# **Learning Notes: Data Engineering Workshop 1**

### **Introduction to Docker**

* Docker packages applications and their dependencies into lightweight, portable containers.
* Containers run consistently across any environment—local, server, or cloud.
* Benefits include smaller OS footprint, easier collaboration (dev, QA, ops), scalability, and cloud readiness.
* Post-install setup: add user to docker group to run Docker without sudo.

### **Essential Docker Commands**

* docker --version — check installed Docker version.
* docker pull <image> — download Docker images.
* docker images — list local images.
* docker run -ti <image> — run container interactively (Ctrl+D to exit).

### **Building Custom Docker Images**

* Dockerfile defines how to build an image, including base OS and app setup.
* Commands to build and run images:
*   
  docker build ./ -t simple\_python
*   
  Use docker images to verify image creation.

### **Web Scraping with Python**

* requests module simplifies sending HTTP requests to fetch website content.
* urllib is a built-in Python module for URL handling and fetching data.
* BeautifulSoup extracts data from HTML/XML; find() and find\_all() locate tags.
* re module allows pattern matching in strings; useful for complex HTML parsing.
* Practical script combines requests, BeautifulSoup, and re to scrape structured data.
* .text attribute extracts readable text from HTML elements.

### **Dockerizing a Web Scraper**

* Docker packages the scraper and dependencies into a container for portability.
* Dockerfile typically includes:
  + Base image (Python alpine)
  + Copying the script
  + Installing required packages (requests, bs4, html5lib)
  + Running the script via ENTRYPOINT and CMD.
* Build and run Docker image commands:
*   
  docker build --no-cache --network=host ./ -t simple\_python

### **Working with Docker Containers**

* Containers are standalone packages with everything needed to run an app.
* Docker Compose manages multi-container apps via a YAML configuration.
* Compose allows easy start-up and management of services with a single command.

### **Hands-On Exercise Summary**

* Create Dockerfile with Python environment setup and script copy.
* Build image: docker build --no-cache --network=host ./ -t workshop1
* Define docker-compose.yml with service configuration, port mapping, and volume mounts.
* Start services in detached mode: docker-compose up -d
* Access container shell: docker exec -it workshop\_python\_container sh
* Run Python script inside container: python web\_scraping\_sample.py

# **PostgreSQL**

### **Key Features**

* Free and open source
* Supports data integrity
* Handles multiple data types
* Highly extensible and secure
* Reliable and robust

### **Types of JOINS**

* **CROSS JOIN**: Pairs every row of table1 with every row of table2
* **INNER JOIN**: Returns rows with matching values in both tables based on a condition
* **LEFT OUTER JOIN**: All rows from left table + matched rows from right table; unmatched right side is NULL
* **RIGHT OUTER JOIN**: All rows from right table + matched rows from left table; unmatched left side is NULL
* **FULL OUTER JOIN**: Combines left and right joins; unmatched rows from both sides included with NULLs

### **Setting Up PostgreSQL in Docker**

**Steps:**

1. Added PostgreSQL container to docker-compose.yml:
2.   
   psql-db:
3.   
   Launched containers:
4.   
   docker-compose up -d
5.   
   Entered PostgreSQL container:
6.   
   docker exec -it psql-db bash

### **PostgreSQL Commands**

#### **Create database and connect**

****CREATE DATABASE demo;

#### **Create tables**

****CREATE TABLE zoo\_1 (

#### **Insert data**

****INSERT INTO zoo\_1(id, animal) VALUES

### **JOIN Queries Examples**

#### **INNER JOIN**

Returns rows with matching animals in both tables

SELECT zoo\_1.id AS id\_a, zoo\_1.animal AS animal\_a, zoo\_2.id AS id\_b, zoo\_2.animal AS animal\_b

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| **id\_a** | **animal\_a** | **id\_b** | **animal\_b** |
| 1 | Lion | 2 | Lion |
| 2 | Tiger | 1 | Tiger |

#### 

#### 

#### **LEFT JOIN**

All rows from zoo\_1 and matching rows from zoo\_2 (NULL if no match)

SELECT zoo\_1.id, zoo\_1.animal, zoo\_2.id, zoo\_2.animal

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **animal** | **id** | **animal** |
| 1 | Lion | 2 | Lion |
| 2 | Tiger | 1 | Tiger |
| 3 | Wolf |  |  |
| 4 | Fox |  |  |

#### 

#### 

#### **RIGHT JOIN / RIGHT OUTER JOIN**

All rows from zoo\_2 and matching rows from zoo\_1 (NULL if no match)

SELECT zoo\_1.id, zoo\_1.animal, zoo\_2.id, zoo\_2.animal

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **animal** | **id** | **animal** |
| 2 | Tiger | 1 | Tiger |
| 1 | Lion | 2 | Lion |
|  |  | 3 | Rhino |
|  |  | 4 | Panther |

#### 

#### 

#### **FULL OUTER JOIN**

All rows from both tables; unmatched rows have NULLs in missing columns

SELECT zoo\_1.id, zoo\_1.animal, zoo\_2.id, zoo\_2.animal

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **animal** | **id** | **animal** |
| 1 | Lion | 2 | Lion |
| 2 | Tiger | 1 | Tiger |
| 3 | Wolf |  |  |
| 4 | Fox |  |  |
|  |  | 4 | Panther |
|  |  | 3 | Rhino |

### **CROSS JOIN**

### Pairs every row from zoo\_1 with every row from zoo\_2 (Cartesian product). No join condition is needed.

SELECT zoo\_1.id, zoo\_1.animal, zoo\_2.id, zoo\_2.animal

**Output:** *(16 rows total — 4 from zoo\_1 × 4 from zoo\_2)*

|  |  |  |  |
| --- | --- | --- | --- |
| **id (zoo\_1)** | **animal (zoo\_1)** | **id (zoo\_2)** | **animal (zoo\_2)** |
| 1 | Lion | 1 | Tiger |
| 1 | Lion | 2 | Lion |
| 1 | Lion | 3 | Rhino |
| 1 | Lion | 4 | Panther |
| 2 | Tiger | 1 | Tiger |
| 2 | Tiger | 2 | Lion |
| 2 | Tiger | 3 | Rhino |
| 2 | Tiger | 4 | Panther |
| 3 | Wolf | 1 | Tiger |
| 3 | Wolf | 2 | Lion |
| 3 | Wolf | 3 | Rhino |
| 3 | Wolf | 4 | Panther |
| 4 | Fox | 1 | Tiger |
| 4 | Fox | 2 | Lion |
| 4 | Fox | 3 | Rhino |
| 4 | Fox | 4 | Panther |

# **Git**

* GitHub allows creating remote repositories to store and manage code collaboratively.
* You can clone a GitHub repo locally using git clone <repo-url>.
* Typical Git workflow:
  1. Modify files locally
  2. Stage changes with git add
  3. Commit changes with git commit -m "message"
  4. Push commits to remote with git push origin <branch>
* Basic Git commands:
  1. **git init** to create a new repo
  2. **git add** to stage changes
  3. **git** **commit** to save changes locally
  4. **git** **status** to check repo state
  5. **git** **config** to set user identity for commits
* Pull Requests allow you to propose code changes from one branch to be merged into another, facilitating collaboration and code review.

# 

# 