Module 1: Getting Started TimescaleDB

In this module, you will:

1. Install PostgreSQL with the TimescaleDB extension.
2. Install InfluxDB (for comparison).
3. Install Grafana for visualization.
4. Use Docker Compose to spin up all services together.
5. Insert your first timestamped record.
6. Run a few basic queries (latest value, range filter).

This module is designed for beginners, so we'll explain **what** we're doing and **why** at each step.

Step 1: Install PostgreSQL with TimescaleDB

What is PostgreSQL?

PostgreSQL is a powerful open-source relational database system. It's widely used for production systems.

What is TimescaleDB?

TimescaleDB is a PostgreSQL extension that adds **time-series database capabilities**—optimized for storing and querying timestamped data (IoT, financial data, metrics, etc.).

Installation Approach

Instead of installing PostgreSQL and TimescaleDB directly on your machine (which can vary by OS), we'll use **Docker** for a consistent environment across macOS, Linux, and Windows.

Step 2: Install InfluxDB

What is InfluxDB?

InfluxDB is a **purpose-built time-series database**, designed from the ground up for timestamped data.

We'll install it alongside PostgreSQL+TimescaleDB so you can compare both.

Step 3: Install Grafana

What is Grafana?

Grafana is a popular visualization tool for dashboards. It supports both PostgreSQL/TimescaleDB and InfluxDB as data sources. We'll use it to visualize data.

Step 4: Spin Up Databases Using Docker Compose

Prerequisites

- **Docker Desktop** installed
  - macOS: Docker for Mac
  - Windows: Docker for Windows
  - Linux: Install Docker + Docker Compose plugin via package manager

Verify installation:
```
docker --version
docker compose version
```

Both should show recent versions (Docker 27+ and Compose v2+ as of 2025).

Create `docker-compose.yml`

Create a new folder for this module and inside it, create a file named `docker-compose.yml`.

```
services:
  postgres:
    image: timescale/timescaledb:latest-pg16
    container_name: timescaledb
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=admin123
      - POSTGRES_DB=metricsdb
    ports:
      - "5432:5432"
    volumes:
      - pg_data:/var/lib/postgresql/data

  influxdb:
    image: influxdb:2.7
    container_name: influxdb
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=admin123
      - DOCKER_INFLUXDB_INIT_ORG=example-org
      - DOCKER_INFLUXDB_INIT_BUCKET=example-bucket
    ports:
      - "8086:8086"
    volumes:
      - influx_data:/var/lib/influxdb2

  grafana:
```

```
    image: grafana/grafana-oss:11.2.0
    container_name: grafana
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin123
    ports:
      - "3000:3000"
    volumes:
      - grafana_data:/var/lib/grafana

volumes:
  pg_data:
  influx_data:
  grafana_data:
```

Start the services

From the same directory:

```
# fetch updated images
docker compose pull

# start stack
docker compose up -d
```

Check containers:

```
docker ps
```

You should see three running containers: `timescaledb`, `influxdb`, and `grafana`.

Step 5: Insert Your First Timestamped Record

Connect to PostgreSQL + TimescaleDB

```
docker exec -it timescaledb psql -U admin -d metricsdb
```

Inside psql, create a table for storing temperature readings:

```
CREATE TABLE sensor_data (
    time TIMESTAMPTZ NOT NULL,
    sensor_id INT,
    temperature DOUBLE PRECISION
);

-- Convert table into hypertable (TimescaleDB magic)
SELECT create_hypertable('sensor_data', 'time');
```

Insert your first record:

```
INSERT INTO sensor_data (time, sensor_id, temperature)
VALUES (NOW(), 1, 22.5);
```

Check that it was inserted:

```
SELECT * FROM sensor_data;
```

You should see one row with your data.

Connect to InfluxDB

InfluxDB uses a different query language (Flux or InfluxQL).
First, open the InfluxDB UI in your browser: http://localhost:8086.
Log in with:

- Username: `admin`
- Password: `admin123`

Create a new bucket or use the default `example-bucket`.
Write a sample point using the InfluxDB CLI:

```
docker exec -it influxdb influx write \
  --org example-org \
  --bucket example-bucket \
  --precision s \
  "temperature,sensor_id=1 value=22.5 $(date +%s)"
```

Step 6: Run a Basic Query

PostgreSQL + TimescaleDB

In psql:

```
-- Get latest value
SELECT * FROM sensor_data ORDER BY time DESC LIMIT 1;

-- Get values in last 1 hour
SELECT * FROM sensor_data
WHERE time > NOW() - INTERVAL '1 hour';
```

InfluxDB

Using Influx CLI:

```
docker exec -i influxdb influx query <<'EOF'
from(bucket:"example-bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "temperature")
EOF
'
```

Step 7: Visualize with Grafana

- Open Grafana: http://localhost:3000
  Username: `admin`, Password: `admin123`
- Add a new data source:
  - PostgreSQL: host `timescaledb:5432`, database `metricsdb`, user `admin`, password `admin123`
- Create a simple dashboard to visualize your temperature values over time.

Troubleshooting Tips for Module 1

Even though Docker simplifies installation, beginners often face common issues. Here's how to solve them.

1. Docker or Docker Compose Issues

Error: `docker: command not found`

- Make sure Docker is installed and running.
  - macOS: open **Docker Desktop** from Applications.
  - Windows: open **Docker Desktop** and enable WSL 2 backend.

Linux: check installation guide:

```
sudo systemctl start docker
sudo systemctl enable docker
```

Verify:

```
docker --version
```

Error: `docker compose: command not found`

- Docker Compose is now built into Docker as `docker compose` (not `docker-compose`).
- If you see this error, upgrade Docker to the latest version.

2. Port Conflicts

Error: `Bind for 0.0.0.0:5432 failed: port is already allocated`

This means something (like an existing PostgreSQL) is already using port 5432.

Solution:
Stop the service using it:

Linux/macOS:
```
sudo lsof -i :5432
sudo kill -9 <PID>
```

Windows PowerShell:
```
netstat -ano | findstr :5432
taskkill /PID <PID> /F
```

Or change the port in `docker-compose.yml`, e.g.:
```
ports:
  - "55432:5432"
```

3. Resetting Data Volumes

If you inserted bad data or want a fresh start, remove volumes:
```
docker compose down -v
```

This deletes all data stored in TimescaleDB, InfluxDB, and Grafana. Start again with:
```
docker compose up -d
```

4. PostgreSQL / TimescaleDB Issues

Error: `psql: could not connect to server`

Make sure container is running:
```
docker ps
```

Enter container shell:
```
docker exec -it timescaledb bash
```

From inside:
```
psql -U admin -d metricsdb
```

5. InfluxDB Issues

Can't log in at `http://localhost:8086`

Make sure the container is healthy:

```
docker logs influxdb
```

If you see `setup not completed`, check that `DOCKER_INFLUXDB_INIT_*` environment variables are correct in `docker-compose.yml`.

Reset InfluxDB by removing its volume:
docker compose down -v
docker compose up -d

6. Grafana Issues

Error: Cannot log in at `http://localhost:3000`

- Default credentials:
  - Username: `admin`
  - Password: `admin123`

If you forgot password, reset by removing the Grafana volume:

docker compose down -v
docker compose up -d

Error: Datasource connection fails

- For PostgreSQL: use `timescaledb` as the host (not `localhost`), because Grafana connects from inside Docker.
- For InfluxDB: use `http://influxdb:8086`.

7. General Debugging

View logs for a container:
docker logs timescaledb
docker logs influxdb
docker logs grafana

Restart a single service:
docker compose restart grafana

Stop all:
docker compose down

Great — below are **two ready-to-run test scripts** (one Bash for macOS / Linux, one PowerShell for Windows/WSL) that:

- Bring up the Docker Compose stack (if not already running),

- Wait for TimescaleDB and InfluxDB to be reachable,
- Create the TimescaleDB hypertable (idempotent),
- Insert a few timestamped records into TimescaleDB and InfluxDB,
- Run two queries for each DB: latest value and a range (last 1 hour),
- Print human-friendly output so students can verify everything is working.

1) Bash script for macOS / Linux / WSL

Create a file named `test_insert_and_query.sh` and paste this content:

```
#!/usr/bin/env bash
# test_insert_and_query.sh
# Portable test script for TimescaleDB (Postgres+Timescale), InfluxDB, Grafana
# Works on macOS, Linux, and WSL.
# Usage: chmod +x test_insert_and_query.sh && ./test_insert_and_query.sh

set -euo pipefail

# === Configuration (match your docker-compose.yml) ===
PG_CONTAINER="timescaledb"
PG_USER="admin"
PG_DB="metricsdb"

INFLUX_CONTAINER="influxdb"
INFLUX_ORG="example-org"
INFLUX_BUCKET="example-bucket"

# === Start stack ===
echo "Starting docker compose (if not already running)..."
docker compose up -d

# === Wait for Postgres/TimescaleDB ===
echo "Waiting for TimescaleDB to be ready..."
attempt=0
max_attempts=60
while ! docker exec -i "${PG_CONTAINER}" pg_isready -U "${PG_USER}" >/dev/null 2>&1; do
  attempt=$((attempt+1))
  echo "  waiting for postgres... attempt ${attempt}/${max_attempts}"
  if [ "${attempt}" -ge "${max_attempts}" ]; then
```

```
    echo "Postgres did not become ready in time. Check container logs: docker logs
${PG_CONTAINER}"
    exit 1
  fi
  sleep 2
done
echo "Postgres is ready."

# === Create table and hypertable (idempotent) ===
echo "Creating sensor_data table and hypertable (if not exists)..."
docker exec -i "${PG_CONTAINER}" psql -U "${PG_USER}" -d "${PG_DB}" -v
ON_ERROR_STOP=1 <<'SQL'
CREATE TABLE IF NOT EXISTS sensor_data (
  time TIMESTAMPTZ NOT NULL,
  sensor_id INT,
  temperature DOUBLE PRECISION
);
DO $$
BEGIN
  IF NOT EXISTS (
    SELECT 1 FROM information_schema.tables
    WHERE table_name = 'sensor_data'
  ) THEN
    -- table created above, nothing else
    NULL;
  END IF;
  -- Create hypertable only if not already a hypertable
  IF NOT EXISTS (SELECT 1 FROM timescaledb_information.hypertables WHERE
hypertable_name='sensor_data') THEN
    PERFORM create_hypertable('sensor_data', 'time');
  END IF;
END;
$$;
SQL

# === Insert sample rows into TimescaleDB ===
echo "Inserting 3 timestamped rows into TimescaleDB..."
docker exec -i "${PG_CONTAINER}" psql -U "${PG_USER}" -d "${PG_DB}" <<'SQL'
INSERT INTO sensor_data (time, sensor_id, temperature) VALUES
  (NOW() - INTERVAL '5 minutes', 1, 21.7),
  (NOW() - INTERVAL '2 minutes', 1, 22.1),
  (NOW(), 1, 22.6);
SQL
```

```
# === Query TimescaleDB: latest and last 1 hour ===
echo
echo "=== TimescaleDB: latest row ==="
docker exec -i "${PG_CONTAINER}" psql -U "${PG_USER}" -d "${PG_DB}" -c "SELECT *
FROM sensor_data ORDER BY time DESC LIMIT 1;"

echo
echo "=== TimescaleDB: rows from last 1 hour ==="
docker exec -i "${PG_CONTAINER}" psql -U "${PG_USER}" -d "${PG_DB}" -c "SELECT *
FROM sensor_data WHERE time > NOW() - INTERVAL '1 hour' ORDER BY time ASC;"

# === Wait for InfluxDB HTTP API to be healthy ===
echo
echo "Waiting for InfluxDB HTTP API to be healthy..."
attempt=0
max_attempts=60
while true; do
  attempt=$((attempt+1))
  set +e
  status=$(docker exec "${INFLUX_CONTAINER}" bash -lc "curl -sS -o /dev/null -w
'%{http_code}' http://localhost:8086/health" 2>/dev/null || echo "000")
  set -e
  if [ "${status}" = "200" ]; then
    break
  fi
  echo "  waiting for influxdb... attempt ${attempt}/${max_attempts} (status=${status})"
  if [ "${attempt}" -ge "${max_attempts}" ]; then
    echo "InfluxDB did not become healthy in time. Check container logs: docker logs
${INFLUX_CONTAINER}"
    exit 1
  fi
  sleep 2
done
echo "InfluxDB appears healthy."

# === Write sample points to InfluxDB (portable, safe) ===
echo "Writing 3 points to InfluxDB bucket=${INFLUX_BUCKET}..."

TS1=$(python3 - <<'PY'
import time
print(int(time.time() - 5*60))
PY
)
TS2=$(python3 - <<'PY'
```

```
import time
print(int(time.time() - 2*60))
PY
)
TS3=$(python3 - <<'PY'
import time
print(int(time.time()))
PY
)

{
  printf "temperature,sensor_id=1 value=21.7 %s\n" "$TS1"
  printf "temperature,sensor_id=1 value=22.1 %s\n" "$TS2"
  printf "temperature,sensor_id=1 value=22.6 %s\n" "$TS3"
} | docker exec -i "${INFLUX_CONTAINER}" influx write --org "${INFLUX_ORG}" --bucket
"${INFLUX_BUCKET}" --precision s -

# === Query InfluxDB: last 1 hour and latest point (Flux) ===
echo
echo "=== InfluxDB: last 1 hour (flux) ==="
docker exec -i "${INFLUX_CONTAINER}" influx query --org "${INFLUX_ORG}" --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "temperature")
  |> filter(fn: (r) => r._field == "value")
  |> sort(columns:["_time"])
FLUX

echo
echo "=== InfluxDB: latest point (flux) ==="
docker exec -i "${INFLUX_CONTAINER}" influx query --org "${INFLUX_ORG}" --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "temperature")
  |> filter(fn: (r) => r._field == "value")
  |> sort(desc: true, columns: ["_time"])
  |> limit(n:1)
FLUX

echo
echo "All done. If you saw rows for TimescaleDB and points for InfluxDB, your stack is working."
```

Make the script executable and run it:

```
chmod +x test_insert_and_query.sh
./test_insert_and_query.sh
```

Notes / explanations (Bash script):

- We use `pg_isready` inside the Postgres container to detect readiness.
- Timescale hypertable creation is guarded so the script is idempotent.
- We write Influx points with the `influx` CLI inside the container; when you `docker exec` into the container, the CLI can use the local setup created by the `DOCKER_INFLUXDB_INIT_*` environment variables.
- If a command fails, the script exits (set -e) so you immediately see where it failed.

2) PowerShell script for Windows (PowerShell 7+ recommended)

Create a file named `test_insert_and_query.ps1` and paste this content:

```
<#
  test_insert_and_query.ps1
  Portable PowerShell test script for TimescaleDB (Postgres+Timescale), InfluxDB, Grafana
  Usage (PowerShell 7+ recommended):
    Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
    .\test_insert_and_query.ps1
#>

# Stop on error
$ErrorActionPreference = "Stop"

# === Configuration (match your docker-compose.yml) ===
$PG_CONTAINER = "timescaledb"
$PG_USER      = "admin"
$PG_DB        = "metricsdb"

$INFLUX_CONTAINER = "influxdb"
$INFLUX_ORG       = "example-org"
$INFLUX_BUCKET    = "example-bucket"

function Write-Info($s) {
  Write-Host $s -ForegroundColor Cyan
}
```

```
# === Start stack ===
Write-Info "Starting docker compose (if not already running)..."
docker compose up -d | Out-Null

# === Wait for Postgres/TimescaleDB ===
Write-Info "Waiting for TimescaleDB to be ready..."
$attempt = 0
$maxAttempts = 60
while ($true) {
  $attempt++
  docker exec -i $PG_CONTAINER pg_isready -U $PG_USER >$null 2>&1
  if ($LASTEXITCODE -eq 0) { break }
  Write-Host "  waiting for postgres... attempt $attempt / $maxAttempts"
  if ($attempt -ge $maxAttempts) {
    Write-Error "Postgres did not become ready in time. Check container logs: docker logs
$PG_CONTAINER"
    exit 1
  }
  Start-Sleep -Seconds 2
}
Write-Info "Postgres is ready."

# === Create table and hypertable (idempotent) ===
Write-Info "Creating sensor_data table and hypertable (if not exists)..."
$createSql = @"
CREATE TABLE IF NOT EXISTS sensor_data (
  time TIMESTAMPTZ NOT NULL,
  sensor_id INT,
  temperature DOUBLE PRECISION
);
DO $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM timescaledb_information.hypertables WHERE
hypertable_name='sensor_data') THEN
    PERFORM create_hypertable('sensor_data', 'time');
  END IF;
END;
$$;
"@

$createSql | docker exec -i $PG_CONTAINER psql -U $PG_USER -d $PG_DB

# === Insert sample rows into TimescaleDB ===
Write-Info "Inserting 3 timestamped rows into TimescaleDB..."
```

```
$insertSql = @"
INSERT INTO sensor_data (time, sensor_id, temperature) VALUES
  (NOW() - INTERVAL '5 minutes', 1, 21.7),
  (NOW() - INTERVAL '2 minutes', 1, 22.1),
  (NOW(), 1, 22.6);
"@
$insertSql | docker exec -i $PG_CONTAINER psql -U $PG_USER -d $PG_DB

# === Query TimescaleDB: latest and last 1 hour ===
Write-Host ""
Write-Info "=== TimescaleDB: latest row ==="
docker exec -i $PG_CONTAINER psql -U $PG_USER -d $PG_DB -c "SELECT * FROM
sensor_data ORDER BY time DESC LIMIT 1;"

Write-Host ""
Write-Info "=== TimescaleDB: rows from last 1 hour ==="
docker exec -i $PG_CONTAINER psql -U $PG_USER -d $PG_DB -c "SELECT * FROM
sensor_data WHERE time > NOW() - INTERVAL '1 hour' ORDER BY time ASC;"

# === Wait for InfluxDB HTTP API to be healthy ===
Write-Host ""
Write-Info "Waiting for InfluxDB HTTP API to be healthy..."
$attempt = 0
$maxAttempts = 60
while ($true) {
  $attempt++
  try {
    $status = docker exec $INFLUX_CONTAINER bash -lc "curl -sS -o /dev/null -w
'%{http_code}' http://localhost:8086/health" 2>$null
  } catch {
    $status = "000"
  }
  if ($status -eq "200") { break }
  Write-Host "  waiting for influxdb... attempt $attempt / $maxAttempts (status=$status)"
  if ($attempt -ge $maxAttempts) {
    Write-Error "InfluxDB did not become healthy in time. Check container logs: docker logs
$INFLUX_CONTAINER"
    exit 1
  }
  Start-Sleep -Seconds 2
}
Write-Info "InfluxDB appears healthy."

# === Write sample points to InfluxDB ===
```

```
Write-Info "Writing 3 points to InfluxDB bucket=$INFLUX_BUCKET ..."

function Get-EpochSeconds([int]$subtractSeconds = 0) {
  $utcNow = (Get-Date).ToUniversalTime()
  $target = $utcNow.AddSeconds(-1 * $subtractSeconds)
  $epoch = [int][double]((New-TimeSpan -Start (Get-Date "1970-01-01T00:00:00Z") -End
$target).TotalSeconds)
  return $epoch
}

$TS1 = Get-EpochSeconds -subtractSeconds (5*60)   # 5 minutes ago
$TS2 = Get-EpochSeconds -subtractSeconds (2*60)   # 2 minutes ago
$TS3 = Get-EpochSeconds -subtractSeconds 0        # now

$payloadLines = @()
$payloadLines += "temperature,sensor_id=1 value=21.7 $TS1"
$payloadLines += "temperature,sensor_id=1 value=22.1 $TS2"
$payloadLines += "temperature,sensor_id=1 value=22.6 $TS3"
$payload = ($payloadLines -join "`n") + "`n"

$bytes = [System.Text.Encoding]::UTF8.GetBytes($payload)
$bytes | docker exec -i $INFLUX_CONTAINER influx write --org $INFLUX_ORG --bucket
$INFLUX_BUCKET --precision s -

# === Query InfluxDB: last 1 hour and latest point ===
Write-Host ""
Write-Info "=== InfluxDB: last 1 hour (flux) ==="
$fluxQuery = @'
from(bucket:"example-bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "temperature")
  |> filter(fn: (r) => r._field == "value")
  |> sort(columns:["_time"])
'@
$fluxQuery | docker exec -i $INFLUX_CONTAINER influx query --org $INFLUX_ORG --raw

Write-Host ""
Write-Info "=== InfluxDB: latest point (flux) ==="
$fluxLatest = @'
from(bucket:"example-bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "temperature")
  |> filter(fn: (r) => r._field == "value")
  |> sort(desc: true, columns: ["_time"])
```

```
  |> limit(n:1)
'@
$fluxLatest | docker exec -i $INFLUX_CONTAINER influx query --org $INFLUX_ORG --raw

Write-Host ""
Write-Info "All done. If you saw rows for TimescaleDB and points for InfluxDB, your stack is working."
```

Run it in PowerShell:

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
.\test_insert_and_query.ps1
```

Notes / explanations (PowerShell script):

- Uses `docker exec` commands just like the Bash script; behavior is the same.
- Times are computed using PowerShell date arithmetic.
- Flux queries are passed to the `influx query` command; results print in CSV-style raw output.

Expected output examples

You should see:

- `psql` printed rows for TimescaleDB with `time`, `sensor_id`, `temperature`.
- Influx CLI printed Flux query results (rows with `_time`, `_value` or `_field` and `_measurement` names).

Example (TimescaleDB `psql`) snippet:

```
              time               | sensor_id | temperature
---------------------------------+-----------+-------------
 2025-09-15 08:19:00.123+00      |         1 |        22.6
(1 row)
```

Example (Influx raw output) snippet:

```
#datatype,string,long,dateTime:RFC3339,string,double
#group,false,false,true,false,false
#default,_result,,,,
,result,table,_time,_field,_value
,,0,2025-09-15T08:19:00Z,value,22.6
```

Troubleshooting hints (quick)

- If the scripts fail on Postgres readiness, run `docker logs timescaledb` to inspect error messages (e.g., password mismatch, file-permissions).
- If Influx write/query errors occur, inspect `docker logs influxdb`.
- If `docker compose up -d` fails due to port conflicts, either stop the conflicting service or change ports in `docker-compose.yml`.
- If Grafana isn't reachable at `http://localhost:3000`, check `docker ps` and `docker logs grafana`.

Final notes & safety

- The scripts are **idempotent** (they create the hypertable only if missing and append sample data). Run them multiple times; you will get more sample points.
- They assume the same credentials you used in the `docker-compose.yml` in the earlier module (user `admin` / password `admin123`, DB `metricsdb`, Influx org `example-org` and bucket `example-bucket`). If you changed those, edit the scripts to match.
- These scripts do **not** modify your host machine except for running Docker containers and the volumes defined in `docker-compose.yml`.

Cleanup Steps

When you finish experimenting with TimescaleDB, InfluxDB, and Grafana, you may want to:

- Stop the services (to free system resources).
- Remove containers and networks.
- Optionally remove stored data (if you want a completely fresh start).

- Uninstall Docker if you don't need it anymore.

Here are the steps.

1. Stop the Containers (but keep data)

This will stop the services but keep their data volumes so you can resume later.

```
docker compose down
```

Next time, simply run:

```
docker compose up -d
```

to continue where you left off.

2. Stop and Remove Data Volumes (reset to clean state)

If you want to remove **all stored data** (Postgres/TimescaleDB tables, InfluxDB buckets, Grafana dashboards):

```
docker compose down -v
```

- `-v` deletes all **volumes** defined in `docker-compose.yml`.
- After this, starting again (`docker compose up -d`) gives you a fresh setup.

3. Remove All Containers and Volumes (if running multiple stacks)

If you want to remove everything Docker is running on your system:

```
docker ps -a        # list all containers
docker stop $(docker ps -aq)
docker rm $(docker ps -aq)

docker volume ls    # list all volumes
docker volume rm $(docker volume ls -q)
```

Warning: This removes all Docker containers and volumes on your machine (not just this project).

4. Remove Images (optional)

If you don't want TimescaleDB, InfluxDB, or Grafana images stored locally:

```
docker rmi timescale/timescaledb:latest-pg16
docker rmi influxdb:2.7
docker rmi grafana/grafana-oss:11.2.0
```

You can also remove unused images:

```
docker image prune -a
```

5. Uninstall Docker (if no longer needed)

- **macOS:** Remove Docker Desktop from Applications.
- **Windows:** Uninstall Docker Desktop from Control Panel.
- **Linux:** Use your package manager (e.g., `sudo apt remove docker-desktop docker-ce docker-ce-cli`).

6. Verify Cleanup

Run:
docker ps -a
docker volume ls
docker images

If everything is cleaned, you should see **no containers**, **no volumes**, and only minimal images (or none).

7. Restarting After Cleanup

If you want to start again later:
Make sure `docker-compose.yml` is still in your folder.

Run:
docker compose up -d
Re-run the **test script** to insert sample data again.