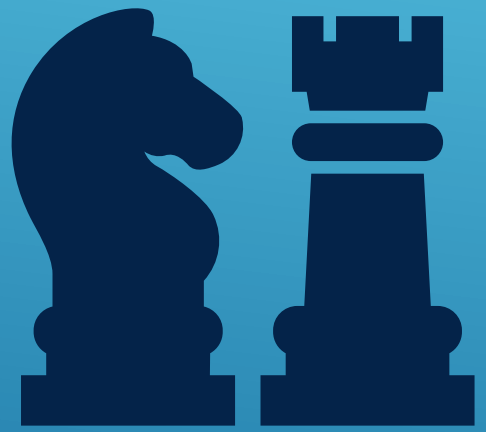

NETCELL EDA PLATFORM

Event-Driven AI Platform Map Strategy

NETCELL-EDA Architecture Microservices Map (Strategy)

1. Kafka topic strategy

Define core

Domains, Contacts, Accounts, Billing, Security

Name topics per domain

contacts-events, accounts-events, billing-events, security-events

Use keys for ordering

Contacts: key = ContactId

Accounts: key = AccountId

Billing: key = InvoiceId or AccountId

Security: key = UserId or AccountId

This guarantees per-entity ordering.

Partitioning strategy

Start with **4–8 partitions** per topic.

Scale up as traffic grows.

Use **key-based partitioning** so all events for one entity go to the same partition.

Retention & compaction

Events topics: long retention (e.g. 30–90 days).

Audit topics: very long retention (e.g. 1–2 years).

Use **compaction** for some "state" topics if needed.

NETCELL-EDA Architecture Microservices Map (Strategy)

2. Tenant isolation model

Step 1:

Decide isolation level

You have two main options:

Per-tenant topics

tenantA.contacts-events, tenantB.contacts-events

Strong isolation, more topics.

Shared topics with TenantId in message

Single contacts-events topic

Filter by TenantId in consumers.

For large orgs, I'd lean toward:

Shared topics + strict security + TenantId in event.

Step 2:

Add TenantId to all events

Every event must include:

TenantId (string or int)

Step 3:

Tenant-aware consumers

Each microservice filters events by TenantId.

Some services may handle multiple tenants; others may be dedicated.

Step 4:

Governance

Use Cloudera Ranger (or similar) to:

control which tenants can access which topics

enforce read/write permissions

NETCELL-EDA Architecture Microservices Map (Strategy)

3. Event schema registry

Step 1:

Define canonical event format

All events share a base structure:

Base fields:

EventId, EventType, EventVersion, TenantId, SourceSystem, OccurredAt, TraceId

Payload:

Data (object specific to event type)

Step 2:

Use a schema registry (Avro/JSON)

Register each event type:

ContactStateChanged, ContactCreated, AccountCreated, InvoiceIssued, etc.

Step 3:

Versioning

Add EventVersion (e.g. 1, 2).

Never break old consumers—add fields, don't remove.

Step 4:

Validation

Producers validate events against schema before sending.

Consumers rely on schema registry to deserialize.

NETCELL-EDA Architecture
Microservices Map (Strategy)

4. Document DB profile structure

Step 1:

Define core profile entities

ContactProfile

AccountProfile

BillingProfile

SecurityProfile

Step 2: part-1

ContactProfile document

json

```
{
  "ContactId": 12345,
  "TenantId": "tenantA",
  "AccountId": 6789,
  "State": "Active",
  "StateHistory": [
    {
      "OldState": "New",
      "NewState": "Active",
      "OccurredAt": "2026-08-04T19:30:00Z",
      "Reason": "First login"
    }],
```

Step 2: part-2

```
"Scores": {
    "Risk": 0.12,
    "Engagement": 0.87,
    "Churn": 0.05
  },
  "LastEventAt": "2026-08-04T19:30:00Z",
  "Metadata": {
    "Channel": "Web",
    "Region": "IL"
  }
}
```

Step 3:

Upsert pattern

Key = TenantId + ContactId

For each event:

- Load profile
- Update fields (state, history, scores)
- Upsert back

NETCELL-EDA Architecture
Microservices Map (Strategy)

5. AI profiling pipeline

Step 1:

Feature extraction

From events, derive features like:

number of logins last 7 days

number of state changes

time since last activity

billing behavior

security incidents

Store features per entity.

Step 2:

Model training

Periodic batch jobs:

- read features from data lake / Document DB
- train models (risk, churn, engagement)
- store models in a Model Registry

Step 3:

Real-time inference

Profiling Service calls **Inference Service**:

- sends features for one entity
- receives scores (risk, churn, etc.)
- writes scores into profile document

Step 4:

Feedback loop

Analytics layer uses scores.

You can later:

- adjust models
- add new features
- refine thresholds

NETCELL-EDA Architecture
Microservices Map (Strategy)

1. Kafka topic strategy (for ContactStateChanged)

Domain: Contacts **Topic:** contacts-events **Key:** ContactId

Why this key: guarantees ordering of events per contact.

Partitions: start with 6–12 partitions; scale later.

Retention: e.g. 90 days for events, longer in data lake.

Producer behavior:

Every time contact state changes, producer sends a message to contacts-events with:

key = ContactId

value = canonical event JSON

NETCELL-EDA Architecture
Microservices Map (Strategy)

2. Tenant isolation model

We'll use **shared topics + TenantId in event**.

Event fields for tenancy:

TenantId: string or int

Consumers:

Filter by TenantId so each tenant's data is logically isolated.

Example:

json


```
{
  "EventId": "b3c9b8c2-1234-4567-8901-abcdefabcdef",
  "EventType": "ContactStateChanged",
  "EventVersion": 1,
  "TenantId": "tenantA",
  "SourceSystem": "Netcell-Core",
  "OccurredAt": "2026-08-04T19:40:00Z",
  "TraceId": "c1f2e3d4-5678-9876-5432-fedcba987654",
  "Data": {
    "ContactId": 12345,
    "AccountId": 6789,
    "OldState": "New",
    "NewState": "Active",
    "Reason": "First login",
    "Channel": "Web",
    "Agent": "System"
  }
}
```

NETCELL-EDA Architecture
Microservices Map (Strategy)

Event schema registry

Define a **canonical schema** for ContactStateChanged.

Base fields:

EventId (string, GUID)

EventType (string, "ContactStateChanged")

EventVersion (int)

TenantId (string)

SourceSystem (string)

OccurredAt (datetime, ISO 8601)

TraceId (string)

Data payload:

ContactId (int)

AccountId (int)

OldState (string)

NewState (string)

Reason (string)

Channel (string)

Agent (string)

You register this schema in the registry as:

Name: ContactStateChanged

Version: 1

NETCELL-EDA Architecture
Microservices Map (Strategy)

4.Document DB profile structure (ContactProfile)

Collection: ContactProfiles **Key:** TenantId + ContactId

Example document: part-1

```
json
{
  "TenantId": "tenantA",
  "ContactId": 12345,
  "AccountId": 6789,
  "State": "Active",
  "StateHistory": [
    {
      "OldState": "New",
      "NewState": "Active",
      "OccurredAt": "2026-08-04T19:40:00Z",
      "Reason": "First login",
      "Channel": "Web",
      "Agent": "System"
    }
  ],
```

Example document: part-2

```
"Scores": {
    "Risk": 0.12,
    "Engagement": 0.87,
    "Churn": 0.05
  },
  "LastEventAt": "2026-08-04T19:40:00Z",
  "Metadata": {
    "Region": "IL"
  }
}
```

Upsert logic for each ContactStateChanged **event:**

Find profile by TenantId + ContactId.

If not exists → create new profile.

Update:

- State = NewState
- append to StateHistory
- LastEventAt = OccurredAt

Optionally call AI to update Scores.

NETCELL-EDA Architecture
Microservices Map (Strategy)

5. AI profiling pipeline (for contacts)

Step 1: Feature extraction from events

From ContactStateChanged and other events, derive features like:

state_change_count_last_30_days

time_since_last_state_change

total_logins_last_7_days

billing_incidents_last_90_days

security_incidents_last_90_days

These features can be stored:

in the profile document (Features section), and/or

in a feature store in the data lake.

Step 2: Model training (batch)

Periodic job (e.g. nightly):

reads features for many contacts

trains models:

RiskScoreModel

ChurnScoreModel

EngagementScoreModel

stores models in a Model Registry.

Step 3: Real-time inference

When a new event arrives and profile is updated:

Profiling service builds a feature vector for that contact.

Sends it to the Inference Service.

Receives updated scores:

Risk, Churn, Engagement.

Writes scores into Scores in the profile document.

End-to-end flow for one event type:

Kafka topic strategy

Tenant isolation

Schema registry

Document profile

AI pipeline

NETCELL-EDA Architecture
Microservices Map (Strategy)

Thank you