Module 6 - Real-Time Queries & Alerts (TimescaleDB + InfluxDB + Grafana)

This end-to-end tutorial shows how to:

- create a TimescaleDB hypertable and continuous aggregate (hourly CPU),
- write example points into InfluxDB (v2) and create v1-compat credentials so Grafana (InfluxQL) works,
- configure Grafana datasources (TimescaleDB/Postgres + InfluxDB/InfluxQL),
- build dashboards (hourly aggregate & live raw) and an alert when CPU > 90%,
- cleanup afterwards.

Assumptions: Docker Compose as in Module 1 is available, services named `timescaledb`, `influxdb`, `grafana`. You run this in a local learning environment and accept `docker compose down -v` will remove demo volumes.

0 - Compose / Token (quick)

If you want a reproducible local demo, put a generated admin token into `.env` and reference it in `docker_compose.yml`:

```
# generate strong token and save to .env (Linux/macOS)
TOKEN=$(openssl rand -hex 32)
printf 'DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=%s\n' "$TOKEN" > .env
```

Ensure `docker_compose.yml` uses that env var:

```
- DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${DOCKER_INFLUXDB_INIT_ADMIN_TOKEN}
```

Start the stack (warning: deletes volumes):

```
docker compose down -v
docker compose up -d
```

1 - TimescaleDB: table + hypertable + sample data

Open psql:

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

Run:

```
CREATE TABLE IF NOT EXISTS cpu_metrics (
  time        TIMESTAMPTZ       NOT NULL,
  sensor_id   INT               NOT NULL,
  cpu         DOUBLE PRECISION  NOT NULL
);

SELECT create_hypertable('cpu_metrics', 'time', if_not_exists => TRUE);

-- Insert synthetic data: 24 hours, every 5 minutes, sensors 1..3
INSERT INTO cpu_metrics (time, sensor_id, cpu)
SELECT
  NOW() - (i * INTERVAL '5 minutes') AS time,
  s.sensor_id,
  50 + (random() * 50)::int
FROM generate_series(0, 288) AS i
CROSS JOIN (SELECT unnest(ARRAY[1,2,3]) AS sensor_id) s;

-- quick verification
SELECT COUNT(*) AS total_rows, MIN(time) AS first_ts, MAX(time) AS last_ts FROM
cpu_metrics;
```

2 - TimescaleDB: continuous aggregate (hourly) + policy

Create view and schedule refresh:

```
CREATE MATERIALIZED VIEW IF NOT EXISTS hourly_cpu_usage
WITH (timescaledb.continuous) AS
SELECT
```

```
  time_bucket('1 hour', time) AS bucket,
  sensor_id,
  AVG(cpu) AS avg_cpu,
  MAX(cpu) AS max_cpu,
  COUNT(*) AS samples
FROM cpu_metrics
GROUP BY bucket, sensor_id
WITH NO DATA;

SELECT add_continuous_aggregate_policy(
  'hourly_cpu_usage',
  start_offset => INTERVAL '1 day',
  end_offset   => INTERVAL '1 hour',
  schedule_interval => INTERVAL '1 hour'
);

-- materialize a recent window (modern Timescale uses a procedure)
CALL refresh_continuous_aggregate(
  'hourly_cpu_usage',
  NOW() - INTERVAL '2 days',
  NOW() - INTERVAL '1 hour'
);

SELECT * FROM hourly_cpu_usage ORDER BY bucket DESC LIMIT 10;
```

Notes:

- `time_bucket('1 hour', ...)` groups timestamps into hourly buckets.
- `add_continuous_aggregate_policy` schedules background refreshes.
- Use `CALL refresh_continuous_aggregate(...)` to materialize immediately (modern Timescale).

3 - InfluxDB: write sample points

(We use InfluxDB v2 but will expose a v1-compat auth for Grafana.)

Write a few points (server will timestamp them):

```
source .env   # loads DOCKER_INFLUXDB_INIT_ADMIN_TOKEN variable
cat <<LP | docker exec -i influxdb influx write \
  --bucket example-bucket --org example-org --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN" --format=lp
cpu_metrics,sensor_id=1 cpu=72.4
```

```
cpu_metrics,sensor_id=2 cpu=33.1
cpu_metrics,sensor_id=3 cpu=88.9
LP
```

Verify points exist with Flux:

```
docker exec -i influxdb influx query --org example-org --raw --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN" <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "cpu_metrics")
  |> limit(n: 10)
FLUX
```

4 - InfluxDB: create v1-compat credentials (for Grafana InfluxQL)

Grafana 11.2 shows the InfluxQL (legacy) datasource form, so create v1 credentials that Grafana can use:

Get the bucket ID:

```
docker exec -i influxdb influx bucket list --org example-org --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"
# note id for example-bucket
```

Create v1 auth (replace `<BUCKET_ID>`):

```
docker exec -i influxdb influx v1 auth create \
  --org example-org \
  --username admin \
  --password admin123 \
  --read-bucket <BUCKET_ID> \
  --write-bucket <BUCKET_ID> \
  --token "$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"
```

Verify:

```
docker exec -i influxdb influx v1 auth list --org example-org --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"
```

Now Grafana can use `admin` / `admin123` for InfluxQL.

5 - Grafana: add datasources

5.1 Add TimescaleDB (Postgres) datasource

1. Grafana → **Connections** → **Data sources** → **Add data source** → **PostgreSQL**
2. Fill:
    - Host: `timescaledb:5432`
    - Database: `metricsdb`
    - User: `admin`
    - Password: `admin123`
    - SSL: disable (demo)
3. Click **Save & test** → should be successful.

5.2 Add InfluxDB datasource (InfluxQL mode)

1. Grafana → **Connections** → **Data sources** → **Add data source** → **InfluxDB**
2. Since your Grafana shows InfluxQL UI, fill:
    - URL: `http://influxdb:8086`
    - Database: `example-bucket`
    - User: `admin`
    - Password: `admin123`
3. Click **Save & test** → *Datasource is working*.

6 - Build dashboards (step-by-step) and explanation

You will generally build two panels per sensor: (A) **hourly aggregate** (fast, uses pre-aggregated data / server-side aggregate) and (B) **raw recent points** (live streaming view).

6.1 TimescaleDB panel (recommended for fast hourly queries)

Use the continuous aggregate you created.

1. Grafana → **Dashboards** → **+ New dashboard** → **Add new panel**.
2. Select data source: your PostgreSQL (TimescaleDB) datasource.
3. SQL (paste into the SQL editor):

```
SELECT
  bucket AS "time",
  avg_cpu AS "Avg CPU",
  max_cpu AS "Max CPU"
FROM hourly_cpu_usage
WHERE sensor_id = 1
  AND $__timeFilter(bucket)
ORDER BY bucket ASC;
```

6.2 InfluxDB panel (InfluxQL hourly mean) - if you prefer querying Influx directly

InfluxQL hourly mean:

```
SELECT mean("cpu") AS "avg_cpu"
FROM "cpu_metrics"
WHERE $timeFilter
GROUP BY time(1h), "sensor_id" fill(null)
```

Steps:

1. New dashboard → Add panel → choose your **InfluxDB** datasource (the InfluxQL one).
2. In the query editor select **InfluxQL** and paste the query above.

Why this query? Break down:

- `SELECT mean("cpu")` — computes average CPU per group/window.
- `WHERE $timeFilter` — Grafana expands `$timeFilter` to the dashboard time range (e.g., `time >= now() - 24h`).
- `GROUP BY time(1h), "sensor_id"` — groups points into 1-hour time windows and separately for each `sensor_id` (so you get one series per sensor).
- `fill(null)` — prevents line interpolation across buckets with no data (shows gaps rather than misleading interpolation).

When to use InfluxQL query vs Timescale continuous aggregate?

- Use **Timescale continuous aggregate** for performance and if you want SQL/centralized view in Postgres.

- Use **InfluxQL/Flux** when your raw data sits in InfluxDB and you prefer to compute aggregates on-the-fly or if you already have existing Influx dashboards.

6.3 Raw live panel (show immediate points)

If you want to see the most recent points (not aggregated), create a panel that queries the raw measurement:

TimescaleDB (raw recent):

```
SELECT time AS "time", cpu AS "CPU"
FROM cpu_metrics
WHERE sensor_id = 1
  AND $__timeFilter(time)
ORDER BY time ASC
LIMIT 1000;
```

InfluxQL (raw recent):

```
SELECT "cpu"
FROM "cpu_metrics"
WHERE $timeFilter AND "sensor_id"='1'
```

Set refresh to `5s` — this shows live values as they arrive.

7 - Grafana alert: CPU > 90%

Use Grafana-managed alerts (latest unified alerting).

Alert on TimescaleDB raw data (recommended for immediate detection)

1. Grafana → **Alerting** → **Alert rules** → **+ New alert rule**.
2. Query (Postgres) — single value:

```
SELECT MAX(cpu) AS value
FROM cpu_metrics
WHERE sensor_id = 1
  AND time > NOW() - INTERVAL '2 minutes';
```

8 - Cleanup

TimescaleDB:

Open psql:

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

SELECT remove_continuous_aggregate_policy('hourly_cpu_usage');
DROP MATERIALIZED VIEW IF EXISTS hourly_cpu_usage;
DROP TABLE IF EXISTS cpu_metrics;
```

InfluxDB:

```
# list v1 auths, then delete if you created one
docker exec -i influxdb influx v1 auth list --org example-org --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"
docker exec -i influxdb influx v1 auth delete --id <AUTH_ID> --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"
# delete measurements (careful)
docker exec -i influxdb influx delete --org example-org --bucket example-bucket \
  --start 1970-01-01T00:00:00Z --stop $(date -u +"%Y-%m-%dT%H:%M:%SZ") \
  --predicate '_measurement="cpu_metrics"' --token
"$DOCKER_INFLUXDB_INIT_ADMIN_TOKEN"
```

Grafana:

- Remove datasources, dashboards, and alert rules via the UI.

Full teardown:

```
docker compose down -v
```