01_setup

June 21, 2025

1 Production 1: Setting Up A Repo

1.1 Introduction

- Working with code in production is hard. Rarely we will have a chance to work on a greenfield development and will get a chance to define all of its specifications.
- Sometimes, we may be offered the option of scraping a system and starting from scratch. This option should be considered carefully and, most of the time, rejected.
- Working with legacy code will be the norm:
  - Legacy code includes our own code.
  - Legacy code may have been written by colleagues with different approaches, philosophies, and skills.
  - Legacy code may have been written for old technology.
- Most of the time, legacy code works and *this* is the reason we are working with it.

1.2 Software Entropy

- Software entroy is the natural evolution of code towards chaos.
- Messy code is a natural consequence of change:
  - Requirements change.
  - Technology change.
  - Business processes change.
  - People change.
- Software entropy can be managed. Some techniques include:
  - Apply a code style.
  - Reduce inconsistency.
  - Continuous refactoring.
  - Apply reasonable architectures.
  - Apply design patterns.
  - Testing and CI/CD.
  - Documentation.
- *Technical debt* is future work that is owed to fix issues with the current codebase.
- Technical debt has principal and interest: complexity spreads and what was a simple *duct tape* solution becomes the source of complexity in downstream consumers.

- ML systems are complex: they involve many components and the interaction among those components determines the behaviour of the system. Adding additional complexity by using poor software development practices can be avoided.
- Building ML Systems is most of the time a team sport. Our tools should be designed for collaboration.

2 A Reference Architecture

2.1 What are we building?

- Agrawal and others propose the reference architecture below.

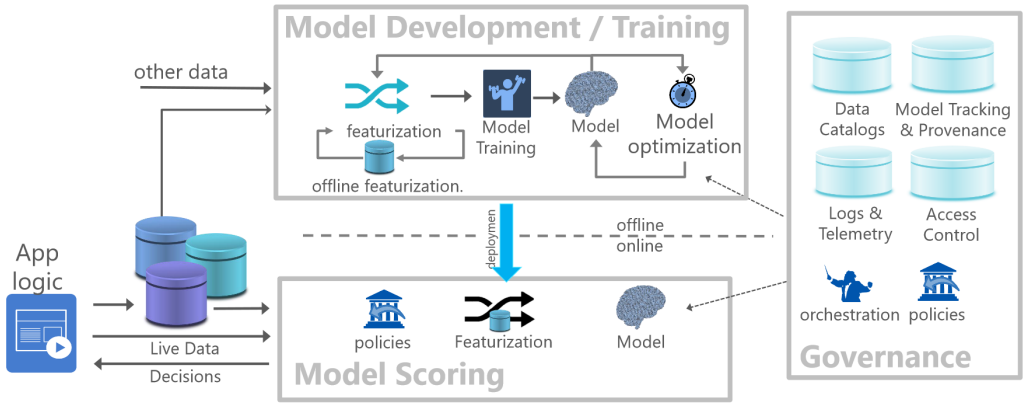


Figure 1: Flock reference architecture for a canonical data science lifecycle.

- Through the course, we will write the code in Python for the different components of this architecture.

3 Source Control

3.1 Git and Github

- Git is a version control system that lets you manage and keep track of your source code history.
- If you have not done so, please get an account on Github and setup SSH authentication:
  - Check for existing SSH keys.
  - If needed, create an SSH Key.
  - Add SSH key to your Github account.
- If you need a refresher of Git commands, a good reference is Pro Git (Chacon and Straub, 2014).

3.2 What do we include in a commit?

- Generally, we will use Git to maintain data transformation and movement *code*.

- It is good practice to not use Git to maintain data inputs or outputs of any kind.
- Some exceptions include: settings, experimental notebooks used to document design choices.
- Things to avoid putting in a repo: Personal Identity Information (PII), passwords and keys.

3.3 Version Control System Best Practices

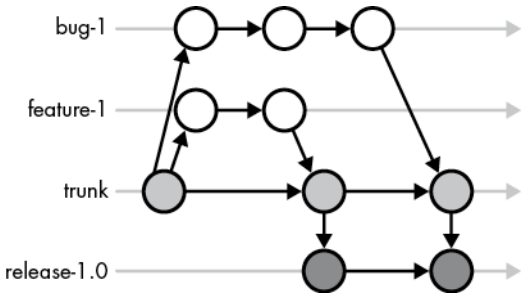
- Commit early and commit often.
- Use meaningful commits:
  - The drawback of commiting very frequently is that there will be incomplete commits, errors and stepbacks in the commit messages. Commit messages include: "Committing before switching to another task", "Oops", "Undoing previous idea", "Fire alarm", etc.
  - In Pull Requests, squash commits and write meaningful messages.
- Apply a branch strategy.
- Submit clean pull requests: verify that latest branch is merged and review conflicts.

3.4 Commit Messages

- Clear commit messages help document your code and allow you to trace the reaoning behind design decisions.
- A few guidelines for writing commit messages:
  - Use markdown: Github interprets commit messages as markdown.
  - First line is a subject:
    - No period at the end.
    - Use uppercase as appropriate.
  - Write in imperative form in the subject line and whenever possible:
    - Do: "Add connection to db", "Connect to db"
    - Do not: "This commit adds a connection to db", "Connection to db added"
  - The body of the message should explain why the change was made and not what was changed.
    - Diff will show changes in the code, but not the reasoning behind it.
  - Same rules apply for Pull Requests.
- Many of these points are taken from How to Write a Git Commit Message by Chris Beams.

3.5 Branching Strategies

- When working standalone or in a team, you should consider your branching strategy.
- A branching strategy is a way to organize the progression of code in your repo.
- In trunk-based branching strategy, each developer works based on the *trunk* or *main* branch.



- After each bug fix, enhancement, or upgrade is complete, the change is integrated to *main*.
- Generally, part of a larger Continuous Integration/Continuous Deployment (CI/CD) process.

3.6 VS Code and Git

- An Interactive Development Environment (IDE) is software to help you code.
- IDEs are, ultimately, a matter of personal taste, but there are advantages to using the popular solutions:
  - Active development and bug fixes.
  - Plugin and extension ecosystems.
  - Active community for help, support, tutorials, etc.
- Avoid the *l33t coder* trap: *vim* and *emacs* may work for some, but *nano* and VS Code are great solutions too.
- Reference: Using Git source control in VS Code.
- A few tips:
  - From the source control menu, one can easily stage files, commit, and push/pull to origin.
  - Other commands can be accessed via the command pallete (`Ctrl + Shift + P`). For instance, one can select or create a new branch using the option *Git: Checkout to*.

4 Python Virtual Environments

- There are many reasons to control our development environment, including version numbers for Python and all the libraries that we are using:
  - Reproducibility: we want to be able to reproduce our process in a production environment with as little change as possible.
  - Backup and archiving: saving our work in a way that can be used in the future, despite Python and libraries evolving.
  - Collaboration: work with colleagues on different portions of the code involves everyone having a standard platform to run the codebase.
- We can achieve the objectives above in many ways, including vritualizing our environments, packaging our code in containers, and using virtual machines, among others.
- Most of the time, creating a virtual environment will be part of the initial development setup. This vritual environment will help us *freeze* the python version and some version libraries.

4.1 Setting up the environment

4.1.1 Using venv

- The simplest way to add a new virtual environment is to use the command: `python -m venv env`.
- This command will start a new virtual environment in the subfolder `./env`.
- To *activate* this environment use `./env/Scripts/Activate.ps` (windows).
- Optionally, consider the Python add-on for VS Code that activates the environment automatically for you.

4.1.2 Conda

- Conda is a command line tool for package and environment management.
- From the terminal, create a virtual environment with: `conda create -n <env-name>`. For example, `conda create -n scale2prod` creates a new environment called `scale2prod`.
- Activate the environment with `conda activate <env-name>`. For example, `conda activate scale2prod`.
- Other useful commands are:
  - Verify conda installation: `conda info` or `conda -V`
  - List current environments: `conda info --envs`

5 Setup a Logger

- We will use Python's logging module and will provision our standard loggers through our first module.
- The module is located in `./05_src/utils/logger.py`.
- Our notebooks will need to add `../05_src/` to their path and load environment variables from `../05_src/.env`. Notice that these paths are based on the notebook's location.

5.0.1 Logger highlights

A few highlights about `./05_src/utils/logger.py`:

- This logger has two handlers:
  - A `FileHandler` that will save logs to files that are datetime index.
  - A `StreamHandler` handler that outputs messages to the stdout.
- Each logger can set its own format.
- The log directory and log level are obtained from the environment.
- According to the Advanced Logging Tutorial:

  > "A good convention to use when naming loggers is to use a module-level logger, in each module which uses logging, named as follows:
  >
  > `logger = logging.get_logger(__name__)`.

> This means that logger names track the package/module hierarchy, and it's intuitively obvious where events are logged just from the logger name."

Run the code below to verify that your setup is working.

[ ]:
```
%load_ext dotenv
%dotenv
```

[ ]:
```
import sys
sys.path.append("../../05_src")
```

[ ]:
```
from utils.logger import get_logger
_logs = get_logger(__name__)
_logs.info("Hello world!")
```

6 Using Docker to Set Up Experiment Tracking

- For our work, we need an environment that resembles the production environment as closely as possible.
- One way to achieve this is to use containers and containerized application.
- Without going into the details, you can think of a container as software that encapsulates the key features of an operating system, a programming language, and the application code.
- Containers are meant to be portable across operating systems: a container will work the same regardless if the underlying Docker application is installed in a Windows, Linux or Mac machine.
- Containers are not Virtual Machines.
- Docker is a popular application that implement containers.

6.1 What is Docker?

- From product documentation:

  Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

6.2 General Procedure

- To setup services using containers, we will do the following:

1. Download an image from Docker Hub or equivalent image repository.
2. If required, set up a volume to persist data.
3. Redirect ports as needed.
4. Start the container.

In our course, we will setup the following services:

- MLFlow: an experiment tracking system. MLFlow requires two backends: a database and an object store.
- PostgreSQL: a database management system.
- MinIO: an object store that resembles S3 buckets in AWS.

6.3 Starting the Containers

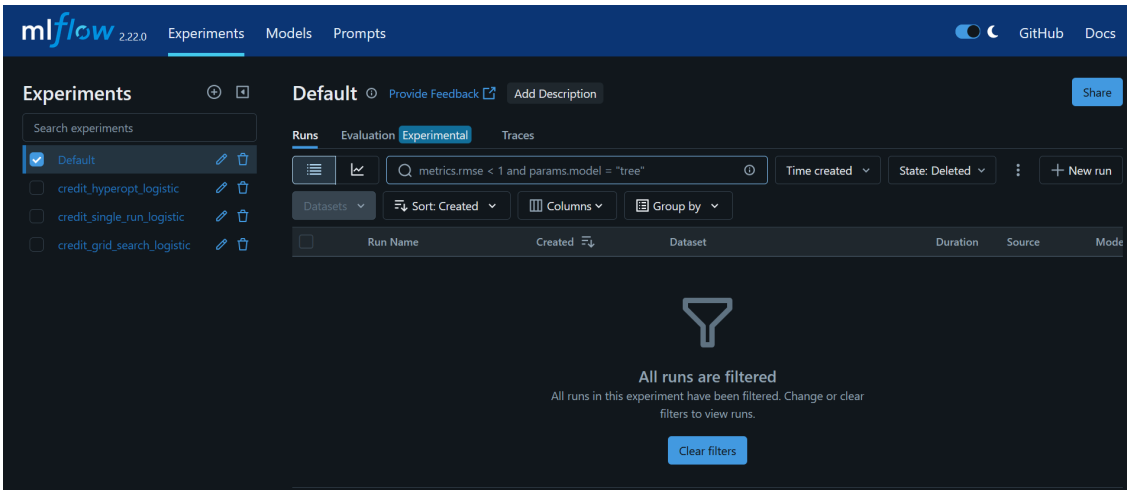
- To run the process above, first navigate to the `./05_src/experiment_tracking/` folder.
- The first time that you set up the containers, you will need to build the MLFlow image. You can build the required image with `docker compose build`.
- After building a local image for MLFlow, run `docker compose up -d`.
- The flag `-d` indicates that we will do a headless run.
- Notice that the containers are set to always restart. You can remove the option or turn the containers off manually. Be aware that if you leave this option on, the containers will run any time your Docker desktop restarts.

6.4 Stopping the Containers

- To stop the containers use (from `./05_src/db/`): `docker compose stop`.
- Alternatively, you can bring all images down including their volumes with: `docker compose down -v`.
  - The `-v` flag removes volumes.
  - It is the best option when you are do not need the data any more because **it will delete the data in your DB** .

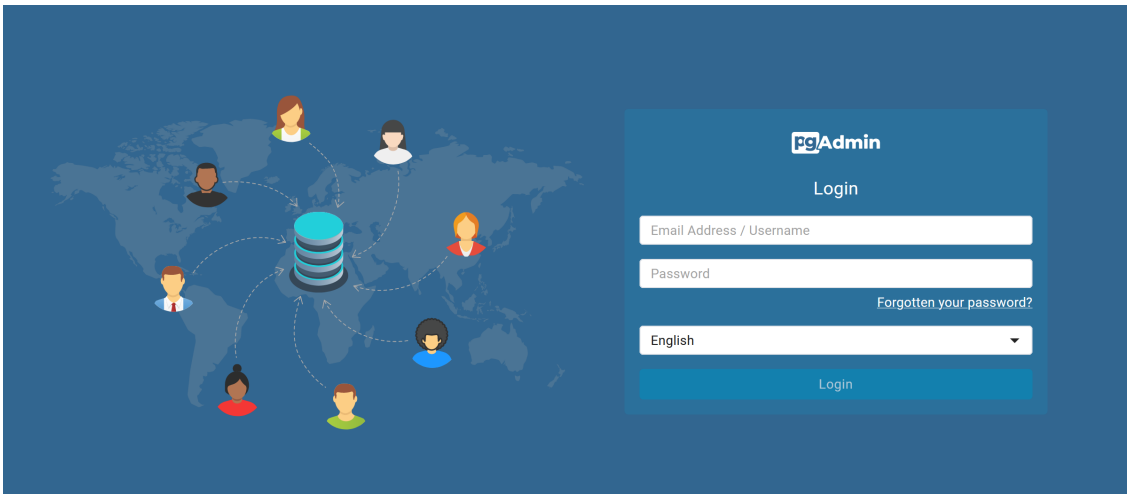
6.5 Connecting to the MLFlow UI

- MLFlow offers a convenient interface that can be accessed via `http://localhost:5001`.

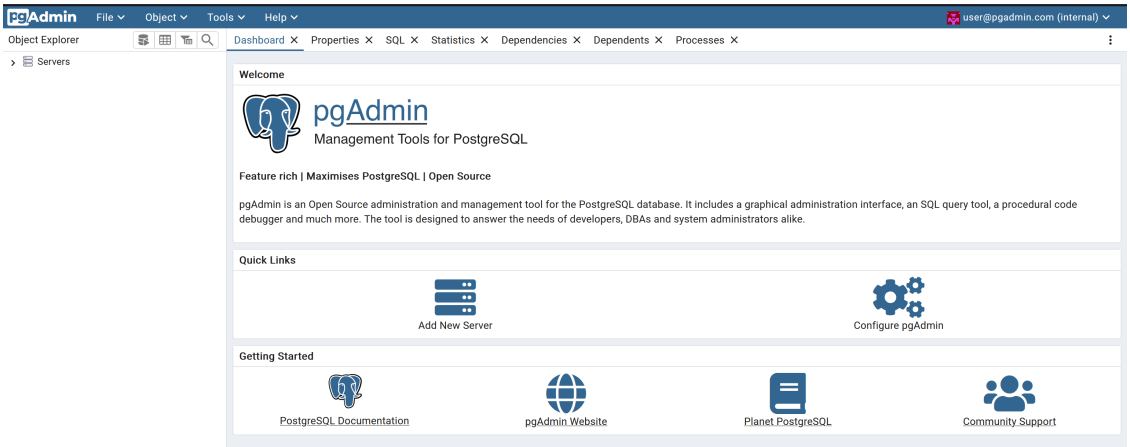


6.6 Connecting to PgAdmin

- PgAdmin4 is management software for PostgreSQL Server.
- You can open the local implementation by navigating to http://localhost:5051. You will find a screen like the one below.



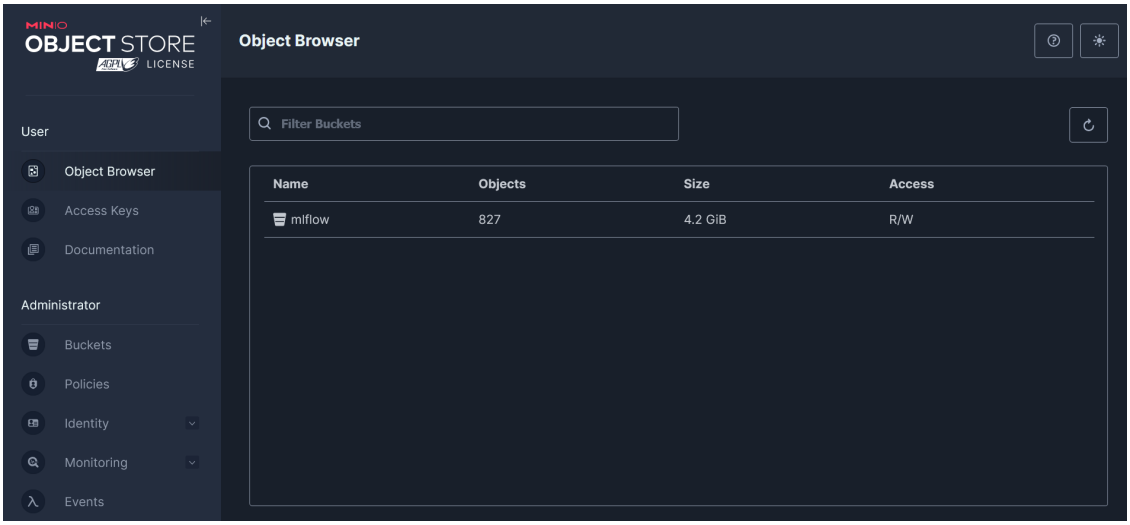
- Login using the credentials specified in the file `./05_src/experiment_tracking/.env`. Notice there are two sets of credentials, use the ones for PgAdmin4. After authentication, you will see a screen like the one below.



- Click on "Add New Server":
  - In the *General* Tab, under Name enter: localhost.
  - Under the *Connection* Tab, use Host name *postgres* (this is the name of the service in the docker compose file).
  - Username and password are the ones found in the `./05_src/experiment_tracking/.env` file.

6.7 Connect to MinIO

- The interface for MinIO can be reached via `http://localhost:9001`
- The credentials can be found in the `./05_src/experiment_tracking/.env` file.



6.8 Learn More

- Containers and containerization are topics well beyond the scope of this course. However, we will use containerized applications to help us implement certain patterns.
- If you are interested in Docker, a good place to start is the Official Docker Guides.

7 On Jupyter Notebooks

- Jupyter Notebooks are great for drafting code, fast experimentation, demos, documentation, and some prototypes.
- They are not great for production code and not great for experiment tracking.

8 A Note about Copilot

- AI-assisted coding is a reality. I would like your opinions about the use of this technology.
- I will start the course with Copilot on, but if it becomes too distracting, I will be happy to turn it off.
- Copilot is a nice tool, but it is not for everyone. If you are starting to code or are trying to level up, I recommend that you leave AI assistants (Copilot, ChatGPT, etc.) for later.