NETCELL EDA PLATFORM

Event-Driven AI Platform Deployment Map

NETCELL-EDA Architecture

Deployment Map

Contacts Domain —Deployment Map (TenantA)

NETCELL-EDA Architecture

Deployment Map

1. Microservices (Kubernetes Pods)

A. Contact Events Processor

Consumes: contacts-events topic

Group: contact-processor-group

Replicas: 4 pods

Responsibility:

- Feature building
- Feature store updates
- Inference call
- Profile update logic

B. Profiling Service

Replicas: 3 pods

Responsibility:

- Feature building
- Feature store updates
- Inference call
- Profile update logic

NETCELL-EDA Architecture

Deployment Map

C. Inference Service

Replicas: 3 pods

Responsibility:

- Load models from Model Registry
- Score features
- Return risk/churn/engagement

D. Feature Store Online API

Replicas: 2 pods

Responsibility:

- Read/write online features
- Low-latency access

E. Model Registry API

Replicas: 2 pods

Responsibility:

- Serve model metadata
- Provide model files
- Track versions

F. Config Service

Replicas: 2 pods

Responsibility:

- Tenant-specific rules
- Thresholds
- Feature definitions

NETCELL-EDA Architecture

Deployment Map

2. Kafka (Cloudera Streams)

Topic: contacts-events

Partitions: 8

Key: ContactId

Tenant field: TenantId = "tenantA"

Consumer group: contact-processor-group

4 pods → Kafka assigns partitions evenly

Ordering guaranteed per ContactId

3. Document DB (Profiles)

Collection: ContactProfiles

Key: TenantId + ContactId

Storage:

MongoDB / Cosmos DB / HBase (Cloudera)

Updated by:

Profiling Service

Contact Events Processor

4. Feature Store

Online Feature Store

Redis / MongoDB / Cassandra

Updated on every event

Read by Inference Service

Offline Feature Store

Cloudera Iceberg tables

Daily batch jobs compute rolling windows

Used for model training

5. Model Registry (Cloudera Storage)

Metadata table: model_registry

Model files: stored in Cloudera object storage

Used by:

Inference Service

Training pipeline

NETCELL-EDA Architecture

Deployment Map

6. Training Pipeline (Cloudera Spark / Flink)

Daily job:

Reads offline features

Trains models

Writes new versions to Model Registry

Weekly job:

Recomputes labels (churn, risk)

Updates training datasets

7. Tenant Isolation

TenantA isolation rules:

TenantId = "tenantA" in every event

Profiles stored under TenantA namespace

Feature Store keys include TenantId

Model Registry supports per-tenant models

Config Service returns TenantA-specific rules

Ranger policies restrict access to TenantA data

8. Dashboard Layer

Dashboard API

Reads profiles, Reads scores, Reads , ggregates

Visualization

Contact timeline, Risk heatmap, Churn, prediction

Engagement score

NETCELL-EDA Architecture

Deployment Map

9. Full Flow Example (ContactStateChanged for TenantA)

Step-by-step:

Event arrives

- ContactStateChanged → contacts-events topic
- Key = ContactId
- TenantId = "tenantA"

Contact Events Processor consumes event

- Validates schema
- Loads profile
- Builds features

Profiling Service

- Updates online features
- Calls Inference Service

Inference Service

- Loads active models for TenantA
- Scores features
- Returns risk/churn/engagement

Profiling Service updates profile

- Writes scores
- Writes state history
- Upserts profile

Document DB stores updated profile

Offline Feature Store updated nightly

Training pipeline retrains models weekly

Dashboard shows updated scores

This is the complete operational picture for one domain and one tenant.

NETCELL-EDA Architecture

Deployment Map

Contacts Domain —Deployment Map (Yamel)

NETCELL-EDA Architecture
Deployment Map

A. Contact Events Processor

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: contact-events-processor
spec:
  replicas: 4
  selector:
    matchLabels:
      app: contact-events-processor
  template:
    metadata:
      labels:
        app: contact-events-processor
spec:
    containers:
      - name: processor
        image: netcell/contact-processor:1.0.0
        env:
          - name: KAFKA_BOOTSTRAP_SERVERS
            value: kafka:9092
          - name: TOPIC_CONTACTS
            value: contacts-events
          - name: CONFIG_SERVICE_URL
            value: http://config-service
        resources:
          requests:
            cpu: "250m"
            memory: "256Mi"
          limits:
            cpu: "1"
            memory: "512Mi"
```

NETCELL-EDA Architecture
Deployment Map

B. Profiling Service

```
yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: profiling-service
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: profiling
          image: netcell/profiling-service:1.0.0
          env:
            - name: DOCUMENT_DB_URL
            - name: FEATURE_STORE_URL
            - name: INFERENCE_SERVICE_URL
```

NETCELL-EDA Architecture
Deployment Map

D. Shared services (Config, Model Registry, Feature Store)

Each has similar Deployment + Service objects, with:

replicas

env vars for DB/storage

resource limits

NETCELL-EDA Architecture

Deployment Map

2. ContactStateChang From event to profile update, step by step.

ed event end-to-end code flow

Step 1 — Event produced

Integration layer or app emits:

json

```
{
  "EventId": "...",
  "EventType": "ContactStateChanged",
  "EventVersion": 1,
  "TenantId": "tenantA",
  "SourceSystem": "Netcell-Core",
  "OccurredAt": "2026-08-04T19:40:00Z",
  "TraceId": "...",
"Data": {
    "ContactId": 12345,
    "AccountId": 6789,
    "OldState": "New",
    "NewState": "Active",
    "Reason": "First login",
    "Channel": "Web",
    "Agent": "System"
  }}
```

Sent to contacts-events with key = ContactId.

Step 2 — Contact Events Processor consumes

Pseudo-flow:

```
pseudo
event = kafkaConsumer.poll()
 validate(event, schemaRegistry)
 tenantConfig = configService.get(event.TenantId, "contact")
 profile = documentDb.getProfile(event.TenantId, event.Data.ContactId)
 features = featureBuilder.build(profile, event, tenantConfig)
 scores = inferenceClient.scoreContact(event.TenantId, event.Data.ContactId, features)
 updatedProfile = profileUpdater.update(profile, event, scores, tenantConfig)
 documentDb.upsert(updatedProfile)
 featureStoreOnline.upsert(event.TenantId, event.Data.ContactId, features)
```

NETCELL-EDA Architecture
Deployment Map

3. Multi-tenant routing logic

How TenantId shapes everything.

Core rule:

Every event, profile, feature, and model is **tagged with TenantId**.

Routing points:

Kafka:

Shared topic contacts-events

Consumers read all events but process per TenantId.

Config Service:

GET /config/{tenantId}/contact → returns tenant-specific rules.

Inference Service:

Chooses model set based on TenantId:

global model or tenant-specific model.

Document DB:

Profiles keyed by TenantId + ContactId.

Feature Store:

Features keyed by TenantId + ContactId.

Example routing in processor:

pseudo

```
event = kafkaConsumer.poll()

tenantId = event.TenantId

tenantConfig = configService.get(tenantId, "contact")

modelSet = modelRegistry.getActiveModels("contact", tenantId)

scores = inferenceClient.scoreWithModels(tenantId, features, modelSet)

ProfileUpdater.applyTenantRules(updatedProfile, scores, tenantConfig)
```

NETCELL-EDA Architecture
Deployment Map

4. AI model lifecycle

From idea → training → production → retirement.

Stage 1 — Design

Define target:

- Risk, Churn, Engagement.

Define labels:

- e.g. churn = no activity for 90 days.

Stage 2 — Feature engineering

Use offline Feature Store:

- build feature sets per contact.

Store in training tables.

Stage 3 — Training

Run training jobs (Spark/Flink/Python):

- train models (XGBoost, etc.).
- evaluate metrics (AUC, F1, etc.).

Stage 4 — Registration

Register model in Model Registry:

- type, version, metrics, path, tenant scope.

Status = validated.

Stage 5 — Promotion to production

Ops or you decide:

- set Status = production.

Inference Service reloads active models.

Stage 6 — Monitoring

Track:

- performance drift,
- data drift, error rates.

Stage 7 — Retraining

Periodic retraining:

- new data → new models.

Old models set to deprecated.

NETCELL-EDA Architecture
Deployment Map

5. Cloudera integration blueprint

How Netcell EDA platform sits on top of Cloudera.

Streaming (Kafka)

Use Cloudera Kafka for:

- contacts-events, accounts-events, etc.

Integrate with:

- Schema Registry
- Ranger (security)

Data Lake (Iceberg/Hive)

Store:

- raw events (append-only tables)
- offline features (contact_features_daily)
- training datasets.

Compute (Spark/Flink)

Use Cloudera Spark/Flink for:

- feature generation (batch)
- model training
- backfills.

Security & Governance

Ranger:

- controls access to topics and tables per tenant.

Atlas:

- tracks lineage of features and models.

Storage

Model files:

- stored in Cloudera object storage (S3/HDFS).

Feature Store offline:

- Iceberg tables.

microservices (Kubernetes)

Run on:

- Cloudera's Kubernetes or external K8s cluster.

Connect to:

- Kafka
- Data Lake
- Object storage
- Ranger/Atlas APIs

NETCELL-EDA Architecture

Deployment Map

Thank you