Cloud Native Observability.

A horizontally scalable, highly available backend for ingesting telemetry (metrics + logs), storing time-series data, firing alerts, and serving queries to a Realtime dashboard.

High-level architecture (backend-focused)

- **Ingress / API Gateway**
  - TLS termination, rate limit, auth, routing to services
- **Ingestion Layer**
  - **Ingest API** (HTTP/gRPC) accepts metrics & logs
  - **Buffering**: Kafka (or Redis Streams) between ingesters and writers
  - **Ingesters**: stateless workers that validate, batch, and forward to storage
- **Persistence**
  - **TimescaleDB** (Postgres + Timescale) for metrics (hypertables + continuous aggregates)
  - **Postgres (vanilla)** for metadata, users, alert rules, dashboards
  - **Elasticsearch or ClickHouse** for high-volume logs & full-text search (optional/hybrid)
- **Processing**
  - **Aggregator Service**: builds rollups / aggregates, writes to continuous aggregates or materialized views
  - **Alerting Service**: subscribes to aggregated metrics, evaluates rules, triggers notifications
- **Query API / Backend for Dashboard**
  - Query endpoints for timeseries, logs, metadata; supports pagination, filters, bucketed aggregation
  - WebSocket / SSE for real-time push (alerts, live metrics)
- **Auth & User Service**
  - JWT + refresh tokens / API keys for services and users
  - Role-based access control (RBAC)
- **Admin / Management**
  - Schema migrations, telemetry of the platform itself
- **Observability of the observability**
  - Prometheus metrics + traces (OTel) for all services

Technology choices (recommended)

- Language: **Go** or **.NET (C#)** or **Node (TypeScript)** for microservices (Go preferred for lightweight concurrency & shipping single binaries)
- Message broker: **Kafka** (production) / **Redis Streams** (simpler)
- Metrics DB: **TimescaleDB** (Postgres + Timescale)
- Logs DB (optional): **Elasticsearch** or **ClickHouse**
- API Gateway: **Traefik** / **NGINX Ingress** on Kubernetes
- Container orchestration: **Kubernetes**
- IaC: **Terraform** (cloud) + **Helm** or kustomize (cluster)
- CI/CD: **GitHub Actions** (or Azure DevOps/GitLab CI)
- Schema migrations: **Flyway** / **golang-migrate** / **EF Core Migrations**
- Secrets: **Vault** / cloud KMS
- Telemetry: **OpenTelemetry** + **Prometheus** + **Grafana**
- Authentication: **OAuth2 / JWT** (or integrate with identity provider)

Services & responsibilities (detailed)

1. **api-gateway**
   - Accepts all public traffic: /v1/ingest, /v1/query, /v1/auth, /v1/logs
   - Rate-limits per API key, basic auth checks, TLS
2. **ingest-service**
   - HTTP/gRPC endpoints for clients/agents
   - Validates payloads, enriches with metadata, writes to Kafka/topic
   - Emits internal metrics
3. **ingest-writer(s)**
   - Consumer(s) of Kafka topic
   - Batch writes to TimescaleDB (metrics) and Elasticsearch/ClickHouse (logs)
   - Idempotency and retry handling
4. **aggregator / rollup-service**
   - Creates continuous aggregates or writes downsampled series
   - Runs scheduled jobs or stream-processing to compute 1m/5m/1h rollups
5. **query-service**
   - Exposes REST GraphQL/gRPC endpoints for dashboard queries
   - Caches frequent queries (Redis) and supports pagination
6. **alerting-service**
   - Continuously evaluates alert rules (pull from DB or subscribe to processed streams)
   - Supports rate-limited notifications (email, webhook, Slack)
   - Records alert history in metadata DB
7. **auth-service**
   - Issues & validates JWTs, issues API keys, rate-limit keys
   - RBAC lookups in metadata DB
8. **user-mgmt & config-service**
   - Dashboard configs, users, projects, environments, alert rules stored in Postgres
9. **ingest-agent (optional)**
   - Lightweight agent to run on VMs/containers to collect system metrics & ship to ingest endpoint
10. **admin-service / operator tools**
    - Tasks: migrations, backup restores, cluster management

API design (core endpoints)

Use REST or gRPC. Example REST endpoints:

Auth

- POST /v1/auth/login → returns JWT
- POST /v1/auth/apikey → create API key (service use)

Ingestion

- POST /v1/ingest/metrics
  payload: { time, service, instance_id, metric_name, value, labels }
- POST /v1/ingest/logs
  payload: { time, service, instance_id, level, message, labels, json }

Query

- GET /v1/query/metrics?service=&metric=&from=&to=&step= → returns time-buckets
- POST /v1/query/logs → search query with filters, pagination
- GET /v1/query/series → list series metadata (for dashboard selectors)

Alerts / Rules

- GET /v1/alerts
- POST /v1/alerts → create rule { name, expression, window, severity, notify }
- GET /v1/alerts/history?service=&from=&to=

Admin

- GET /v1/health → platform health
- GET /v1/metrics/platform → internal metrics

Realtime

- WebSocket endpoint /v1/ws for push updates & alerts

Security:

- All endpoints require API key or bearer token
- Admin endpoints require elevated role

Data models & schemas (key tables)

Timescale metrics table (hypertable)

```
CREATE TABLE metrics (
  time       TIMESTAMPTZ   NOT NULL,
  org_id     UUID          NOT NULL,
  service    TEXT          NOT NULL,
  instance_id TEXT,
  metric_name TEXT         NOT NULL,
  value      DOUBLE PRECISION NOT NULL,
  labels     JSONB,
  PRIMARY KEY (time, org_id, service, metric_name, instance_id)
);
SELECT create_hypertable('metrics', 'time', chunk_time_interval => INTERVAL '1 day');
```

Metadata (Postgres)

```
-- users
CREATE TABLE users (
  id UUID PRIMARY KEY,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT,
  role TEXT,
  created_at TIMESTAMPTZ DEFAULT now()
);


-- api_keys
CREATE TABLE api_keys ( id UUID PRIMARY KEY, user_id UUID REFERENCES users(id), key TEXT,
scopes TEXT[], created_at TIMESTAMPTZ );


-- alert_rules
CREATE TABLE alert_rules (
  id UUID PRIMARY KEY,
  org_id UUID,
  name TEXT,
  expression TEXT, -- e.g., "avg(cpu_usage) > 80"
  window_interval INTERVAL,
  severity TEXT,
  notify JSONB,
  created_by UUID,
  enabled BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

```
-- alert_history
CREATE TABLE alert_history ( id UUID PRIMARY KEY, alert_id UUID REFERENCES alert_rules(id), triggered_at TIMESTAMPTZ, state TEXT, details JSONB );
```

Logs (Elasticsearch mapping) — or ClickHouse table if using CH

Store logs as JSON with indexed fields: time, service, instance_id, level, message, plus an analyzed message field for full-text.

Ingestion patterns & batching

- Clients should send batched payloads (100–500 datapoints) to reduce overhead.
- Ingest API writes to Kafka topic metrics.incoming and logs.incoming.
- Writers consume batches, convert to COPY/INSERT batches for TimescaleDB (COPY is fastest).
- Use connection pooler (pgbouncer) between writers and Postgres.

Scaling & HA patterns

- **Stateless services**: scale via replica count in K8s HPA (CPU or custom metrics).
- **Kafka**: partition by org_id or service to parallelize writers.
- **Timescale**: use vertical scaling + partitioning; for huge scale, consider multi-node Timescale.
- **Read replicas**: run Postgres replicas for query-service to protect writes.
- **Health checks**: readiness & liveness probes for each pod.
- **Leader election**: for scheduled aggregator jobs (use Kubernetes leader election or distributed lock via Redis).

Durability & retention

- Use chunk retention policies in Timescale (e.g., drop raw 90d, keep rollups longer).
- Compress older chunks (Timescale compression).
- Archive older raw data to S3 (Parquet) for cold storage and restore if needed.
- Snapshot Elasticsearch indices or export ClickHouse.

CI/CD & release strategy (backend)

Repository layout: mono-repo or services per repo. Either OK; recommend mono-repo for a portfolio.

CI pipeline (GitHub Actions)

- push to main:
    1. Lint & unit tests for service changed
    2. Build Docker image (service-specific)
    3. Run integration tests in lightweight k3s / Docker Compose
    4. Publish image to container registry (GitHub Packages / Docker Hub / ACR)
    5. Create Helm chart update / k8s manifests
    6. Deploy to staging via helm (automated)
    7. Run smoke tests
    8. Deploy to production with manual approval (or automated canary)

CD features

- Canary / blue-green deployments (k8s + helm + service mesh optional)
- Rollbacks on health-check failures
- DB migration job in pipeline (reviews + migrations run in pre-prod automatically)

Schema migrations

- Migrations run as Kubernetes Job with migrate binary at deploy time; fail deployment if migration fails.

Observability & testing of the platform

- **Tracing**: OpenTelemetry traces captured for request flows (ingest → writer → DB)
- **Metrics**: Each service exposes Prometheus metrics; configure a Prom server for platform.
- **Logs**: Centralized logs (Elasticsearch/ClickHouse)
- **SLOs & Alerts**: Monitor ingest_latency, write_failures, db_connections, kafka_lag, api_error_rate.

Testing:

- Unit tests per service
- Integration tests: start local TimescaleDB + Kafka + services in GitHub Actions using Docker Compose / testcontainers
- Load tests: k6 or Vegeta to simulate ingestion patterns with burstiness and sustained load
- Chaos tests: kill pods / introduce latency in staging

Security & secrets

- TLS everywhere (ingest endpoints & internal)
- RBAC in Kubernetes
- Use vault / secrets manager for DB passwords, API keys
- Rate-limit unauthenticated or misbehaving clients
- Input validation & schema strictness to avoid injection

Dev & local environment

- docker-compose.yml for local dev:
    - TimescaleDB, Postgres (metadata), Kafka (or Redis), Zookeeper (if Kafka), Elasticsearch (optional), pgbouncer, and local versions of services
- Scripts to seed sample data + example ingestion client

Backup & restore

- Postgres/Timescale: base backups + WAL archiving, test restores
- Elasticsearch / ClickHouse snapshots to object storage
- Regular recovery drills documented in repo

Deliverables & phased roadmap (milestones)

I. **Phase 0 — Scaffolding & infra (MVP backend)**

- Repo & service skeletons: ingest-service, ingest-writer, query-service, auth-service, alerting-service
- Docker Compose for local dev (TimescaleDB + Kafka / Redis)
- Basic API endpoints: /v1/ingest/metrics, /v1/query/metrics
- Timescale metrics hypertable + basic inserts
- README + architecture diagram
- Deliverable: demo ingest → store → query flow

II. **Phase 1 — Core functionality**

- Batched writes + COPY optimization
- Kafka buffering + idempotent consumers
- Query API with aggregation + cache (Redis)
- Simple React dashboard integration (real time via WebSockets)
- CI: unit tests + Docker build + push
- Deliverable: functional dashboard showing live metrics for sample services

III. **Phase 2 — Alerting & Aggregation**

- Continuous aggregates / rollups (1m/5m/1h)
- Alert rule CRUD + evaluator + notification hooks (webhook/email)
- RBAC + API key auth
- Deliverable: alerts triggered by synthetic load, alert history visible in UI

IV. **Phase 3 — Scale & Resilience**

- Kubernetes manifests / Helm charts
- Production-ready DB config (pgbouncer, backups, replica)
- Autoscaling rules + Kafka partitioning tuning
- Load testing & performance benchmarks
- Deliverable: deploy to cloud cluster (AKS/EKS/GKE) with docs

V. **Phase 4 — Logs, Search & Long-term storage**

- Integrate Elasticsearch/ClickHouse for logs
- Implement log ingestion & search endpoints
- Cold storage -> S3 + Parquet
- Deliverable: searchable logs & retention policies enforceable

VI. **Phase 5 — Harden & polish**

- Observability for platform (Prom + Grafana dashboards)
- Security review + secrets management
- Canary deployments, automatic rollbacks
- Production runbook & SLA documentation
- Deliverable: portfolio-ready repo + case study README and diagrams

Repo structure suggestion (mono-repo)

```
/repo-root
 /docs
  architecture.md
  runbooks.md
 /infra
  k8s/ (helm charts)
  terraform/
  docker-compose.yml
 /services
  /ingest-service
   main.go
   Dockerfile
   /migrations
  /ingest-writer
  /query-service
  /alerting-service
  /auth-service
 /clients
  /agent (example agent)
 /scripts
  seed-data.sh
 /ci
  workflows/
 README.md
```

Example CI snippet (GitHub Actions - build & test)

```
name: CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Go
        uses: actions/setup-go@v4
        with: { go-version: '1.20' }
      - name: Build services
        run: |
          cd services/ingest-service && go build -o ingest
          cd ../ingest-writer && go build -o writer
      - name: Run unit tests
        run: |
          go test ./...
      - name: Build Docker images
        run: |
          docker build -t ghcr.io/${{ github.repository }}/ingest-service:pr-${{ github.run_id }}
services/ingest-service
      - name: Push images
        uses: docker/login-action@v2
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GHCR_TOKEN }}
```

Metrics & SLO ideas (for platform)

- **Ingest latency (p95) < 200ms**
- **Write success rate > 99.9%**
- **Query latency (p95) < 300ms for 1m window**
- **Kafka consumer lag < 1000 messages**

Security / Compliance notes

- Obfuscate/omit sensitive data from logs
- GDPR: retention & deletion APIs per org id
- Audit logs of admin actions