CCE Solution Design

Version 0.3 – Draft

Table of Contents

CCE Solution Design

Version: 0.3 – Draft

---

1. Overview

The **Care Coordination Engine (CCE)** is an event-driven platform that addresses two fundamental challenges in primary healthcare delivery:

1. **Compliance** — Tracking protocol compliance and generating intelligence when defined conditions are met
2. **Coordination** — Enabling care handoffs across multiple independent health systems

CCE operates as an **event observer and care coordinator**. It receives events from participating systems (CHW apps, facility EMRs, disease registries, labs), tracks what has happened, compares against Compliance Protocols, and acts — either by generating intelligence events (alerts, reminders, escalations) or by coordinating cross-system handoffs.

1.1 Core Principle

CCE knows **"what happened and what should have happened"** — not **"what the patient's blood pressure is"** or **"how your EMR should work."**

Each participating system remains authoritative for its own clinical data and internal workflows. CCE observes and coordinates across system boundaries but does not control individual systems.

---

2. Purpose & Motivation

2.0 The Fundamental Problem

India's health systems generate millions of events daily across RCH, NCD Portal, Nikshay, U-WIN, facility EMRs, and other systems. Yet **60-70% of patients requiring follow-up care never complete their care pathway**. Events are **recorded** but not **coordinated**, and deviations from Compliance Protocols go undetected until it's too late.

2.1 Compliance

> **Note: Compliance Protocol vs Clinical Protocol**
>
> CCE does not deal with **Clinical Protocols** directly. Clinical Protocols define the medical/clinical decision-making logic — what treatment to prescribe, what

diagnosis to make, what tests to order based on clinical findings. These decisions remain entirely within participating systems (EMRs, clinical apps) and their clinicians.

CCE works with **Compliance Protocols** — these define the **expected care management pathway** that a patient should follow. A Compliance Protocol specifies: - **What events should happen** (e.g., visits, referrals, lab tests, follow-ups) - **When they should happen** (timing, sequence, dependencies) - **What to do when deviations occur** (alerts, reminders, escalations)

In other words, a Compliance Protocol answers: *"Given that a patient is enrolled in this care program, what is the expected sequence of care events, and are they happening on time?"* — not *"What clinical decisions should be made during those events?"*

CCE loads Compliance Protocols from external Knowledge Libraries and continuously compares actual events against expected care pathways.

2.1.1 Compliance Tracking

CCE tracks what has happened and compares it against what should have happened according to Compliance Protocols.

Capability	Description
Event Timeline	Maintains complete event history per patient journey across all participating systems
Protocol Comparison	Compares actual events against protocol expectations (timing, sequence, required actions)
Compliance Metrics	Calculates compliance rates at patient, facility, and program levels
Journey Visibility	Provides unified view of care pathway across system boundaries
Dashboard	Real-time visualization of patient journeys, compliance metrics, and care gaps via APIs (see Section 4.3.6.2)

Example: For an ANC protocol requiring visits at weeks 12, 20, 26, 30, 34, 36, 38, 40 — CCE tracks each visit event and calculates whether the patient is on track, ahead, or behind schedule.

2.1.2 Intelligence

CCE generates **intelligence** — actions taken in response to events, conditions, or patterns defined in the Compliance Protocol. Intelligence includes both notifications (alerts, reminders) and coordination (cross-system handoffs).

Key principle: CCE takes action; each participating system decides whether and how to respond. CCE acts, systems decide.

2.1.2.1 Notifications

Notification intelligence generates alerts that participating systems can consume and act upon.

Notification Type	Description	Example
Missed Events	Expected event did not occur within protocol timeline	"ANC visit due at week 20, now 5 days overdue"
Late Events	Event occurred but outside acceptable tolerance	"Referral appointment was 8 days late (protocol: 7 days)"
Out-of-Sequence	Events occurred in wrong order	"Lab results reviewed before sample collection logged"
Event Combinations	Specific combination of event types triggers alert	"High BP event combined with missed medication pickup event"
Escalations	Conditions that require supervisor attention	"High-risk pregnancy patient missed 2 consecutive visits"
Reminders	Proactive alerts before events become overdue	"ANC visit due in 3 days"

2.1.2.2 Coordination

Coordination intelligence enables **care handoffs across system boundaries**. When an event in one system requires action in another system, CCE routes the work — creating tasks, forwarding data, or triggering actions in the destination system.

> **Note:** Coordination is a business use case achieved through CCE's Intelligence capability. Technically, coordination actions (send task request, route data) are defined and triggered the same way as notification actions (send alert, reminder). CCE sends these actions to adaptors; each adaptor translates them into actual

records (e.g., tasks) within its target system. The distinction between notification and coordination is meaningful for understanding CCE's value proposition, but the underlying mechanism is unified.

The Coordination Gap:

No standardized infrastructure exists to coordinate care across system boundaries. Community health workers screen patients in mobile apps, facilities manage care in EMRs, and disease programs track enrollees in specialized systems — yet these activities remain disconnected.

Challenge	Current State	With CCE
Manual Handoffs	Care handoffs rely on phone calls, paper, or patient memory	Standardized tasks automatically route work between systems
Siloed Integrations	Point-to-point integrations that don't scale	Standards-based coordination that any system can join via adaptors

Example: Hypertension Referral Pathway

1. **CHW screens patient** in mobile app → detects high blood pressure
2. **Referral needed** → CHW app emits referral event
3. **CCE coordinates** → Sends coordination event to facility adaptor; adaptor creates task in facility EMR
4. **Facility receives task** → Doctor reviews patient, confirms diagnosis, prescribes treatment
5. **Follow-up scheduled** → CCE sends coordination event to CHW app adaptor; adaptor creates follow-up task
6. **CHW sees outcome** → Referral status and follow-up appear in CHW's task list

Without CCE, steps 2-6 require manual coordination (phone calls, paper forms) and have no systematic tracking. With CCE, the entire pathway is coordinated automatically — each system does its part, CCE ensures work flows across system boundaries.

Coordination Tracking:

When CCE creates a coordination action (e.g., referral task), tracking its completion uses the same Compliance Tracking mechanism. The protocol expects a completion event (e.g., an Encounter matching the referral completion trigger) within a defined timeline, and intelligence rules trigger if the event is late or missed.

2.1.3 Example: High-Risk Pregnancy Follow-up

```
Day 0:  ANM identifies high-risk pregnancy
        ✓ Event logged in RCH app
        → CCE: Patient enrolled in high-risk ANC protocol, tracking initiated
```

```
Day 1:  Doctor creates referral to District Hospital
        ✓ Event logged in facility EMR
        → CCE: Tracking referral, expecting appointment within 7 days

Day 5:  No appointment event received
        → CCE: Generates "referral_pending_alert" (medium severity)

Day 8:  No appointment event received
        → CCE: Generates "referral_overdue_alert" (high severity)
        → CCE: Generates "supervisor_escalation" event

Day 10: Appointment completed
        ✓ Event logged
        → CCE: Referral loop closed (3 days late, logged for metrics)
```

Without compliance tracking, the Day 5-8 gap goes unnoticed. With CCE, deviations are detected in real-time and intelligence events enable timely intervention.

2.2 Unified Model: Compliance + Intelligence

CCE's value comes from the combination of Compliance Tracking and Intelligence (including both Notifications and Coordination).

Scenario	With CCE
Referral created	Task request sent to facility adaptor + completion tracked against protocol timeline
Patient misses follow-up	Alert generated + follow-up task request sent + escalation if needed
Lab results delayed	Deviation detected + notification to ordering physician + patient reminder
High BP during ANC visit	Referral task request sent to facility adaptor + CHW notified + completion tracked

Without CCE, these scenarios require manual coordination (phone calls, paper forms) and deviations go unnoticed. With CCE, the entire care pathway is tracked and coordinated automatically.

2.3 Phased Implementation

CCE can be deployed incrementally, starting with visibility and adding automation over time. This approach aligns with the Emitter/Receiver Adaptor model (see Section 4.3.7).

Phase	What's Deployed	CCE Capabilities	Value Delivered
Phase 1: Compliance	Emitter Adaptor(s) only	Compliance Tracking, Dashboard	Visibility into patient journeys, compliance metrics, care gap identification
Phase 2: Intelligence	Add Receiver Adaptor(s)	+ Notifications, Coordination	Automated alerts, escalations, cross-system task routing

Phase 1: Compliance

- Deploy Emitter Adaptor(s) to capture events from participating systems
- CCE tracks patient journeys and calculates compliance metrics
- Dashboard provides visibility into care gaps (no automated actions yet)
- Use this phase to validate protocol configurations and establish baseline metrics

Phase 2: Intelligence

- Deploy Receiver Adaptor(s) for systems that should receive intelligence events
- CCE generates notifications (alerts, reminders, escalations) and coordination actions
- Systems receive and act on intelligence events, closing care gaps
- Add incrementally — start with high-priority protocols or facilities

---

3. Prerequisites

CCE coordinates care across multiple independent systems. The following must be in place before deploying CCE:

3.1 Common Patient Registry (Mandatory)

A **shared patient registry** (Master Patient Index / Client Registry) must exist across all participating systems. This provides a common patient identifier that CCE uses to correlate events and actions across systems.

Requirement	Description
Shared patient ID	All participating systems must use the same patient identifier (e.g., national health ID, MPI-assigned ID)
Patient lookup	Systems must be able to resolve the shared ID to their local patient records
Event correlation	Events submitted to CCE must include the shared patient ID in the context

Why this is mandatory:

- CCE cannot perform patient matching — it relies on explicit shared identifiers
- Without a common ID, an event about "Patient A" from SmartCare cannot be linked to an action for the same patient in OpenMRS
- Cross-system coordination fundamentally depends on knowing "this is the same patient"

CCE does not provide:

- Patient registry functionality — this is outside CCE's scope
- Patient matching algorithms — CCE uses the ID provided, it does not attempt to match patients

> **Note:** If CCE is deployed for a **single source system only** (e.g., one CHW app with no cross-system coordination), a common patient registry is **not mandatory**. In this scenario, the source system's own patient identifier can be used directly as the `subject` in events, since cross-system patient identity resolution is not needed.

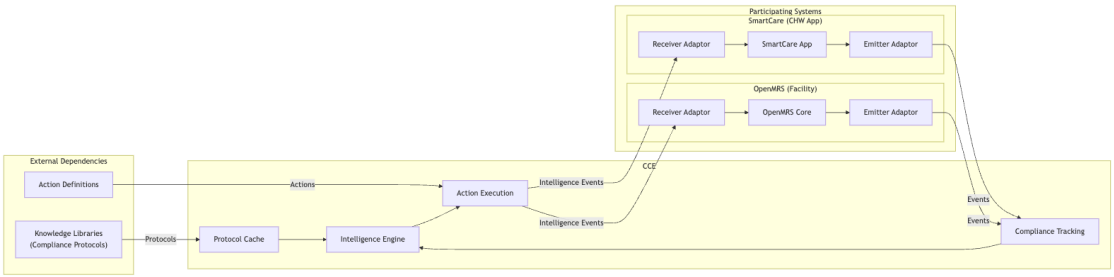
3.2 Identity Provider (Mandatory)

An **OIDC/OAuth 2.0 compatible Identity Provider** must be available for authenticating participating systems (see Section 6.2 for details). If an existing Identity Provider is not available, an instance can be set up. This activity is separate from CCE setup.

---

4. Architecture

4.1 Context Diagram



CCE Context Diagram

4.2 Design Principles

- **Event observer, not controller** — CCE observes events and generates intelligence; participating systems decide how to act
- **Events are triggers, not state** — Authoritative state lives within participating app resources, not event streams

- **CCE owns coordination and compliance, not clinical data** — Each system remains authoritative for its own clinical data; CCE tracks compliance state and coordinates handoffs
- **Participating systems remain autonomous** — Systems can use any internal architecture; adaptors translate between CCE and each system
- **Protocol-first** — Configurable compliance Protocols and Action Definitions; no ad-hoc logic
- **Hybrid CloudEvents + FHIR events** — Events use the CloudEvents envelope with clinical data in the `data` payload — preferably FHIR resources (Encounters, Observations, ServiceRequests, etc.) for interoperability, but non-FHIR JSON payloads are also supported.
- **Clinical Data** — Does not need to be a complete clinical record, limited to what is necessary for compliance tracking.

4.3 Integration Interface

From an integrator's perspective, CCE exposes the following touchpoints:

4.3.1 Events

Participating systems emit events to CCE when clinical activities occur.

4.3.1.1 Event Structure

Design Decision: Hybrid CloudEvents + FHIR (ADR Accepted)

CCE events follow the **CloudEvents** specification (CNCF graduated standard) as the envelope, with **FHIR resources** in the `data` payload for clinical content. This decision is documented in the CCE Event Structure ADR.

- **CloudEvents envelope** — provides a lightweight, standards-based structure with built-in support for idempotency, content typing, and extensibility.
- **FHIR in `data`** — clinical payloads use FHIR resources (Encounter, Observation, etc.) with `datacontenttype: "application/fhir+json"`. Non-FHIR JSON payloads are also supported with `datacontenttype: "application/json"`.
- **CCE extension attributes** — compliance tracking context (protocol instance, action ID, facility) is carried as CloudEvents extension attributes. These are optional routing hints — CCE uses **inferred matching** by default.

Rationale: - **Inferred matching** — CCE determines which protocol step an event fulfills by evaluating trigger definitions against the `data` payload's fields. No CCE-specific vocabulary or step IDs required from the source. - **Lower adaptor burden** — Emitter adaptors produce a CloudEvents message with the clinical data in `data`. They do not need to understand CCE's protocol model. - **Interoperability** — CloudEvents is widely supported (SDKs in every major language); FHIR resources enable integration with any FHIR-producing system (e.g., ABDM, RHIE). - **Minimal data exposure** — The `data` payload carries only the

fields needed for matching. Full clinical detail remains authoritative in source systems.


```
{
  "specversion": "1.0",
  "id": "evt-uuid-12345",
  "source": "smartcare-chw",
  "type": "org.openphc.cce.encounter",
  "subject": "ABHA-1234567890",
  "time": "2026-01-15T10:30:00Z",
  "datacontenttype": "application/fhir+json",

  "sourceeventid": "enc-991122",
  "protocolinstanceid": "pi-uuid-67890",
  "protocoldefinitionid": "anc-high-risk-v2.1",
  "actionid": "anc-visit-2",
  "facilityid": "PHC-456",

  "data": {
    "resourceType": "Encounter",
    "status": "finished",
    "class": {
      "system": "http://terminology.hl7.org/CodeSystem/v3-ActCode",
      "code": "AMB"
    },
    "type": [
      {
        "coding": [
          {
            "system": "http://openphc.org/encounter-types",
            "code": "anc-visit"
          }
        ]
      }
    ],
    "serviceType": {
      "coding": [
        {
          "system": "http://openphc.org/service-types",
          "code": "high-risk-anc"
        }
      ]
    },
    "period": {
      "start": "2026-01-15T10:00:00Z",
      "end": "2026-01-15T10:30:00Z"
    }
  }
}
```

Key Elements:

CloudEvents Core Attributes:

Attribute	Required	Description
specversion	Always	CloudEvents specification version (always `"1.0"`).
id	Always	Unique identifier for this event (UUID). Combined with `source`, forms the CloudEvents uniqueness key for idempotency.
source	Always	Identifier of the registered system emitting the event (e.g., `"smartcare-chw"`).
type	Always	Event category (e.g., `encounter`, `observation`, `referral`, `enrollment`). Used for routing, filtering, and observability — **not** for step matching (that uses the two-tier matching model against `data`).
subject	Always	Shared patient identifier from the common patient registry (e.g., `"ABHA-1234567890"`).
time	Always	When the clinical activity actually occurred at the source (ISO 8601, UTC). CCE also records `received_at` internally.
datacontenttype	Always	Content type of the `data` payload. `"application/fhir+json"` for FHIR resources; `"application/json"` for non-FHIR payloads.
data	Always	The clinical payload representing the activity. Can be a **FHIR resource** (matched via `data-added` triggers with `DataRequirement`) or an **arbitrary JSON payload** (matched via `named-event` triggers with JSONLogic conditions). Should include the fields needed for trigger matching — it does not need to be a complete clinical record.

CCE Extension Attributes (all optional):

Attribute	Description
sourceeventid	The event's ID in the source system. Combined with `source`, provides secondary idempotency key.
protocolinstanceid	Explicit routing hint: which protocol instance (patient journey) this event belongs to. When provided, CCE skips protocol instance inference. Useful when a patient has multiple active instances of the same protocol.
protocoldefinitionid	Explicit routing hint: which protocol definition this event

Attribute	Description
	belongs to. Value corresponds to the PlanDefinition's `identifier.value`. When provided, CCE narrows step matching to only steps within this protocol.
actionid	Explicit routing hint: which step this event fulfills. Value corresponds to the PlanDefinition's `action.id`. When provided along with protocol context, CCE bypasses inferred step matching entirely and uses explicit matching (validate trigger match + step state only).
facilityid	Facility where the event occurred. Used for intelligence routing and analytics.

> **Note:** CloudEvents extension attribute names must be lowercase with no dots, hyphens, or underscores per the CloudEvents spec. All CCE extensions follow this convention (e.g., `actionid`, `protocolinstanceid`). In CCE's internal data model and API responses, the equivalent fields use snake_case (e.g., `action_id`, `protocol_instance_id`). In FHIR PlanDefinition, the native notation is dot-separated (e.g., `action.id`).

Event Idempotency:

CCE guarantees idempotent event processing: - Events with the same `id` + `source` combination are accepted only once (CloudEvents uniqueness key) - Duplicate submissions return success but do not modify state - Secondary idempotency via `source` + `sourceeventid` pair

Timestamps:

- `time` — When the clinical activity actually happened (provided by source system, CloudEvents core attribute)
- `received_at` — When CCE received the event (set by CCE internally, not in the CloudEvents message)

This distinction is critical for handling late-arriving events and accurate compliance calculations.

4.3.2 Compliance Protocol

A **Compliance Protocol Definition** defines a care pathway as a sequence of expected clinical events with timing, dependencies, and intelligence rules.

4.3.2.1 Protocol Definition Structure

Protocol definitions use the **FHIR PlanDefinition** resource as the canonical format. The API accepts and returns PlanDefinition resources. Initially, protocols are authored as PlanDefinition JSON and submitted via the API. A **visual builder UI** for authoring protocols

— where users design protocols visually and the UI generates PlanDefinition JSON — is planned as a future enhancement.

> **Note:** In case of CCE, FHIR PlanDefinition is not being used to drive clinical workflows. It is being re-purposed to express expected compliance to a given workflow.

Protocol definitions are immutable once published (`status: "active"`). To change a protocol, publish a new version (`status: "retired"` on the old version). FHIR PlanDefinition's built-in `version` and `status` fields support this lifecycle natively.

4.3.2.2 Why PlanDefinition

- **Standard structure** — Steps, sequencing, timing, dependencies, conditions, and action references are native PlanDefinition concepts
- **Interoperability** — Protocols can be published to FHIR registries, shared across implementations, and consumed by FHIR-aware tooling
- **Version pinning** — FHIR's canonical URL + version scheme (`PlanDefinition/anc-high-risk|2.1`) provides built-in version pinning for protocol instances
- **Trigger definitions** — PlanDefinition's `TriggerDefinition` with `DataRequirement` maps naturally to CCE's resource-based matching

> **Note — Intelligence Rules:** PlanDefinition does not have native constructs for CCE's intelligence model (step state monitoring, severity, escalation targets). Intelligence rules are modeled as **nested sub-actions with extensions**. This is acknowledged as an area for refinement — the mapping works structurally but the intelligence semantics are carried by extensions rather than native FHIR elements.

The following skeleton shows the overall shape of a PlanDefinition protocol — how actions nest, where triggers and intelligence rules sit, and how timing, dependencies, and extensions are used.

```
{
  "resourceType": "PlanDefinition",
  "url": "http://openphc.org/fhir/PlanDefinition/anc-high-risk",
  "version": "2.1",
  "status": "active",

  "action": [
    //Step 1
    {
      "id": "enrollment",
      "trigger": [
        {
          "type": "data-added",
          "data": [
            { "type": "EpisodeOfCare", "codeFilter": [...] }
```

```
          ]
        }
      ]
    },
    //Step 2
    {
      "id": "anc-visit-1",
      //Event matching pattern
      "trigger": [
        {
          "type": "data-added",
          "data": [
            {
              "type": "Encounter",                          // Tier 1: resource
 type
              "codeFilter": [
                {
                  "path": "type",                           // Tier 1: encounte
r type
                  "code": [{ "system": "http://openphc.org/encounter-types",
"code": "anc-visit" }]
                },
                {
                  "path": "status",                         // Tier 1: must be
completed
                  "code": [{ "code": "finished" }]
                },
                {
                  "path": "class",                          // Tier 1: ambulato
ry visit
                  "code": [{ "system": "http://terminology.hl7.org/CodeSystem
/v3-ActCode", "code": "AMB" }]
                },
                {
                  "path": "serviceType",                    // Tier 1: high-ris
k ANC service
                  "code": [{ "system": "http://openphc.org/service-types", "c
ode": "high-risk-anc" }]
                }
              ]
            }
          ],
          "condition": {                                    // Tier 2: cross-fi
eld logic via JSONLogic
            "language": "text/jsonlogic",
            "expression": "{\">\": [{\"var\": \"resource.participant.length\
"}, 0]}"
            // e.g., require at least one participant recorded
          }
        }
```

```
      ],
      "relatedAction": [
        {
          "actionId": "enrollment",
          "relationship": "after-start",
          "offsetDuration": { "value": 0, "unit": "wk" }
        }
      ],
      "extension": [
        { "url": ".../tolerance-days", "valueInteger": 3 }
      ],
      "action": [
        {
          "id": "anc-visit-1-overdue-alert",
          "condition": [
            {
              "kind": "applicability",
              "expression": { "language": "text/jsonlogic", "expression": "...
" }
            }
          ],
          "definitionCanonical": "ActivityDefinition/send-alert",
          "extension": [
            { "url": ".../intelligence-severity", "valueCode": "medium" },
            { "url": ".../intelligence-target", "valueCode": "assigned_worker
" }
          ]
        }
      ]
    },
    //Step 3
    {
      "id": "anc-visit-2",
      "trigger": [
        {
          "type": "data-added",
          "data": [
            { "type": "Encounter", "codeFilter": [...] }
          ]
        }
      ],
      "relatedAction": [
        {
          "actionId": "enrollment",
          "relationship": "after-start",
          "offsetDuration": { "value": 8, "unit": "wk" }
        },
        {
          "actionId": "anc-visit-1",
          "relationship": "after-end"
```

```
        }
      ],
      "extension": [
        { "url": ".../tolerance-days", "valueInteger": 5 }
      ],
      "action": [
        {
          "id": "anc-visit-2-due-reminder",
          "condition": [...],
          "definitionCanonical": "ActivityDefinition/send-reminder"
        },
        {
          "id": "anc-visit-2-overdue-escalation",
          "condition": [...],
          "definitionCanonical": "ActivityDefinition/send-escalation"
        },
        {
          "id": "anc-visit-2-on-complete",
          "condition": [...],
          "definitionCanonical": "ActivityDefinition/schedule-next-anc-visit"
        }
      ]
    },
    //Step 4
    {
      "id": "high-bp-followup",
      "requiredBehavior": "could",
      "trigger": [
        {
          "type": "data-added",
          "data": [
            { "type": "Observation", "codeFilter": [...] }
          ],
          "condition": {
            "language": "text/jsonlogic",
            "expression": "..."
          }
        }
      ],
      "action": [
        {
          "id": "high-bp-referral-action",
          "definitionCanonical": "ActivityDefinition/hypertension-referral-ac
tion"
        },
        {
          "id": "hypertension-referral",
          "trigger": [...],
          "relatedAction": [
            {
```

```
            "actionId": "high-bp-followup",
            "relationship": "after-end",
            "offsetDuration": { "value": 7, "unit": "d" }
          }
        ]
      }
    ]
  },
  //Step 5
  {
    "id": "monthly-checkup",
    "trigger": [
      {
        "type": "data-added",
        "data": [
          { "type": "Encounter", "codeFilter": [...] }
        ]
      }
    ],
    "timingTiming": {
      "repeat": {
        "count": 6,
        "frequency": 1,
        "period": 1,
        "periodUnit": "mo"
      }
    },
    "extension": [
      { "url": ".../tolerance-days", "valueInteger": 5 }
    ]
  }

 ]
}
```

4.3.2.3 How PlanDefinition Elements Map to Compliance Concepts

PlanDefinition Element	Compliance Concept	Notes
`action.id`	Step identifier	Unique within the PlanDefinition. Referenced as `action_id` in step instances and intelligence output.
`action.trigger`	Event matching	`TriggerDefinition` — either `type: "data-added"` with `DataRequirement` for FHIR resource matching, or `type: "named-event"` with `condition` for non-FHIR payload matching. See Trigger Definition below.

PlanDefinition Element	Compliance Concept	Notes
`action.relatedAction`	Timing + dependencies	`relationship: "after-start"` with `offsetDuration` for timing relative to enrollment or other steps. `relationship: "after-end"` for dependencies (`depends_on`).
`action.timingTiming.repeat`	Recurring steps	`count`, `frequency`, `period`, `periodUnit` for repeating actions.
`action.requiredBehavior`	Optional steps	`"could"` for optional steps (e.g., high-bp-followup).
Nested `action[]` with `condition`	Intelligence rules, actions, branches	Sub-actions with `kind: "applicability"` conditions. `definitionCanonical` references the ActivityDefinition to execute. **Distinguishing sub-action types at runtime:** a nested action that contains its own `trigger` is a **branched step** (a dependent sub-step activated by an event); a nested action without a `trigger` is an **intelligence rule** (evaluated against step runtime state).
`action.definitionCanonical`	Action definition reference	Points to an `ActivityDefinition` resource that defines what to do.

4.3.2.4 Trigger Definition

Each step's `trigger` uses FHIR's `TriggerDefinition` to express *what event fulfills this step*. CCE's trigger matching is built on a **two-tier matching model**:

Tier	Mechanism	Purpose	Performance
Tier 1: Structural Filters	`DataRequirement` (FHIR) or `name` (non-FHIR)	Declarative field matching — resource type, coded values (type, status, class, serviceType), date ranges, profiles	**Fast and indexable.** CCE indexes these at protocol load time. When an event arrives, the engine uses inverted index to find candidate steps directly — no full scan.
Tier 2: Condition (JSONLogic)	`condition` expression (`text/jsonlogic`)	Complex logic that structural filters cannot express — cross-	**Expressive but evaluated at runtime.** Only applied to the small candidate set that passed Tier 1.

Tier	Mechanism	Purpose	Performance
		field comparisons, array operations, value thresholds, computed checks	

Why two tiers: Structural filters alone are fast but limited to simple code matching. Conditions alone are expressive but expensive to evaluate across all steps. The two-tier model gives protocol authors both speed and flexibility — Tier 1 narrows the candidate set cheaply, Tier 2 handles the complex matching scenarios.

CCE supports two trigger modes depending on whether the event carries a FHIR resource or a non-FHIR payload:

FHIR Mode: `data-added` — for FHIR resource payloads (preferred)

Uses `DataRequirement` for structural matching (Tier 1), with an optional `condition` for complex logic (Tier 2).

Field	Description
type	`"data-added"`
data[].type	FHIR resource type to match (e.g., `Encounter`, `Observation`, `ServiceRequest`)
data[].codeFilter	Array of coded field filters. Each specifies a `path` (e.g., `type`, `status`, `code`) and a list of `code` values to match against.
data[].dateFilter	(Optional) Date field filters for temporal matching.
data[].profile	(Optional) Required FHIR profiles the resource must conform to.
condition	(Optional) FHIR `Expression` with `language: "text/jsonlogic"`. For complex matching that `DataRequirement` cannot express (cross-field logic, value comparisons, array operations).

`DataRequirement` fields are structural, declarative, and indexable — they enable fast pre-filtering before any expression evaluation.

Non-FHIR Mode: `named-event` — for non-FHIR payloads

Uses a `name` string for coarse-grained indexing and a `condition` expression for matching against arbitrary JSON fields.

Field	Description
type	`"named-event"`
name	A category string for the event (e.g., `"lab-result"`, `"visit-completed"`). Used as an index key for fast pre-filtering — analogous to `DataRequirement.type` for FHIR mode.

Field	Description
condition	FHIR `Expression` with `language: "text/jsonlogic"`. Evaluated against the event's payload fields via the `%resource` variable (e.g., `resource.visit_type`, `resource.status`). This is the primary matching mechanism for non-FHIR payloads.

A single PlanDefinition can mix both modes — some steps matching FHIR resources from modern systems, others matching custom JSON from legacy emitters.

	`data-added` (FHIR)	`named-event` (non-FHIR)
Structural pre-filter	`DataRequirement` — resource type, codeFilter, dateFilter. Highly indexable.	`name` — category string. Lightweight index key.
Condition evaluation	Optional, for complex cases	Primary matching mechanism
Emitter produces	FHIR resource	Arbitrary JSON with agreed-upon fields
Interoperability	Standard, portable	Custom, per-deployment

4.3.2.5 Expression Language (JSONLogic)

Expressions in trigger conditions use **JSONLogic** — registered as a custom expression language (`"language": "text/jsonlogic"`) within FHIR's `Expression` type. JSONLogic is easy to work with and has well supported libraries.

> **Note — Alternative Expression Languages:** If JSONLogic proves insufficient for complex expressions during implementation, FHIR's `Expression` type natively supports several alternatives:

Language	`Expression.language`	Strengths	Trade-offs
FHIRPath	`text/fhirpath`	Native FHIR resource navigation; collection filtering (`.where()`, `.exists()`); understands FHIR data types; lighter than CQL; no compilation step	Less powerful than CQL for terminology operations and cross-resource logic; no library/reuse model
CQL (Clinical Quality Language)	`text/cql`	Purpose-built for clinical decision support; native terminology operations (`memberOf`, `in` ValueSet); date arithmetic; type-aware quantity comparisons	Heavier runtime (requires CQL engine + ELM compilation); may need a terminology

Language	`Expression.language`	Strengths	Trade-offs
		(e.g., `>= 140 'mmHg'`); supports shared expression libraries via `Expression.reference`	service for ValueSet operations; primarily Java implementations
FHIR Query	`application/x-fhir-query`	Reuses familiar FHIR search syntax as a match expression (e.g., `Encounter?type=anc-visit&status=finished&date=ge2026-01-01`); highly readable for simple conditions	Limited expressiveness for complex logic; no cross-field comparisons or computed checks

These can coexist — the `Expression.language` field allows each trigger condition to declare its own language, and CCE's condition evaluator dispatches to the appropriate engine.

JSONLogic Examples: | Expression | JSONLogic | |————|————| | `stepState == 'overdue'` | `{"==": [{"var": "stepState"}, "overdue"]}` | | `daysOverdue > 3` | `{">": [{"var": "daysOverdue"}, 3]}` | | `stepState == 'overdue' && daysOverdue > 3` | `{"and": [{"==": [{"var": "stepState"}, "overdue"]}, {">": [{"var": "daysOverdue"}, 3]}]}` | | `resource.status == 'finished'` | `{"==": [{"var": "resource.status"}, "finished"]}` | | Check resource has participants | `{">": [{"var": "resource.participant.length"}, 0]}` |

4.3.2.6 CCE Extensions

CCE defines the following FHIR extensions for compliance-specific concepts that PlanDefinition does not natively support:

Extension URL	Type	Used On	Description
`http://openphc.org/fhir/StructureDefinition/tolerance-days`	`valueInteger`	`action` (step)	Grace period in days before a step is marked "overdue"
`http://openphc.org/fhir/StructureDefinition/intelligence-severity`	`valueCode`	Nested `action` (intelligence rule)	Severity level: `low`, `medium`, `high`, `critical`
`http://openphc.org/fhir/StructureDefinition/intelligence-target`	`valueCode`	Nested `action` (intelligence rule)	Who receives the intelligence output: `patient`, `assigned_worker`, `supervisor`, `facility`

Note — Refinement Needed: Intelligence rules are currently modeled as nested sub-actions with extensions. This works structurally but the intelligence semantics (step state monitoring, escalation chains, severity routing) are carried entirely by extensions. This area will be refined as the design matures — potential approaches include a CCE-specific FHIR profile for PlanDefinition, or a dedicated intelligence layer that references PlanDefinition actions.

4.3.2.7 Step Definition (`action.id`) vs Step Instance

Concept	Where Used	Description
action.id	PlanDefinition	The **step definition identifier**. Each top-level `action` in a PlanDefinition *is* a step definition — it contains the trigger, timing, dependencies, tolerance, and intelligence rules for that step. The `action.id` uniquely identifies it within the protocol (e.g., `anc-visit-2`, `monthly-checkup`).
step_instance_id	Runtime	Unique identifier for a specific *occurrence* of a step for a patient (e.g., the 3rd monthly checkup)

This distinction is critical for: - **Repeating steps:** `monthly-checkup` has one `action.id` but 6 `step_instance_ids` (one per month) - **Branched steps:** Dynamically added steps get unique instance IDs - **Auditing:** Know exactly which occurrence was completed

4.3.2.8 Intelligence Rules

Intelligence rules are modeled as **nested sub-actions** within a PlanDefinition step action. Each intelligence rule is a child action with: - A `condition` (JSONLogic expression evaluated against step runtime state) - A `definitionCanonical` pointing to an `ActivityDefinition` (what to do) - Extensions for `intelligence-severity` and `intelligence-target`

```
{
  "id": "anc-visit-2-overdue-escalation",
  "title": "Escalation — supervisor (3+ days overdue)",
  "condition": [
    {
      "kind": "applicability",
      "expression": {
        "language": "text/jsonlogic",
        "expression": "{\"and\": [{\"==\": [{\"var\": \"stepState\"}, \"overd
ue\"]}, {\">\": [{\"var\": \"daysOverdue\"}, 3]}]}"
      }
    }
  ],
  "definitionCanonical": "ActivityDefinition/send-escalation",
  "extension": [
    { "url": "http://openphc.org/fhir/StructureDefinition/intelligence-severi
```

```
ty", "valueCode": "high" },
    { "url": "http://openphc.org/fhir/StructureDefinition/intelligence-target
", "valueCode": "supervisor" }
  ]
}
```

> **Note:** This is an area for refinement. Intelligence rules are structurally valid as nested PlanDefinition actions, but the compliance-specific semantics (step state monitoring, severity-based routing, escalation chains) are carried entirely by extensions and CCE's runtime engine. The PlanDefinition structure acts as a container; CCE's engine provides the intelligence behavior.

4.3.3 Protocol Instances (Journeys)

A **Protocol Instance** represents a specific patient's journey through a **Compliance Protocol** (PlanDefinition) — one enrollment, one episode of care.

> **FHIR Alignment:** A Protocol Instance is conceptually equivalent to a FHIR `CarePlan` — a plan applied to a specific patient that references a PlanDefinition. CCE may adopt CarePlan as the native representation for protocol instances in the future. Currently, the runtime representation is CCE-internal.

4.3.3.1 Why Protocol Instances Matter

A patient may be enrolled in the same protocol multiple times: - Pregnancy 1 (2024) → ANC Protocol Instance A - Pregnancy 2 (2026) → ANC Protocol Instance B - NCD enrollment → relapse → re-enrollment

Without distinct instance IDs, CCE cannot distinguish which episode an event belongs to.

4.3.3.2 Protocol Instance Creation

When CCE receives an event whose `data` payload matches a protocol's enrollment action trigger (the `enrollment` action in the PlanDefinition), it creates a new Protocol Instance:

```
{
  "protocol_instance_id": "pi-uuid-67890",
  "patient_id": "ABHA-1234567890",
  "protocol_canonical": "http://openphc.org/fhir/PlanDefinition/anc-high-risk
|2.1",
  "enrolled_at": "2026-01-01T00:00:00Z",
  "status": "active",
  "step_instances": [ ]
}
```

Key properties: - `protocol_instance_id` — Unique identifier for this journey - `protocol_canonical` — Canonical URL with version (`PlanDefinition/anc-high-risk|2.1`). The instance is **pinned** to this version (immutable). This uses FHIR's built-in canonical URL + version scheme. - `status` — `active`, `completed`, `withdrawn`, `expired` -

`step_instances` — Runtime step instances created progressively as the journey progresses (see below)

Progressive Step Instantiation:

CCE uses **progressive instantiation** — step instances are created only when their dependencies are satisfied, not all at once at enrollment. This keeps the runtime model clean: every step instance that exists is actionable.

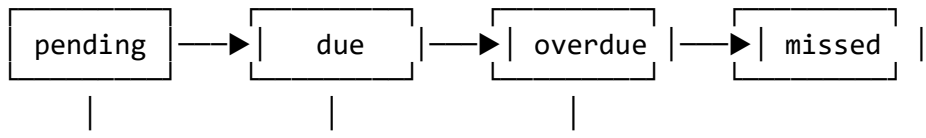
```
Day 0 — Enrollment event arrives:
  Protocol Instance created (active)
  → anc-visit-1 instantiated as 'pending' (due: Day 14)
    (no dependencies — immediate)
  → anc-visit-2 NOT instantiated yet
    (depends on anc-visit-1 completion)

Day 10 — anc-visit-1 completed:
  → anc-visit-1 transitions to 'completed'
  → anc-visit-2 instantiated as 'pending' (due: Day 56)
    (dependency satisfied — now created)

Day 56 — anc-visit-2 due date reached:
  → anc-visit-2 transitions to 'due'
```

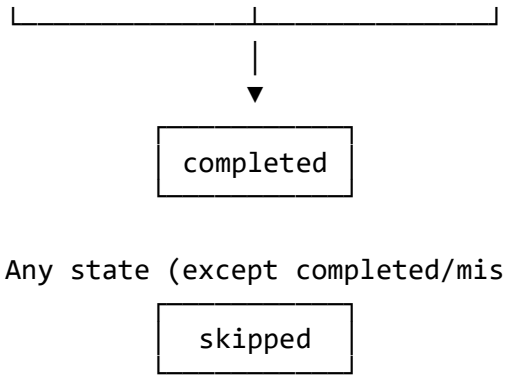
Why progressive instantiation: - **`pending` is always meaningful** — every `pending` step has its dependencies met and is trackable. There is no ambiguity between "waiting on a prerequisite" and "waiting for due date". - **Optional and branched steps** are naturally handled — they are instantiated only when their trigger fires, not speculatively at enrollment. - **Full pathway visibility** is not lost — the PlanDefinition itself serves as the blueprint. Dashboards overlay the PlanDefinition structure (all defined steps) with current step instance states (actively tracked steps). Steps not yet instantiated are shown as "upcoming" directly from the definition.

> **Cold Start Problem:** Protocol instances are created only when CCE observes an enrollment event. When CCE is introduced into a source system that already has enrolled patients, those patients will not have protocol instances — their enrollment events occurred before CCE existed. This is a known limitation addressed in Section 8.1 (Brownfield Deployment).

4.3.3.3 Step Instance Lifecycle

A step instance is created only when its dependencies are satisfied (progressive instantiation — see above). Once created, it enters the `pending` state and transitions through the following states:





Transition	Trigger
(created) → pending	Step instantiated when dependencies are satisfied. Dependencies met, due date not yet reached.
pending → due	Due date reached (scheduler)
pending → completed	Event received before due (early completion)
pending → skipped	Step explicitly skipped (e.g., optional step, protocol withdrawn)
due → overdue	Overdue date reached (scheduler). A `deviation` record is created (see Section 7.2.2).
due → completed	Event received while due
due → skipped	Step explicitly skipped
overdue → missed	Missed date reached (scheduler). A `deviation` record is created (see Section 7.2.2).
overdue → completed	Event received while overdue (late completion)
overdue → skipped	Step explicitly skipped

4.3.3.4 Protocol Version Pinning

Once created, a protocol instance is pinned to a specific PlanDefinition version via the `protocol_canonical` URL (e.g., `PlanDefinition/anc-high-risk|2.1`). If the PlanDefinition is updated (new version published, old version retired), existing instances continue using their pinned version. Only new enrollments use the new version.

This ensures compliance tracking consistency — a patient's journey is evaluated against the rules that were in effect when they enrolled. FHIR's canonical versioning scheme (`url|version`) provides this pinning natively.

4.3.4 Event Processing & Matching

This section describes how CCE processes incoming events and matches them to protocol steps. It bridges the concepts defined in Events (Section 4.3.1), Compliance Protocol (Section 4.3.2), and Protocol Instances (Section 4.3.3).

4.3.4.1 Resource-Based Matching

Events carry a clinical payload in the CloudEvents `data` field, and protocol steps (PlanDefinition actions) define **trigger definitions** that match against the `data` fields using the **two-tier matching model** (see Section 4.3.2.4): structural filters for fast indexed lookup, then optional JSONLogic conditions for complex logic. CCE uses these triggers to infer which protocol step an event fulfills — emitters do not need to understand CCE's protocol model.

FHIR resource payloads (trigger mode: `data-added`) are the preferred format. Common FHIR resource types and their compliance use:

FHIR Resource	Compliance Use	Key Fields for Matching
Encounter	Visit tracking (ANC visits, follow-ups, referral completions)	`type`, `class`, `serviceType`, `status`, `period`
Observation	Screening results, vital signs detection (e.g., high BP)	`code`, `status`, `value[x]`, `effectiveDateTime`
ServiceRequest	Referrals, lab orders	`code`, `category`, `status`, `priority`, `intent`
MedicationDispense	Medication pickup tracking	`medicationCodeableConcept`, `status`, `whenHandedOver`
EpisodeOfCare	Program enrollment	`type`, `status`, `period`

Non-FHIR payloads (trigger mode: `named-event`) are supported for legacy or non-FHIR systems. The emitter produces an arbitrary JSON payload with fields agreed upon during implementation. Matching is handled by the trigger's JSONLogic condition expression.

> **Note:** The specific resource types, profiles, code systems, and payload schemas used in a deployment are determined during implementation and documented in the deployment's configuration. CCE does not prescribe a fixed set — it matches whatever the trigger definitions in the protocol specify.

4.3.4.2 CCE Processing Flow

```
1. Event arrives
   - Check idempotency (id + source, or source + sourceeventid)
   - If duplicate → return success, no state change
   - Record received_at timestamp
                    ↓
2. Protocol Instance Resolution
   - Look up active protocol instances for this patient (subject)
   - If protocolinstanceid provided → use it directly (explicit mode)
   - If protocoldefinitionid provided → narrow to instances of that protocol
   - Otherwise → consider all active instances for this patient
```

```
                    ↓
3. Step Matching (Inferred)
   - For each active protocol instance, evaluate the event's `data` payload
     against each step's trigger definition:
     a. Structural match — does `data` match the step's trigger filters?
        (resource type, code filters, status)
     b. Condition match — does the JSONLogic condition (if any) evaluate to t
rue?
     c. State filter — is this step instance in an eligible state?
        (pending, due, or overdue — not already completed or missed)
     d. Timing filter — does the event's time fall within the step's
        expected window (considering tolerance)?
   - If actionid is provided → bypass inference, validate
     trigger match and step state only (explicit mode)
   - Result: zero, one, or multiple candidate matches (see Section 4.3.4.5)
                    ↓
4. Step Completion
   - Mark matched step instance(s) completed (completed_at = time)
   - Record on-time/late/early status
   - Evaluate intelligence rules
                    ↓
5. Action Execution
   - Check step's action triggers (defined in protocol)
   - Evaluate conditions (resource fields, state, timing)
   - If condition met → execute linked action definition
```

4.3.4.3 Step Matching — Inferred Model

CCE uses **inferred step matching** — the engine determines which protocol step an event fulfills by evaluating the event's `data` payload against trigger definitions in the protocol. Emitters do not need to know CCE's step identifiers or protocol model.

Matching applies the **two-tier matching model** defined in Section 4.3.2.4, followed by runtime filters:

1. **Tier 1 — Structural filters** (`DataRequirement` or `name`): resource type, coded fields, status values. Indexed at protocol load time for fast lookup.
2. **Tier 2 — Condition** (JSONLogic, if defined): cross-field logic, value comparisons, array operations. Evaluated only on candidates that passed Tier 1.
3. **Runtime filters** — step state (is this step eligible?) and timing (does `time` fall within the expected window?).

Each stage narrows the candidate set for the next. In practice, most matching resolves at Tier 1; Tier 2 and runtime filters handle edge cases.

> **Performance Note:** CCE indexes PlanDefinition and ActivityDefinition trigger metadata (resource type, code systems, code values) at protocol load time. When an event arrives, the engine uses this index to look up candidate actions directly — avoiding a full scan across all protocols and steps. Only the small set

of structurally matched candidates undergoes condition evaluation and runtime filtering. The runtime step matching design document details the inverted index data structure, algorithm, and narrowing funnel — including an optimization where structural matching via the global index precedes patient context intersection, rather than the sequential protocol-instance-first order described in the processing flow above (Section 4.3.4.2).

4.3.4.4 Explicit Matching

When an emitter provides `actionid` (and optionally `protocolinstanceid`), CCE uses **explicit matching** — it goes directly to the named step and validates that: - The step's trigger matches the incoming `data` payload (structural filters + condition) - The step instance is in an eligible state

This allows protocol-aware emitters (e.g., tightly integrated apps) to use precise routing while protocol-unaware emitters (e.g., ABDM, generic FHIR sources) rely on inference.

4.3.4.5 Ambiguity Handling

Inferred matching can produce zero, one, or multiple candidate step matches. CCE uses a **fail-safe** approach — if the engine cannot deterministically resolve to exactly one step within a single protocol instance, it does not auto-complete any step. The event is accepted (it is a valid clinical event), the ambiguity is logged, and an intelligence event is generated for monitoring.

Scenario	Resolution	Auto-complete?
Same trigger, different steps (one protocol)	Dependencies → State → Timing proximity → Fail safe	Yes, if filters resolve to one; No, if ambiguous
Repeating steps	Complete the current active instance; instantiate next repeat	Always yes (deterministic)
Cross-protocol	Complete in each protocol independently	Yes, in all matching protocols
Multiple instances, same protocol	Require `protocolinstanceid`	Only if extension attribute provided or single-instance match
Zero matches	Accept, log, no action	N/A
Optional steps	Evaluate independently, don't compete with required	Yes, independently
Truly unresolvable	Accept event, log, intelligence event	No — manual resolution via extension attributes

4.3.4.6 Late and Out-of-Order Events

CCE accepts late-arriving events within configurable cutoffs: - Events are processed based on `time`, not `received_at` - Late events can still complete overdue steps - Step state is recalculated based on actual event timing

4.3.5 Action Definition

An **Action Definition** specifies *what CCE does* when an intelligence rule fires. Intelligence rules (Section 4.3.2.8) define *when* to act — they are modeled as nested sub-actions within PlanDefinition steps, with JSONLogic conditions evaluated against step runtime state. Each intelligence rule references an action definition via `definitionCanonical` (e.g., `"ActivityDefinition/send-alert"`), which defines the execution behavior.

An action definition specifies:

- **Action type** — What CCE should do (e.g., send notification, create task, forward data to an external system)
- **Message template** — The content to deliver (e.g., alert message, task description), with variable placeholders resolved from the intelligence event context
- **Routing** — Which Receiver Adaptor(s) should receive the action (e.g., facility adaptor, supervisor notification channel)

Example (conceptual):

```
ActivityDefinition: send-escalation
  Action: Send notification
  Target: supervisor (resolved from intelligence-target extension)
  Message: "Patient {patient_name} ANC visit is {days_overdue} days overdue"
  Delivery: Route to Receiver Adaptor registered for the patient's facility
```

> **Note:** The intelligence rule's condition (in PlanDefinition) determines *when* this fires. The action definition determines *what happens* when it does. This separation allows the same action definition to be reused across multiple intelligence rules.

Action Definition Format (Pending Refinement)

The PlanDefinition intelligence model already references FHIR `ActivityDefinition` resources via `definitionCanonical`. The reference format is established. What remains pending is the **internal structure and execution model** of each ActivityDefinition — specifically, how integrators express the execution details (message templates, routing logic, retry behavior).

Intelligence Event Output

When an intelligence rule fires (see Section 4.3.2.8), CCE produces an intelligence event with the following structure:

```
{
  "intelligence_type": "escalation",
```

```
  "timestamp": "2026-01-28T08:00:00Z",
  "context": {
    "patient_id": "ABHA-1234567890",
    "protocol_instance_id": "pi-uuid-67890",
    "protocol_canonical": "http://openphc.org/fhir/PlanDefinition/anc-high-ri
sk|2.1",
    "action_id": "anc-visit-2",
    "step_instance_id": "si-uuid-11111"
  },
  "alert": {
    "action": "escalation",
    "severity": "high",
    "target": "supervisor",
    "message": "Patient Priya Sharma ANC visit is 5 days overdue"
  }
}
```

This output is delivered to Receiver Adaptors via the intelligence event delivery mechanisms described in Section 4.3.7.4.

4.3.6 API Interface

Systems interact with CCE through a REST API.

4.3.6.1 API Versioning

All endpoints are prefixed with `/v1/`. Example: `/v1/events`, `/v1/protocol-definitions`.

When breaking changes are introduced, a new version (`/v2/`) will be released. Previous versions remain supported for a deprecation period (minimum 6 months).

4.3.6.2 Endpoints

Operation	Description
`POST /v1/events`	Submit a CloudEvents message (with FHIR or JSON data payload) for compliance tracking
Compliance Protocol Definitions (PlanDefinition)	
`POST /v1/protocol-definitions`	Register a compliance protocol definition (FHIR PlanDefinition resource)
`GET /v1/protocol-definitions`	List registered protocol definitions
`GET /v1/protocol-definitions/{id}`	Retrieve a specific PlanDefinition by ID
`DELETE /v1/protocol-definitions/{id}`	Remove a protocol definition (only if not in use)
Protocol Instances	
`GET /v1/protocol-instances`	List protocol instances (filterable by patient,

Operation	Description
	protocol, status, facility)
Patient Compliance	
`GET /v1/patients/{patient_id}/protocol-tracking`	List all protocol instances being tracked for a patient
`GET /v1/patients/{patient_id}/protocol-tracking/{protocol_instance_id}`	Get tracking details including all step instances with current state
`GET /v1/patients/{patient_id}/events`	Event timeline for patient journey visualization
Dashboard & Analytics	
`GET /v1/protocols/{protocol_definition_id}/compliance-summary`	Aggregate compliance metrics (% on-track, overdue, missed) for a protocol
`GET /v1/facilities/{facility_id}/compliance-summary`	Facility-level compliance metrics across all protocols
`GET /v1/protocols/{protocol_definition_id}/patients`	List patients by compliance status
`GET /v1/intelligence/summary`	Intelligence events summary (counts by type, period)
Intelligence Action Definitions	
`POST /v1/action-definitions`	Register an action definition that defines actions to execute for specific events
`GET /v1/action-definitions`	List registered action definitions
`GET /v1/action-definitions/{id}`	Retrieve a specific action definition by ID
`PUT /v1/action-definitions/{id}`	Update an existing action definition
Action Runs	
`GET /v1/action-runs`	List action runs
`GET /v1/action-runs/{id}`	Retrieve a specific action run by ID
`GET /v1/action-runs/{id}/status`	Get current execution status of an action run
`POST /v1/action-runs/{id}:cancel`	Cancel an in-progress action run

Note — Adaptor Registration: CCE does not maintain a registry of emitter adaptors. Emitters are external services that authenticate via OAuth and call `POST /v1/events` — CCE identifies the source system from the CloudEvents `source` attribute. Receiver adaptor registration (if needed for intelligence event delivery) will be defined as part of the Intelligence Subsystem design.

> **Note:** Protocol definitions (PlanDefinition resources) are immutable once published (`status: "active"`). To update a protocol, publish a new version (new PlanDefinition with incremented `version`, old one set to `status: "retired"`). The `PUT` operation is intentionally not provided for protocol definitions.

4.3.6.3 Response Envelope

Success Response (list endpoints):

```
{
  "data": [ ... ],
  "pagination": {
    "limit": 50,
    "next_cursor": "eyJpZCI6MTIzfQ==",
    "has_more": true
  }
}
```

Success Response (single resource):

```
{
  "data": { ... }
}
```

Error Response:

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Missing required field: 'data' must contain a valid payload",
    "details": {
      "field": "data",
      "value": null
    }
  }
}
```

Example: GET /v1/patients/{patient_id}/protocol-tracking/{protocol_instance_id}

> **Note:** `step_instances` contains only steps that have been instantiated (dependencies satisfied). Steps not yet instantiated can be derived from the PlanDefinition for full pathway visualization.

```
{
  "protocol_instance_id": "pi-uuid-67890",
  "patient_id": "ABHA-1234567890",
  "protocol_canonical": "http://openphc.org/fhir/PlanDefinition/anc-high-risk
|2.1",
  "enrolled_at": "2026-01-01T00:00:00Z",
  "status": "active",
  "compliance_rate": 0.5,
```

```
  "step_instances": [
    {
      "step_instance_id": "si-uuid-001",
      "action_id": "anc-visit-1",
      "state": "completed",
      "due_date": "2026-01-15T00:00:00Z",
      "completed_at": "2026-01-13T10:30:00Z",
      "completed_by": "smartcare-chw",
      "completion_status": "on_time"
    },
    {
      "step_instance_id": "si-uuid-002",
      "action_id": "anc-visit-2",
      "state": "overdue",
      "due_date": "2026-02-26T00:00:00Z",
      "days_overdue": 5
    }
  ]
}
```

In this example, `anc-visit-1` is completed and `anc-visit-2` has been instantiated (its dependency on `anc-visit-1` is satisfied) and is now overdue. Steps like `anc-visit-3` and `monthly-checkup` are not yet listed because their dependencies have not been met — they remain defined in the PlanDefinition and can be shown as "upcoming" by the UI.

4.3.7 Adaptors

Adaptors are not part of CCE core. An adaptor is a service that bridges between CCE and external systems, translating between CCE's event format and each system's internal APIs or data models.

Adaptor ownership: - **System owners** are generally responsible for building, hosting, and maintaining adaptors for their systems - **CCE team** may provide reference adaptors for popular systems (e.g., OpenMRS, DHIS2) and Medtronic Labs products (e.g., Spice) to accelerate adoption

4.3.7.1 Adaptor Types

Adaptors have two distinct responsibilities, which can be implemented together or separately:

Type	Direction	Aligns With	Purpose
Emitter Adaptor	External System → CCE	Compliance Tracking	Transforms application events into CloudEvents messages and submits to CCE
Receiver Adaptor	CCE → External System	Intelligence	Receives intelligence events and translates to application actions

This separation enables: - **Independent deployment** — Deploy Emitter only for tracking, add Receiver later for intelligence - **Different ownership** — Emitter and Receiver can be built by different teams - **Different granularity** — One Emitter can serve multiple applications (e.g., via shared infrastructure)

4.3.7.2 Emitter Adaptors

An Emitter Adaptor captures events from an external system and submits them to CCE for compliance tracking.

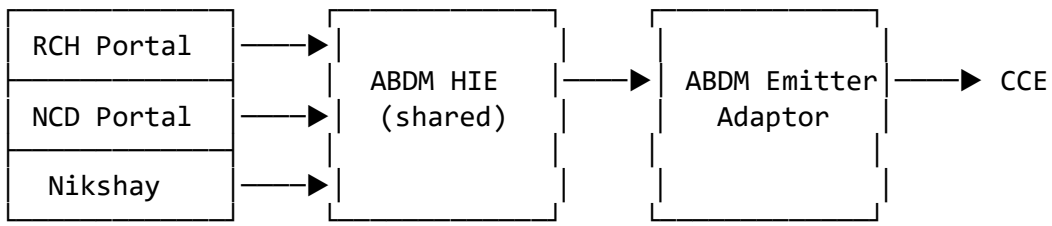
Responsibilities: - Monitor or receive events from the source system - Transform source data into a **CloudEvents message** — set the `data` payload to a FHIR resource (e.g., Encounter, Observation) with `datacontenttype: "application/fhir+json"`, or an arbitrary JSON payload for non-FHIR systems with `datacontenttype: "application/json"` - Populate CloudEvents core attributes (`source`, `type`, `subject`, `time`, etc.) - Submit events to CCE via `POST /v1/events` - Handle errors and retries

> **Key simplification from inferred matching:** Emitter adaptors do **not** need to understand CCE's compliance protocol model. They do not need to determine which protocol or step an event belongs to — that is CCE's responsibility. The adaptor's job is to produce a well-formed CloudEvents message with the clinical activity in the `data` payload. CCE's trigger-based matching handles the rest.

Shared Infrastructure as Emitter:

In some deployments, a shared health information infrastructure can serve as the event source for multiple applications. The Emitter Adaptor is built for the shared infrastructure rather than each individual application.

Example: ABDM (Ayushman Bharat Digital Mission)



This approach reduces integration effort — ABDM-compliant systems require no individual emitter adaptors.

Example Emitter Adaptors: smartcare-emitter, abdm-emitter, RHIE mediator

4.3.7.3 Receiver Adaptors

A Receiver Adaptor receives intelligence events from CCE and translates them into actions within the target system.

Responsibilities: - Receive intelligence events from CCE via the configured delivery mode - Translate CCE intelligence events into system-specific actions (e.g., create task, send notification) - Acknowledge successful processing

4.3.7.4 Intelligence Event Delivery

> **Note:** The intelligence delivery mechanism described here is preliminary. The detailed design — including receiver registration, delivery guarantees, and filtering — will be finalized as part of the Intelligence Subsystem design.

CCE delivers intelligence events to Receiver Adaptors. The following delivery modes are under consideration:

Delivery Modes:

Mode	How It Works	Best For
Webhook (Push)	CCE POSTs events to adaptor's registered URL	Simple integrations; always-online systems
Topic Subscription (Pull)	Adaptor subscribes to a message topic; pulls events at its own pace	Robust integrations; systems with potential downtime

4.3.7.5 Adaptor Authentication

Emitter Adaptors authenticate to CCE using standard OAuth tokens — the same mechanism used by any participating system calling CCE APIs. No separate adaptor registration is needed. CCE identifies the source system from the CloudEvents `source` attribute in each event.

Receiver Adaptors may require registration with CCE if the Intelligence Subsystem uses a push-based delivery model (e.g., webhooks). This would include delivery endpoint, authentication credentials, and event filters. The registration mechanism will be defined as part of the Intelligence Subsystem design.

---

5. Error Handling

CCE handles errors at each layer of the processing pipeline. Each layer has distinct failure modes and handling strategies.

> **Terminology Note:** This section uses "step" exclusively in the compliance/protocol sense (PlanDefinition `action.id` entries). For action execution, "operation" is used to avoid ambiguity.

5.1 API & Gateway Layer

The CCE Gateway Service (Section 7.2.6) returns errors to callers using the error envelope defined in Section 4.3.6.5. All errors at this layer are logged in the audit log (Section 6.5).

HTTP Status	Error Code	Cause	Handling
400	`VALIDATION_ERROR`	Request body fails schema validation	Caller fixes and resubmits
401	`AUTHENTICATION_FAILED`	Invalid or expired access token	Caller re-authenticates; do not retry
403	`AUTHORIZATION_FAILED`	Valid token but insufficient scope	Caller requests appropriate scopes; do not retry
404	`NOT_FOUND`	Resource does not exist	Caller verifies identifier
429	`RATE_LIMITED`	Rate limit exceeded	Retry after `Retry-After` period
502/503	`SERVICE_UNAVAILABLE`	Backend service failure	Transient; retry with backoff

5.2 Event Ingestion Layer

The Collector Service (Section 7.2.1) validates events and publishes to Kafka. Errors occur after the Gateway has authenticated the request.

Error	Caller Response	CCE Handling
CloudEvents schema invalid (missing required attributes, bad format)	400 `VALIDATION_ERROR`	Event rejected
Duplicate event (`id` + `source` already exists)	200 (idempotent acceptance)	No state change
Kafka publication failure	503 `SERVICE_UNAVAILABLE`	Event written to dead-letter store; automatically reprocessed when Kafka recovers

Dead-letter handling: Events that cannot be published to Kafka are written to a durable dead-letter store with the full payload, failure reason, and timestamp. They are automatically retried (preserving per-patient ordering) and monitored via queue-depth metrics with configurable alert thresholds.

---

6. Security

6.1 Data Security in Transit

All communication involving CCE must be encrypted using TLS.

Communication Path	Requirement
External → CCE (Participating systems calling CCE APIs)	HTTPS required, TLS 1.2 minimum (TLS 1.3 recommended)
CCE → Adaptors (CCE calling registered adaptor endpoints)	HTTPS required, TLS 1.2 minimum (TLS 1.3 recommended)
Internal (within CCE) (Between CCE components if distributed)	TLS required between components

Requirements:

- **HTTPS only** — All CCE API endpoints are served over HTTPS. HTTP connections are rejected.
- **TLS version** — TLS 1.2 is the minimum supported version. TLS 1.3 is recommended for improved security and performance.
- **Adaptor endpoints** — Receiver adaptors that receive intelligence events must expose HTTPS endpoints.
- **Certificate validation** — CCE validates server certificates for all outbound connections. Invalid or expired certificates cause connection failures.

Not required at this stage:

- **Application-level payload encryption** — Events carry `data` payloads with minimal fields for matching (type codes, status, timing) — not full clinical records. TLS provides sufficient transport-level protection. If events begin carrying sensitive clinical data (e.g., lab values, diagnoses) in the future, additional encryption (e.g., JWE) may be considered based on sensitivity requirements.

6.2 Inbound Authentication and Authorization

Inbound refers to requests **coming into CCE** from participating systems (e.g., submitting events, querying action runs).

6.2.1 Authentication

CCE delegates authentication to an **external Identity Provider** using OIDC/OAuth 2.0.

- **External Identity Provider** — CCE does not manage users or credentials. Authentication is handled by an external OIDC/OAuth-compatible provider (e.g., Keycloak, Auth0, country-specific identity systems).

- **Token-based** — Participating systems obtain access tokens from the Identity Provider and include them in API requests to CCE (`Authorization: Bearer <token>`).
- **Token validation** — CCE validates tokens against the configured Identity Provider (signature verification, expiration, issuer).

6.2.2 Authorization

CCE includes **built-in authorization** using OAuth scopes. This can be **disabled** when an external system (e.g., API gateway, OpenHIM) handles authorization.

Scopes:

Scope	Grants Access To
`events:write`	Submit events to CCE (`POST /v1/events`)
`protocol-definitions:read`	View protocol definitions
`protocol-definitions:write`	Create/update/delete protocol definitions
`protocol-tracking:read`	View patient compliance tracking data
`dashboard:read`	View dashboard and analytics data (compliance summaries, patient lists, intelligence summaries)
`action-definitions:read`	View action definitions
`action-definitions:write`	Create/update/delete action definitions
`action-runs:read`	View action runs and status
`action-runs:write`	Cancel action runs
`admin`	Full access (all scopes)

6.4 Data Security at Rest

Event Storage:

CCE persists event records as required for: - Compliance tracking and explainability (why was a step marked late?) - Reprocessing and debugging - Audit trails

> **Note:** Events carry `data` payloads with minimal fields for trigger matching (type codes, status, timing). While these payloads may contain some clinical indicators (e.g., blood pressure values for high-BP detection), they are not full clinical records. Full clinical detail remains authoritative in source systems. If richer clinical data is included in the future, additional measures (field-level encryption, stricter access controls) may be required.

Data protection measures:

Measure	Description
Encryption at rest	Event store is encrypted using AES-256 or equivalent

Measure	Description
Retention policies	Configurable TTL per resource type or protocol (e.g., retain ANC events for 2 years)
Access control	Event data accessible only via authorized API calls with appropriate scopes
Minimum necessary	Events contain only identifiers and metadata needed for compliance tracking

6.5 Audit Logging

CCE maintains **audit logs separate from application logs**. Audit logs provide an immutable record of who did what and when, for compliance and accountability purposes.

6.5.1 What is Audited

Some audit information is captured in CCE's persistence storage as part of resource metadata (e.g., `created_by`, `updated_by`, `created_at` on action definitions and adaptors). Audit logs capture additional events, especially transient operations:

Event Category	Examples
Authentication	Successful/failed login attempts, token validation failures
Resource changes	Action definition created/updated/deleted, adaptor registered/updated/deleted
Action execution	Action run started, step executed, action completed/failed
Adaptor operations	Adaptor operation invoked, success/failure response
Authorization	Access denied due to insufficient scope

7. Compliance Subsystem Design

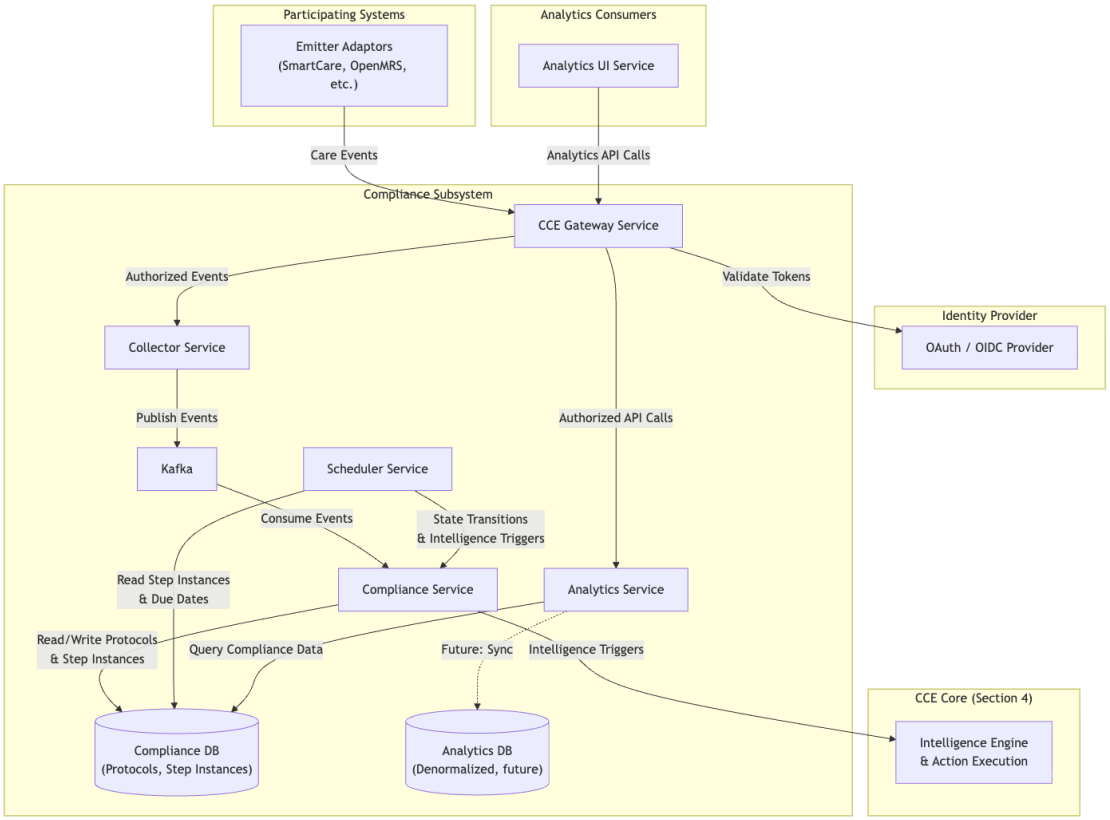
Relationship to Section 4 (Architecture):

Section 4 describes CCE's **logical architecture** — the conceptual components (Compliance Tracking, Protocol Cache, Intelligence Engine, Action Execution) and the integration interface (events, protocols, APIs, adaptors). That view is implementation-agnostic.

This section describes the **service-level architecture** — how the compliance functionality is decomposed into deployable services, how they communicate (Kafka, REST), and how data flows through them. The Collector Service handles event ingestion and validation, and the Compliance Service implements the Compliance Tracking and Protocol Cache concerns from Section 4. Intelligence triggers generated by the Compliance Service feed into the Intelligence Engine and Action Execution pipeline described in Section 4. The Analytics Service and Analytics UI Service implement the Dashboard & Analytics APIs defined in Section 4.3.6.

The Compliance Subsystem is the set of services responsible for ingesting care events, evaluating them against compliance protocols, persisting compliance state, and surfacing analytics to authorized consumers. It is composed of six services that communicate through Kafka and REST APIs.

7.1 Architecture Overview



Compliance Subsystem Architecture

7.2 Component Descriptions

7.2.1 Collector Service

The Collector Service validates and ingests care events, publishing them to Kafka topics for downstream processing. It does not receive traffic directly from external callers — all inbound requests arrive via the CCE Gateway Service, which has already validated the access token and authorized the operation.

Aspect	Detail
Responsibility	Validate event structure and publish to Kafka
Input	Pre-authorized care events forwarded by the CCE Gateway Service
Output	Events published to Kafka topic(s)
Validation	CloudEvents schema validation (required attributes, format), duplicate detection (idempotency by `id` + `source`)
Authentication	Not handled directly — the CCE Gateway Service validates tokens and authorizes operations before forwarding requests
Scalability	Stateless; horizontally scalable

Key behaviors:

- Receives requests only from the CCE Gateway Service (not exposed to external traffic directly)
- Performs lightweight structural validation — does **not** interpret clinical or compliance meaning
- Assigns a `received_at` timestamp and `correlation_id` if not already present
- Publishes events to a partitioned Kafka topic (partitioned by `subject` / patient ID to preserve per-patient ordering)
- Returns an acknowledgement to the caller once the event is durably written to Kafka
- Dead-letters malformed or unprocessable events for later inspection

7.2.2 Compliance Service

The Compliance Service is the core processing engine. It consumes events from Kafka, resolves which compliance protocol(s) and step(s) each event matches, and persists the compliance state (protocol instances, step instances, deviations).

Aspect	Detail
Responsibility	Match incoming events to compliance protocols and steps; track compliance state
Input	Events consumed from Kafka
Output	Updated protocol instances and step instances in the Compliance DB; intelligence triggers
Data Store	Compliance DB — hosts protocol definitions (containing action/step definitions), protocol instances (per-patient enrollments), and step instances (pending/due/overdue/completed/missed/skipped)
Processing Model	Consumer-group based Kafka consumption; at-least-once delivery with idempotent writes

Key behaviors:

- Consumes events from the Kafka topic in order (per partition / per patient)
- For each event, performs **inferred matching** to resolve:
    - **Which protocol instance(s)** the event pertains to — by looking up active instances for the patient, optionally narrowed by `protocoldefinitionid` or `protocolinstanceid` extension attributes
    - **Which step(s)** within those instances the event satisfies — by evaluating the `data` payload against each step's **trigger definition** (structural filters or JSONLogic condition), then filtering by step state and timing

    - If `actionid` extension attribute is provided, bypasses inference and uses explicit matching (validate trigger + state only)
- Creates or updates **protocol instances** (a patient's enrollment in a specific protocol) and **step instances** (progressively instantiated when dependencies are satisfied; tracks individual step completions, misses, or deviations)
- Evaluates timing constraints — marks steps as `completed`, `overdue`, or `missed` based on protocol-defined windows
- Generates **intelligence triggers** (e.g., deviation detected, step overdue) that can feed into the existing Intelligence Engine and Action Execution pipeline
- Maintains an idempotent processing model — reprocessing the same event does not create duplicate state

Compliance DB schema (conceptual):

Entity	Description
`plan_definition`	A FHIR PlanDefinition resource loaded from Knowledge Libraries (e.g., ANC High-Risk v2.1). Stored as the canonical protocol definition.
`protocol_instance`	A specific patient's enrollment in a protocol, pinned to a PlanDefinition version via `protocol_canonical` (e.g., patient ABHA-123 enrolled in `PlanDefinition/anc-high-risk|2.1`)
`step_instance`	A specific step occurrence for a patient — references the PlanDefinition `action.id`, tracks status (`pending`, `due`, `overdue`, `missed`, `completed`, `skipped`), matched event(s), and timing
`deviation`	A recorded deviation from the expected protocol pathway. Created when a step instance transitions to `overdue` or `missed`, or when ambiguity handling (Section 4.3.4.5) flags an unresolvable match. Captures the deviation type, timestamp, and associated intelligence event (if any). Enables deviation-focused analytics and audit trails beyond what step state alone provides.

7.2.3 Scheduler Service

The Scheduler Service is responsible for **time-based step state transitions**. While the Compliance Service handles event-driven transitions (an event arrives → step is completed), many transitions in the step lifecycle are triggered by the passage of time — due dates being reached, tolerance windows expiring, reminders needing to fire. The Scheduler Service handles all of these.

Aspect	Detail
Responsibility	Progress step instances through time-based state transitions; trigger time-based intelligence rules

Aspect	Detail
Data Source	Compliance DB — reads step instances with their due dates, tolerance windows, and current states
Output	State transition requests to the Compliance Service; intelligence triggers for time-based rules
Processing Model	Periodic polling with configurable interval (e.g., every 5 minutes). Scans for step instances whose time thresholds have been crossed since the last run.

Time-based transitions handled:

Transition	Condition	Action
`pending → due`	Current time ≥ step's `due_date`	Update state to `due`
`due → overdue`	Current time ≥ `due_date` + `tolerance_days`	Update state to `overdue`; create deviation record
`overdue → missed`	Current time ≥ protocol-defined missed cutoff	Update state to `missed`; create deviation record

Progressive instantiation triggers:

When a step instance transitions to `missed`, the Scheduler checks whether any dependent steps (those with `relatedAction.relationship: "after-end"` pointing to this step) can now be instantiated. If all dependencies for a downstream step are satisfied, the Scheduler creates the new step instance in `pending` state with a computed `due_date`.

> **Note:** Progressive instantiation on `completed` transitions is handled by the Compliance Service when an event arrives. The Scheduler's role is to pick up the `missed` case — time-based transitions where a step's missed cutoff is reached and its dependents should still be instantiated. For example, if `anc-visit-1` is missed, `anc-visit-2` may still need to be created depending on protocol rules.

Intelligence rule evaluation (time-based):

The Scheduler also evaluates time-based intelligence rules — conditions that fire based on elapsed time rather than incoming events:

Rule Type	Example	How Scheduler Handles It
Reminders	"Send reminder 3 days before due"	Scans `pending`/`due` steps where `due_date - now ≤ 3 days`; fires intelligence trigger
Overdue alerts	"Alert when step is 5+ days overdue"	Scans `overdue` steps where `days_overdue > threshold`; fires intelligence trigger
Escalations	"Escalate to supervisor after 7	Scans `overdue` steps matching escalation conditions; fires intelligence trigger

Rule Type	Example	How Scheduler Handles It
	days overdue”	

Intelligence triggers from the Scheduler follow the same pipeline as event-driven triggers — they are sent to the Compliance Service, which evaluates the intelligence rules in the PlanDefinition and generates intelligence events for the Intelligence Engine.

Key behaviors:

- Runs on a configurable schedule (default: every 5 minutes)
- Processes step instances in batches, ordered by `due_date`
- Idempotent — re-running the same cycle produces no duplicate transitions or intelligence events
- Tracks a high-water mark to avoid reprocessing; recovers gracefully from missed cycles
- Does **not** handle event-driven transitions — those remain the Compliance Service’s responsibility
- Stateless between runs — all state is in the Compliance DB

7.2.4 Analytics Service

Relationship to Section 4 Dashboard & Analytics APIs:

Section 4.3.6 defines Dashboard & Analytics API endpoints such as `GET /v1/protocols/{id}/compliance-summary` and `GET /v1/facilities/{id}/compliance-summary`. The Analytics Service is the **backend implementation** of those endpoints for the Compliance Subsystem. It serves the same data — protocol adherence, facility compliance, patient status — through the API surface defined in Section 4.

The Analytics Service serves compliance analytics data — protocol adherence rates, deviation trends, facility-level summaries, and patient-level compliance timelines. It does not receive traffic directly from external callers — all inbound requests arrive via the CCE Gateway Service, which has already validated the access token and authorized the operation.

Key capabilities:

- **Protocol adherence** — Percentage of patients completing all steps within defined windows, filterable by protocol, facility, time range
- **Deviation analysis** — Most common deviations, average delay, deviation trends over time
- **Facility-level dashboards** — Compliance summary per facility or geographic region
- **Patient timeline** — Full compliance timeline for a specific patient across all enrolled protocols

- **Export** — Supports data export in standard formats (CSV, JSON) for external analysis

Phase 2 — Denormalized Analytics Store:

In the initial phase, the Analytics Service queries the Compliance DB directly. As data volumes grow and query patterns diverge from transactional needs, the Analytics Service will maintain its own denormalized read-optimized store. This transition will be transparent to API consumers.

Phase	Data Source	Trade-off
Phase 1	Compliance DB (direct)	Simpler deployment; acceptable at low-to-moderate scale
Phase 2	Dedicated Analytics DB	Query performance at scale; eventual consistency with Compliance DB

7.2.5 Analytics UI Service

The Analytics UI Service is the front-end application that provides dashboards, reports, and visualizations for compliance data. All API calls from the UI are directed to the CCE Gateway Service, which validates the user's access token, authorizes the operation, and forwards the request to the Analytics Service.

Aspect	Detail
Responsibility	Render compliance dashboards and reports for end users
Backend	All API calls go through the CCE Gateway Service, which routes authorized requests to the Analytics Service
Authentication	Obtains OAuth access tokens from the Identity Provider; includes tokens in all API requests to the CCE Gateway Service
Users	Program managers, district health officers, facility administrators

Key views:

- **Overview dashboard** — High-level compliance metrics across all protocols and facilities
- **Protocol drill-down** — Adherence and deviation details for a specific protocol
- **Facility view** — Compliance performance for a specific facility or region
- **Patient view** — Individual patient compliance timeline and status
- **Alerts and deviations** — Active deviations requiring attention, with filtering and prioritization

7.2.6 CCE Gateway Service

Relationship to Section 6.2 (Inbound Authentication and Authorization):

Section 6.2 defines CCE's authentication and authorization model — OIDC/OAuth 2.0 token validation, scope-based access control, and the `authorization_enabled` toggle. The CCE Gateway Service described here is the **service-level implementation** of that model for the Compliance Subsystem. It enforces the same scopes defined in Section 6.2 and respects the same Identity Provider configuration. When `authorization_enabled` is set to `false` (e.g., when an external API gateway handles authorization upstream), the CCE Gateway Service passes requests through without performing authorization checks, but still handles routing.

The CCE Gateway Service is the **single entry point** for all inbound traffic into the Compliance Subsystem. Every external request — whether an event submission from an Emitter Adaptor or an analytics query from the UI — passes through the CCE Gateway Service first. It validates the caller's OAuth access token, evaluates authorization based on token claims, and then routes the authorized request to the appropriate backend service. Backend services (Collector, Analytics Service) are not exposed to external traffic directly.

Aspect	Detail
Role	API gateway — single entry point for all inbound requests to the Compliance Subsystem
Responsibility	Authenticate callers, authorize operations, and route requests to backend services
Identity Provider	Delegates token validation to the configured OAuth / OIDC provider (signature verification, expiry, audience)
Authorization Model	Claims-based — extracts roles, scopes, and contextual claims (e.g., `facility_id`, `district_id`) from the token to determine access
Routing	Forwards authorized requests to the appropriate backend service based on the request path
Backend Services	Collector Service (event ingestion), Compliance Service (protocol and instance management, patient compliance queries), Scheduler Service (time-based step transitions), Analytics Service (dashboard and analytics queries)

Request flow:

```
External Caller
    │
    ▼
CCE Gateway Service
    ├── 1. Validate access token (signature, expiry, issuer, audience)
    ├── 2. Extract claims (roles, scopes, facility_id, district_id, user_id)
    ├── 3. Authorize operation (do claims permit this request?)
    │       ├── If denied → return 403 with reason
```

```
    │       └── If allowed ↓
    └── 4. Route to backend service
            ├── POST /v1/events                          → Collector Service
            ├── /v1/protocol-definitions/*               → Compliance Service
            ├── /v1/protocol-instances/*                 → Compliance Service
            ├── /v1/patients/*/protocol-tracking/*       → Compliance Service
            ├── /v1/patients/*/events                    → Compliance Service
            ├── /v1/protocols/*/compliance-summary        → Analytics Service
            ├── /v1/facilities/*/compliance-summary       → Analytics Service
            ├── /v1/protocols/*/patients                 → Analytics Service
            ├── /v1/intelligence/summary                 → Analytics Service
            ├── /v1/action-definitions/*                 → Compliance Service
(provisional)
            └── /v1/action-runs/*                        → Compliance Service
(provisional)
```

> **Note:** Action definition and action run endpoints are provisionally routed to the Compliance Service. These endpoints conceptually belong to the Intelligence Engine & Action Execution pipeline (Section 4). Routing will be updated when the Intelligence Subsystem design is finalized.

Key behaviors:

- **Single entry point** — All external HTTP traffic enters the Compliance Subsystem through the CCE Gateway Service. Backend services are deployed on internal networks and are not directly accessible from outside.
- **Token validation** — Verifies token signature, expiry, issuer, and audience against the configured Identity Provider
- **Claims extraction** — Parses claims from the validated token (e.g., `roles`, `scope`, `facility_id`, `district_id`, `user_id`)
- **Operation authorization** — Evaluates whether the extracted claims permit the requested operation. Examples:
  - A `facility_admin` role with `facility_id=PHC-456` can view analytics for PHC-456 only
  - A `district_officer` role with `district_id=D-12` can view analytics for all facilities in district D-12
  - An Emitter Adaptor's system-level token with `scope=events:write` can submit events
- **Request routing** — After successful authorization, forwards the request (with extracted claims attached as headers or context) to the appropriate backend service
- **Consistent error handling** — Returns standardized error responses for authentication failures (401) and authorization failures (403) across all endpoints
- **Token caching** — Caches validated tokens and authorization decisions (with short TTL) to reduce latency on repeated calls

- **Claims forwarding** — Passes extracted claims (e.g., `user_id`, `facility_id`, `roles`) to backend services so they can apply data-level filtering (e.g., Analytics Service returns only data for the caller's facility)

7.3 Data Flow Summary

Step	From	To	Mechanism	Description
1	Emitter Adaptors / Analytics UI	CCE Gateway Service	HTTP/REST	All inbound requests enter through the CCE Gateway Service
2	CCE Gateway Service	Identity Provider	HTTP/REST (OIDC)	Token validation (signature, expiry, issuer)
3	CCE Gateway Service	Collector Service	HTTP/REST (internal)	Authorized event submissions are forwarded to the Collector
4	CCE Gateway Service	Analytics Service	HTTP/REST (internal)	Authorized analytics queries are forwarded to the Analytics Service
5	Collector Service	Kafka	Kafka Producer	Validated events are published to Kafka
6	Kafka	Compliance Service	Kafka Consumer	Events are consumed for compliance evaluation
7	Compliance Service	Compliance DB	Database writes	Protocol/step instances are created or updated
8	Scheduler Service	Compliance DB	Database reads	Scans step instances for time-based transitions (pending→due, due→overdue, overdue→missed)
9	Scheduler Service	Compliance Service	Internal call	Sends state transition requests and time-based intelligence triggers
10	Analytics Service	Compliance DB	Database reads	Compliance data is queried for analytics

7.4 Deployment Considerations

- **CCE Gateway Service** — The externally-facing entry point; deploy behind a load balancer with TLS termination. Stateless with short-lived token cache. Must be highly available — if the CCE Gateway Service is down, no inbound traffic reaches backend services. Deploy multiple instances for redundancy.

- **Collector Service** — Internal service; not exposed to external traffic. Stateless; horizontally scalable.
- **Compliance Service** — Internal service. Stateless consumers in a Kafka consumer group; scaling is achieved by adding consumers (up to the number of partitions).
- **Scheduler Service** — Internal service. Runs on a configurable schedule (e.g., every 5 minutes). Typically a single instance with leader election to avoid duplicate processing. Lightweight — scans the Compliance DB for step instances that need time-based transitions. Must be resilient to missed cycles (idempotent, catches up on next run).
- **Analytics Service** — Internal service; not exposed to external traffic. Stateless; horizontally scalable; may benefit from caching (e.g., Redis) for frequently requested aggregations.
- **Analytics UI Service** — Static front-end assets served via CDN or web server; minimal infrastructure footprint. The UI makes API calls to the CCE Gateway Service's public endpoint.
- **Kafka** — Managed Kafka cluster with appropriate partitioning (by `subject` / patient ID) and replication for durability.
- **Compliance DB** — Relational database (e.g., PostgreSQL) with appropriate indexing on `patient_id`, `protocol_instance_id`, and temporal fields.
- **Network topology** — The CCE Gateway Service is the only service exposed on the public/external network. All backend services (Collector, Compliance, Analytics) run on an internal network accessible only from the CCE Gateway Service and from each other.

8. Open Issues

8.1 Brownfield Deployment — Existing Patient Enrollments

Problem:

CCE creates Protocol Instances when it receives an enrollment event that matches a PlanDefinition's enrollment action trigger (see Section 4.3.3.2). However, when CCE is introduced into a source system that is already running, many patients will already be enrolled in care programs — their enrollment events occurred before CCE existed.

When these patients' ongoing clinical events arrive (e.g., ANC Visit 4 for a patient enrolled 3 months ago), CCE has no Protocol Instance for them. The event enters the processing flow (Section 4.3.4.2), Protocol Instance Resolution finds no active instances, and the event falls into zero-match handling (Section 4.3.4.5, Scenario 5) — it is logged but no compliance tracking occurs.

Impact:

Until this is addressed, **only new program enrollments that occur after CCE deployment will be trackable**. Patients already enrolled in care programs at the time of

CCE onboarding will not have compliance tracking, intelligence generation, or analytics coverage — regardless of how many clinical events their source systems emit. For deployments with large existing patient populations, this represents a significant coverage gap during the initial rollout period.

system's ability to replay events, and the acceptable complexity for Emitter Adaptors. This decision should be made before the first production deployment.