Event-Driven Task Scheduler & Processing

Problem Tackled:

production-style .NET platform that lets clients create tasks via a secure Web API; tasks are stored in SQL, broadcast as events over Redis Pub/Sub, and processed by a background Worker with full metrics, centralized logging, dashboards, and systemd services.

Architecture

- **Clients (Postman / curl / Angular app):**
  - send HTTPS calls with JWT tokens.
- **TaskHub.API (ASP.NET Core Web API)**
  - validates JWT + claims -> writes to **SQL** (SQLite locally / SQL Server in prod) -> publishes an **event** (JSON) to **Redis** -> emits **Prometheus** metrics -> logs to **Serilog** (files in /var/log/taskhub).
- **Redis (Pub/Sub)**
  - acts like a loudspeaker: when API publishes “TaskCreated”, any subscriber hears it.
- **TaskHub.Worker (Background service)**
  - subscribes to the Redis channel -> consumes events -> loads the task from SQL -> executes business logic (simulate processing) -> updates status -> logs + metrics.
- **Observability stack**
  - **Serilog** writes structured logs (compact JSON) to /var/log/taskhub/*.
  - **Promtail** tails those files and pushes to **Loki**.
  - **Grafana** reads from **Loki** (logs) + **Prometheus** (metrics) and renders dashboards.
  - **Cloudflare Tunnel** exposes Grafana on a subdomain.
- **Host/OS Layer**
  - Both API and Worker are published as **self-contained single files** and managed by **systemd** (taskhub-api.service, taskhub-worker.service) for auto-start and restart.

Domain Model

- **TaskItem**
  - Id (GUID / int), Title (string), Status (enum: Pending, Processing, Done, Failed), CreatedAt (UTC), UpdatedAt (UTC), Attempts (int).
- **Audit**
  - Id, At (UTC), Message.

Data Flow Scenarios

- **Create Task**
  - **Client → API**: POST /tasks with JWT in Authorization: Bearer <token>, body { "title": "Resize images" }.
  - **Auth**: API validates the token signature, issuer/audience, and required claims (e.g., scope: tasks.write or role Task.Creator).
  - **Persist**: API inserts TaskItem → Status=Pending, CreatedAt=UtcNow.
  - **Publish Event**: API serializes a compact JSON: [<sample> it is not real project json]

{ "type":"TaskCreated", "id":123, "title":"Resize images", "at":"2025-09-28T22:17:17Z" }

---

and publishes to Redis channel

  - **Metrics + Logs**: [these are only sample metrics not real data I have worked on]
    - Increments taskhub_api_tasks_created_total.
    - Serilog info: TaskCreated {Id, Title} (structured).
    - **Response**: 201 Created with the Task resource.
- **Worker Consumption & Processing**
  - **Subscribe**: Worker is subscribed to Redis taskhub:events.
  - **On Message**: When "TaskCreated" arrives, it:
    - Parses payload, loads Task from SQL by Id.
    - Moves Status -> Processing; increments Attempts.
    - Executes "work" (simulated queue job like delay, transformation).
    - On success: Status -> Done. On exception: Status -> Failed and logs error.
  - **Idempotency Guard**:
    - If the worker double-sees an event, it checks current Status; if already Done/Processing with a newer timestamp, it no-ops. (This avoids duplicate execution.)
  - **Metrics + Logs**: [these are only sample metrics not real data I have worked on]
    - taskhub_worker_tasks_processed_total{result="success|failed"}.
    - taskhub_worker_task_duration_seconds (histogram).
    - Serilog: TaskProcessed {Id, DurationMs, Result}.
  - **Heartbeat / Liveness:**
    - Worker writes an **Audit** "heartbeat" row every 10s and logs Audit heartbeat saved.
    - API exposes:
      - GET /health → liveness, returns "ok".
      - GET /metrics → Prometheus metrics exposition.

API Surface

- POST /tasks → Create new task (requires tasks.write).
- GET /tasks → List tasks (requires tasks.read).
- GET /health → "ok".
- GET /metrics → Prometheus exposition (scraped by Prometheus or the node_exporter you configured).

Security & Authentication (JWT & Claims)

- **Token Validation**
  - **ASP.NET Core** Authentication middleware:
    - Validates issuer, audience, lifetime.
    - Validates signature (symmetric HMAC secret or asymmetric public key).
  - Config via environment variables (no secrets in code):
    - Jwt__Authority, Jwt__Audience, Jwt__IssuerSigningKey (or Jwt__MetadataAddress for OIDC).
  - **CORS**: allow your SPA or test origins.
  - **HTTPS**: When exposed publicly (Cloudflare), SSL termination at the edge; API can remain internal.
  - **Rate Limiting**: ASP.NET rate limiting middleware to protect POST /tasks.
- **Authorization**
  - **Policy-based**:
    - RequireClaim("scope", "tasks.read") for reads.
    - RequireClaim("scope", "tasks.write") for writes.
  - **Role-based:**
    - RequireRole("Admin", "Operator") for management endpoints.

Messaging (Redis Pub/Sub | Azure Service Bus)

- **Channel**: taskhub:events.
- **Payload**: compact JSON with type, id, title, at.
- **Password**: set via env var.
- **Resilience**:
  - Worker reconnects to Redis|Azure service bus on transient errors.
  - Idempotent handler to handle duplicate deliveries.

Persistence (SQL)

- **Local**: SQLite file `/var/lib/taskhub/taskhub.db` for quick setup.
- **Production**: SQL Server / Azure SQL.
- **EF Core:**
  - DbContext with Tasks and Audit sets.
  - EnsureCreated() for local demo; migrations for real environments.

- **Constraints / Indexes**:
    - PK on Id, index on Status, optional index on CreatedAt.

Observability

- **Logging**
    - **Serilog** in both API & Worker:
        - Sinks: rolling files in /var/log/taskhub/apiYYYYMMDD.log and /var/log/taskhub/workerYYYYMMDD.log.
        - Template: compact JSON for easy parsing.
        - Properties: app=taskhub-api|taskhub-worker, plus domain fields.
- **Metrics**
    - **Prometheus:**
        - API exposes /metrics (standard ASP.NET metrics + our counters/histograms).
        - Worker can expose its own /metrics if hosted as Kestrel; or push via a pushgateway; for simplicity we instrument logs + heartbeats and keep API metrics.
- **Logs → Loki → Grafana**
    - **Promtail** config tails:
        - /var/log/taskhub/api*.log with labels job=taskhub, app=taskhub-api
        - /var/log/taskhub/worker*.log with labels job=taskhub, app=taskhub-worker
    - Grafana Dashboards:
        - Logs panel filtered by {app="taskhub-worker"} with fields @m, @t, lvl, TaskId.
        - Metrics panels: request duration histogram, tasks processed rate, failures, heartbeats.
- **Exposure View-only:**
    - **Cloudflare Tunnel** to expose Grafana on a subdomain (read-only/anonymous viewer).
    - **Grafana:**
        - GF_AUTH_ANONYMOUS_ENABLED=true, role Viewer.
        - GF_SERVER_ROOT_URL set to the public URL.

Deployment & Operations

- **Publish API & Worker.**
- **systemd Services**
    - Auto restart on reboot and set appropriate permissions.
    - Services are up when reboot automatic once daemons are set.
    - Helps with instant logs about on execute via journalctl.

Hardening & Production Polish

- **CORS**: restrict to your frontend origins.
- **Rate limiting** on write endpoints.
- **Retry policies** (Polly) around DB and Redis connect.
- **Idempotency keys** on POST /tasks to protect against client retries.
- **DLQ** (future): if a task fails N times, mark Failed and optionally publish TaskFailed for an alert pipeline.
- **RBAC**: authorize admin endpoints (e.g., retry/cancel) with Admin role.
- **Backpressure**: if worker lags, you could convert from Pub/Sub to a queue with explicit ack (Azure Service Bus, Redis Streams).

Azure Mapping

- **App Services**: host API & Worker (Worker as a WebJob or background worker in a hosted app service).
- **Azure Functions**: alternative worker triggered by Service Bus or Event Grid.
- **Azure Service Bus**: Redis Pub/Sub with Topic/Subscription for durability.
- **Azure SQL** / **Cosmos DB**: drop-in persistence change.
- **Azure Monitor / Log Analytics**: push logs; query with Kusto (KQL) in Kusto Explorer.
- **Azure Application Insights**: add distributed tracing and request/dep. telemetry.

Runbook

- **Services verification:** systemctl status
- **Health check:** /health -> “ok”
- **Task Creation:** curl -H "Authorization: Bearer <JWT>" -H "Content-Type: application/json" -d '{"title":"Demo Task"}' “supposedURL/tasks”
- **Worker logs:** tail worker.logs to creations & processed.
- **Metrics:** curl url/metrics
- **Grafana:** log panels and dashboard with metrics.

Conclusion:

A secure, observable, event-driven task platform in .NET: API writes to SQL and signals work via Redis; a Worker consumes and processes tasks; logs and metrics flow into Grafana/Loki/Prometheus; everything is productionized with systemd and environment-based configuration, and it maps 1-to-1 onto Azure service.