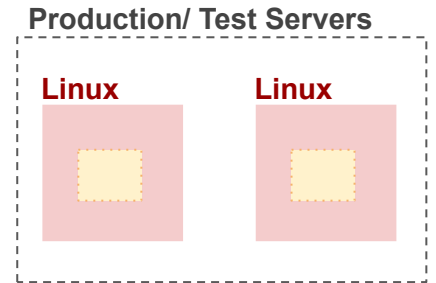
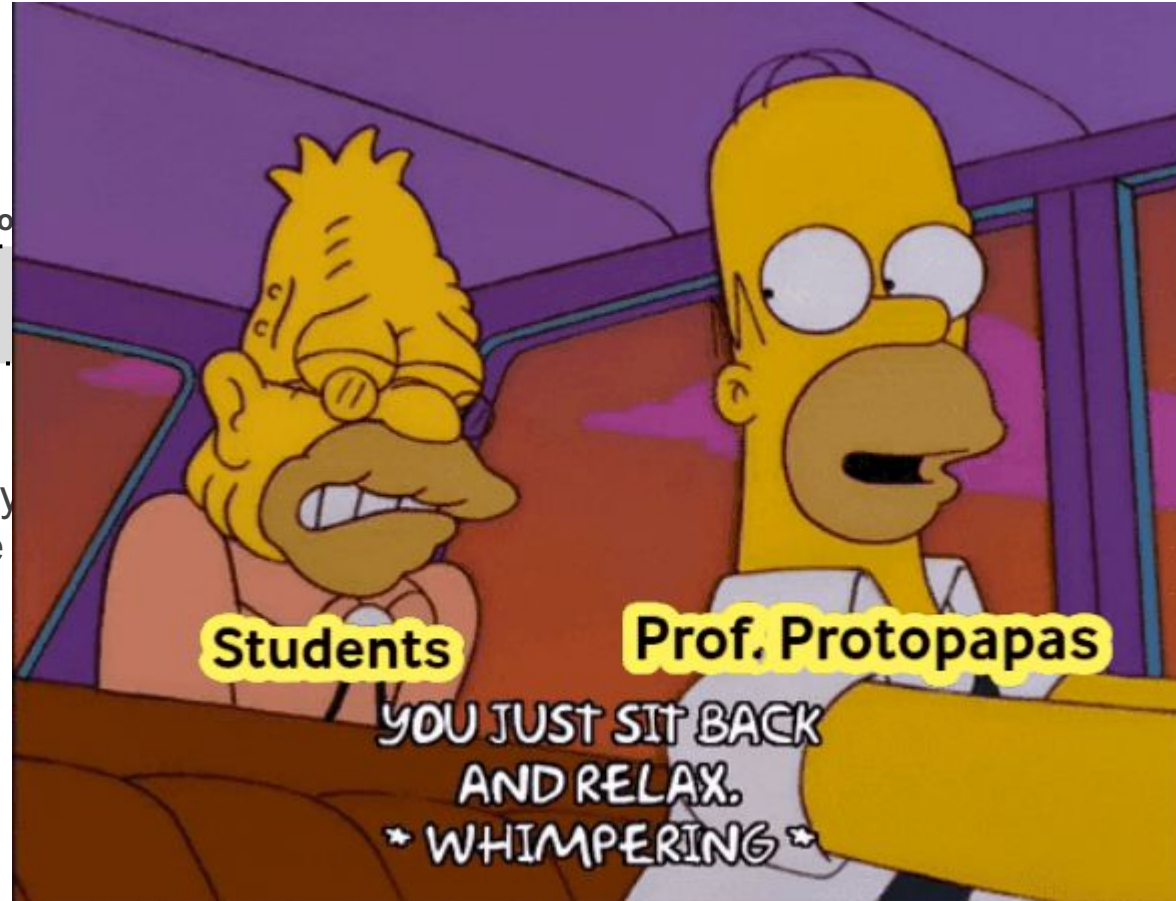


Development machines only needs **Docker installed**.

**Containers** need to be setup only once.



Software  
Every code



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.

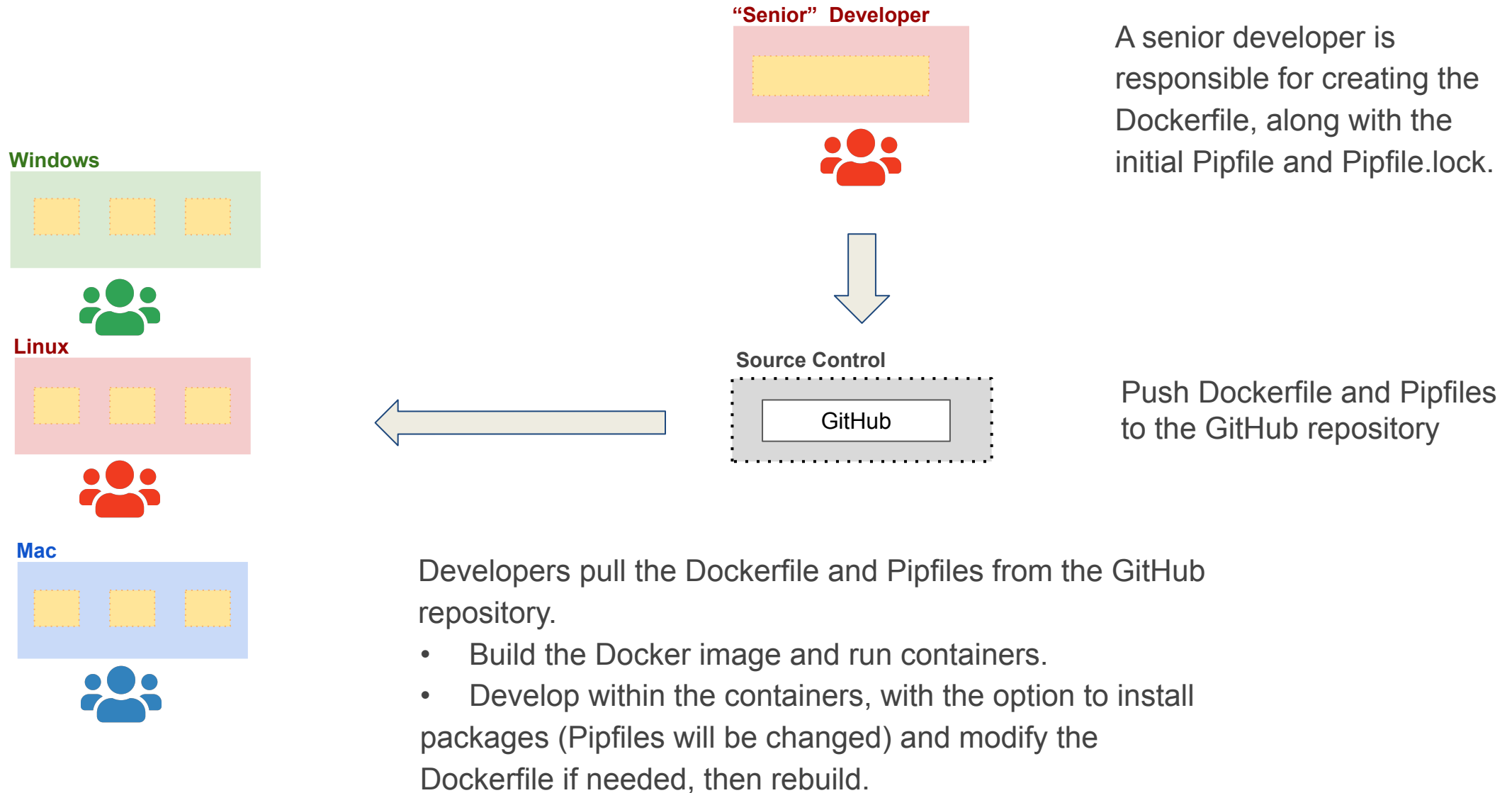
Is there any way to optimize the process?

# Outline

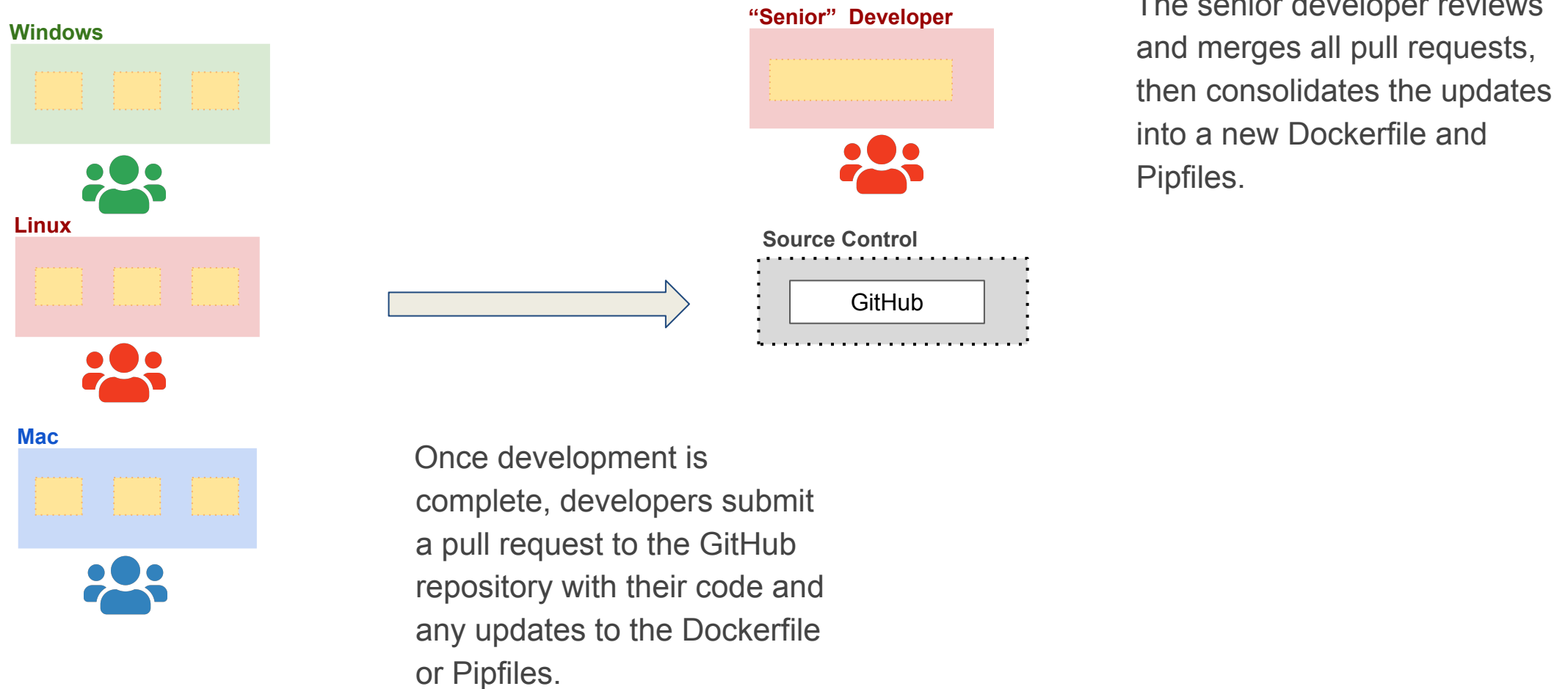
---

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

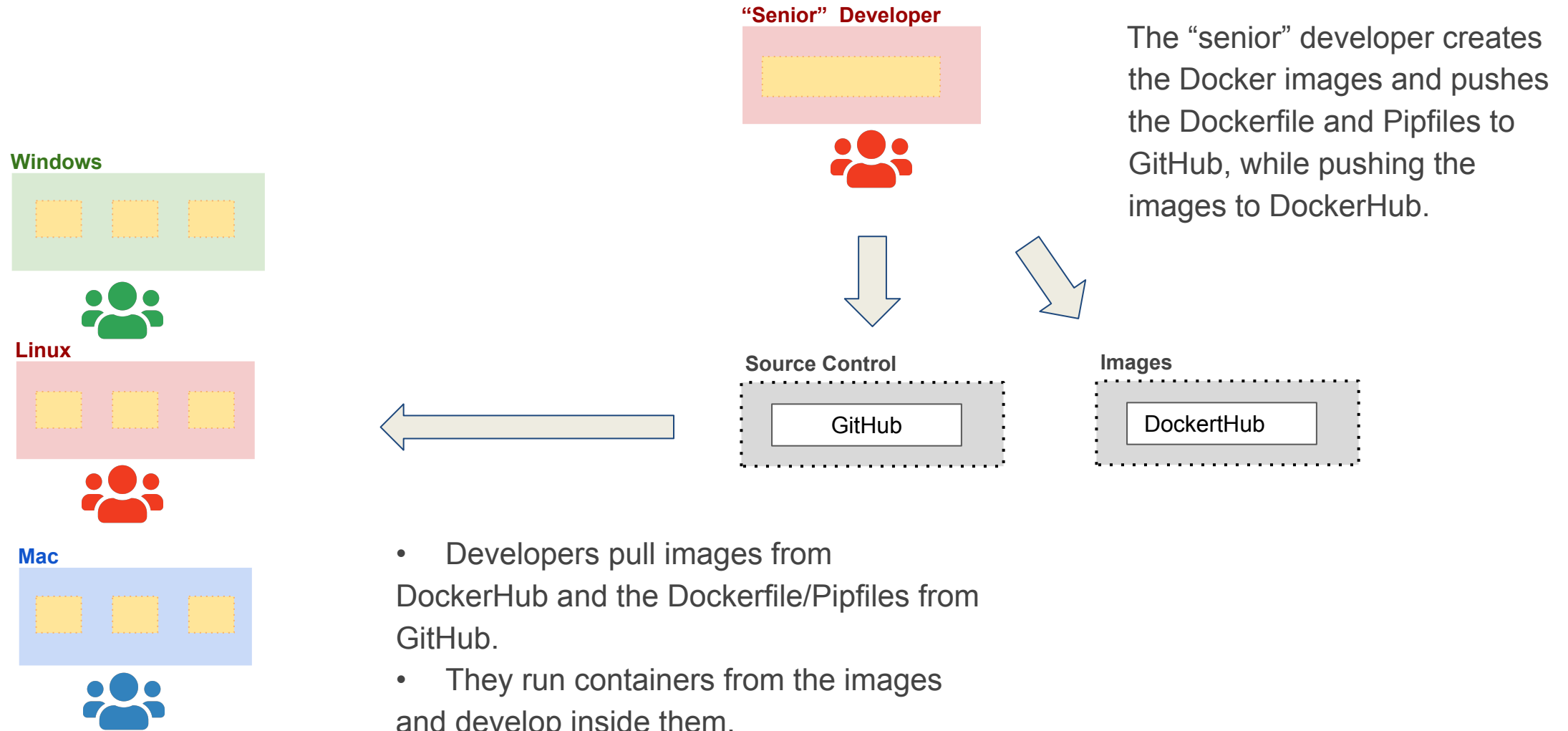
# Workflow with Docker: Scenario 1 (early stages of development)



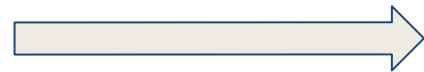
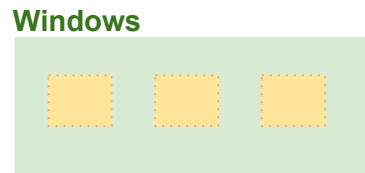
# Workflow with Docker: Scenario 1 (early stages of development)



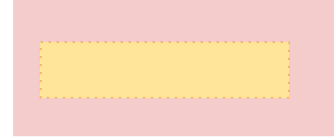
# Workflow with Docker: Scenario 2 (later stages of development)



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



**“Senior” Developer**



**Source Control**



The senior developer reviews and merges all pull requests, then consolidates the updates into a new Dockerfile and Pipfiles and builds new images which are pushed to DockerHub.

**Images**



- Developers submit a pull request to GitHub for their code.
- In some cases, they may also include changes to the Dockerfile or Pipfiles in the pull request.

# 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

- 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 [FUTURE LECTURE]

- **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

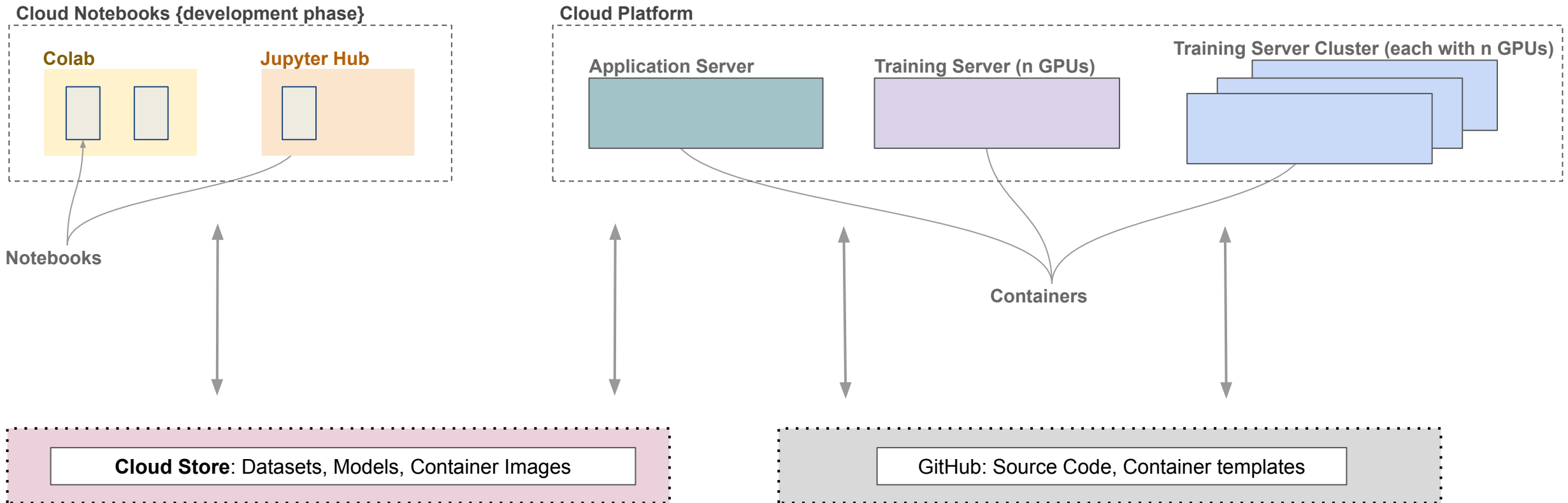
---

## 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.

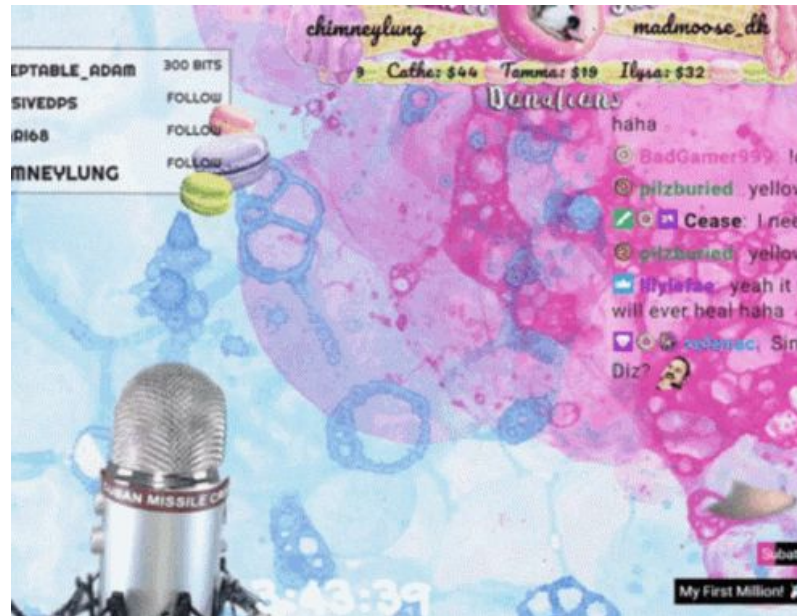
# What are Pipelines

## Example components of an AI App:

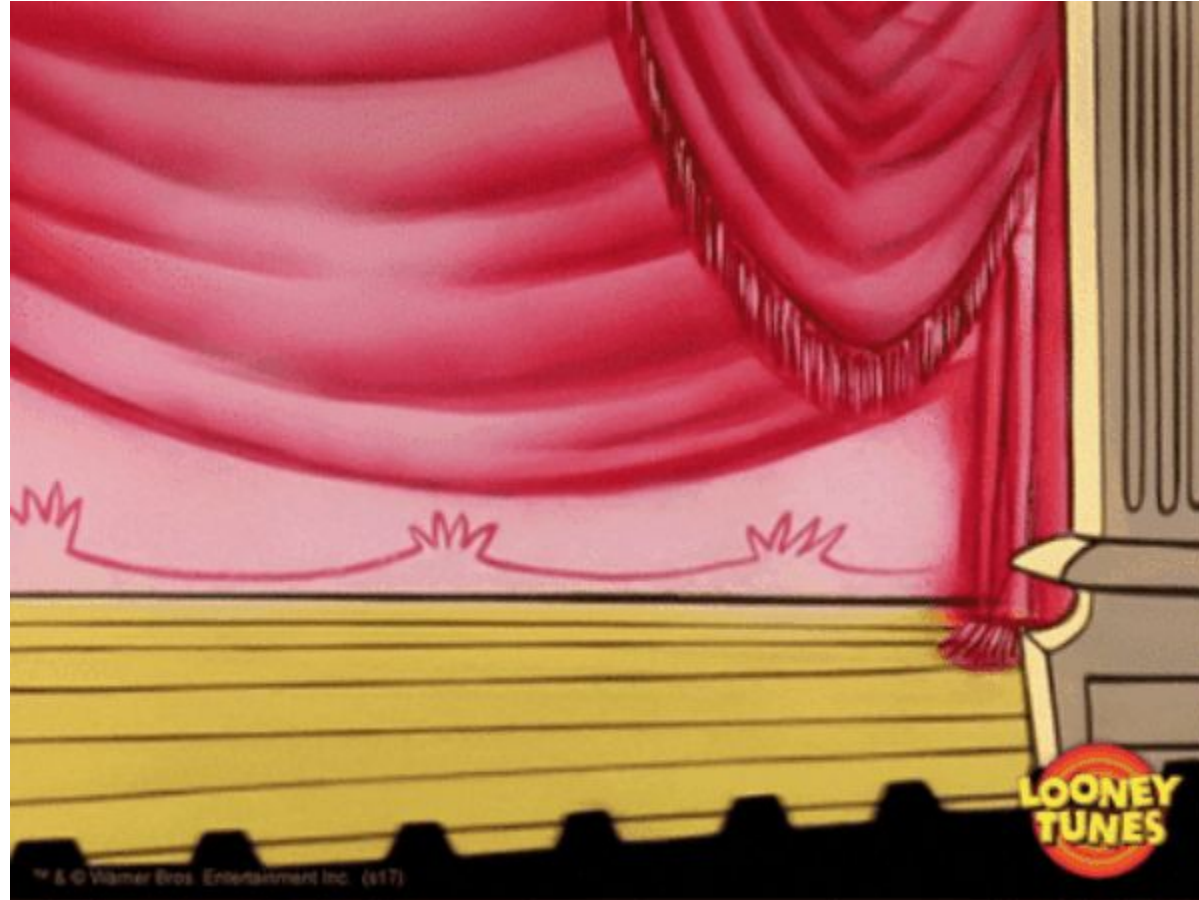


# Streamline with Tools

- **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.







**THANK YOU**