



EDB Agent Governance

Release 1.0.0

EnterpriseDB Corporation

Jul 06, 2026

Contents

1	What you can do	3
1.1	Where to start	3
1.2	Using Agent Governance with Hybrid Manager	3
2	Key concepts	5
2.1	Airman MCP	5
2.2	Session	5
2.3	Step	5
2.4	Purpose	5
2.5	Instance	6
2.6	Cluster	6
2.7	Loki and LogQL	6
2.8	Sync watermark	6
3	Architecture and data flow	7
3.1	The data path	7
3.2	Component responsibilities	7
3.3	How agent activity is tagged	8
3.4	Log entry format	8
3.5	Query paths	9
3.6	Background sync	9
3.6.1	Sync pipeline	9
3.6.2	Adaptive chunk splitting	9
3.6.3	Retry policy	9
3.6.4	Resync	10
4	Deploying and configuring	11
4.1	Understanding the container stack	11
4.2	Prerequisites	11
4.3	Pre-deployment configuration	12
4.3.1	Required	12
4.3.2	Optional	12
4.4	Deploying with pre-built images	12
4.5	Managing identity providers	13
4.6	Securing and monitoring your deployment	13
5	Connecting data sources	15

5.1	Instance types	15
5.2	Adding an HM instance	15
5.2.1	Requirements for an HM instance to show session data	16
5.3	Adding a Loki instance	16
5.4	Removing an instance	16
5.5	Resyncing an instance	16
6	Browsing sessions	19
6.1	Finding the sessions list	19
6.2	Reading the sessions table	19
6.3	Searching sessions	20
6.4	Filtering by outcome	20
6.5	Configuring page size	20
6.6	Opening a session	20
7	Inspecting a session	21
7.1	Inspecting the page header	21
7.2	Viewing the step-by-step reconstruction	21
7.2.1	Collapsed step	21
7.2.2	Expanded step	22
7.2.3	Accordion mode	22
7.3	Viewing the flat audit log	22
7.3.1	Audit log columns	22
7.3.2	Searching the audit log	22
7.3.3	Configuring page size	22
8	Securing access and handling credentials	23
8.1	Authenticating users	23
8.2	Authenticating upstream sources	24
8.3	Storing credentials at rest	24
8.4	Protecting the network	24
8.5	Handling data	24

EDB Agent Governance is a standalone application that provides observability and oversight for AI agents operating on Postgres databases. It lets database administrators, security teams, and AI operations engineers monitor, audit, and reason about how AI agents interact with enterprise data.

The application connects to one or more upstream log sources — an EDB Hybrid Manager (HM) deployment or a standalone Grafana Loki endpoint — to retrieve the Postgres query logs that AI agents generate through the Airman Model Context Protocol (MCP) server. It presents those logs as structured sessions (logical groupings of SQL statements executed within a single AI agent interaction) with tools for filtering, inspecting, and auditing agent behavior. Because it reads from any qualifying log source, it isn't limited to HM-managed databases.

CHAPTER 1

What you can do

- **Reconstruct agent sessions** — See every SQL statement an AI agent executed, with full metadata such as duration, target database, Postgres role, and error severity.
- **Audit across multiple data sources** — Connect to several HM or Loki sources at once and audit clusters across all of them from one place.
- **Read a flat audit trail** — Scan every statement in a session as a single filterable table, optimized for review.
- **Rely on cached, incremental sync** — The application incrementally syncs and caches session data, so audit views load quickly without re-querying the full log history on every visit.

1.1 Where to start

- *Key concepts* — the key terms used throughout the application: sessions, steps, purpose, instances, and how Airman MCP tags activity.
- *Architecture and data flow* — how agent activity travels from Postgres logs into the application.
- *Deploying and configuring* — for administrators deploying and configuring the application.
- *Connecting data sources* — register an HM or Loki instance so its agent sessions appear.
- *Browsing sessions* — find and filter the AI agent sessions detected for a cluster.
- *Inspecting a session* — inspect a single session step by step and read its audit log.
- *Securing access and handling credentials* — authentication, credential storage, and how your data is treated.

1.2 Using Agent Governance with Hybrid Manager

If your Postgres clusters are managed by Hybrid Manager, see [Governance in Hybrid Manager](#) for the HM-specific integration details — machine user API keys, HM’s Loki pipeline, and project and cluster discovery.

CHAPTER 2

Key concepts

Review these terms to understand and use the AI governance audit log viewer effectively.

2.1 Airman MCP

Airman MCP is the Model Context Protocol (MCP) server that brokers Postgres access for AI agents. It exposes Postgres operations as tools the agents can call, and it tags every query it runs via the Postgres `application_name` variable in the form `airman:<purpose>/<session-short>`, so the activity can be reconstructed later. The viewer relies entirely on these tags to group and attribute agent activity.

2.2 Session

A session is a logical grouping of SQL statements executed by an AI agent within a single Airman MCP interaction. Each session is the primary unit of audit in the viewer: when you open a cluster, you see its sessions, and when you open a session, you see the statements it contains.

Airman MCP identifies a session with a session short — the first eight hexadecimal characters of the MCP session token. All log entries that share a session short belong to the same session.

2.3 Step

A step is a single SQL statement within a session — one parsed Postgres log record. Each step carries the SQL text and its execution metadata: target database, Postgres role, backend process ID, duration, and error severity.

2.4 Purpose

Purpose is a declared intent label configured per Airman MCP instance through the `AIRMAN_MCP_PURPOSE` environment variable (for example, `billing`, `support`, or `analytics`). It appears as the first segment of the tag Airman writes to each query and is surfaced in both the sessions list and session detail. Purpose is set on the agent side, not in the viewer.

2.5 Instance

An instance is a registered upstream data source — either a Hybrid Manager (HM) deployment or a standalone Loki endpoint. The viewer can connect to multiple instances at once, aggregating the clusters and sessions from all of them. See [Connecting data sources](#) .

2.6 Cluster

A cluster is a Postgres cluster managed by HM or monitored through Loki. The viewer lists clusters as the entry point for auditing: you pick a cluster, then review the AI agent sessions detected for it.

2.7 Loki and LogQL

Loki is the log aggregation system that stores the Postgres query logs the viewer reads. It is optimized for label-based queries expressed in LogQL, its query language. HM includes a Loki pipeline. You can also point the viewer at a standalone Loki instance that receives Postgres logs. The viewer's backend constructs LogQL queries to find the session-tagged log entries for a cluster.

2.8 Sync watermark

A sync watermark is a per-instance, per-cluster record that tracks how far back the viewer's backend has cached session data from Loki. It records the earliest known entry and how far the cache has been populated, which is what enables fast, incremental syncing instead of re-querying the full log history on every visit. See [Architecture and data flow](#) .

Architecture and data flow

AI agent activity travels from the SQL an agent runs, through Postgres logs and Loki, to the screens you read. Understanding the path helps you reason about what the viewer can show and why session data sometimes needs to sync before it appears.

3.1 The data path

Each query an AI agent runs passes through four stages before it appears as an auditable session in the viewer.

```
AI agent (Airman MCP)
| Tags each query: application_name = airman:<purpose>/<session-short>
▼
Postgres cluster
| Logs every statement with application_name, SQL, duration,
| error severity, database, role, and PID (JSON log format)
▼
Loki (HM Loki pipeline, or a standalone Loki instance)
| Stores log lines, labeled by container and cluster, queryable via LogQL
▼
Viewer backend
| Queries Loki via LogQL, parses Postgres log lines into sessions,
| caches session summaries, and injects upstream credentials server-side
▼
Viewer (browser)
  Renders clusters, sessions, and session detail
```

3.2 Component responsibilities

Three components handle distinct parts of the data path, with the backend mediating all upstream access so credentials never reach the browser.

Viewer (browser): The single-page application renders the clusters, sessions, and session-detail screens, manages client-side routing, and calls the backend for all data.

Viewer backend: A lightweight service that:

- Stores instance configuration and the associated upstream credentials.
- Routes each request to the correct HM or Loki endpoint based on the instance type.
- Parses raw Postgres JSON log lines into structured session and step data.
- Caches session summaries and syncs them incrementally to avoid redundant queries.
- Injects the appropriate authentication header on every proxied request, so credentials never reach the browser.

Upstream sources: Either an HM deployment — which exposes projects, clusters, and Loki log queries through its API and authenticates with a machine user API key — or a standalone Loki instance queried directly, with optional bearer-token authentication.

3.3 How agent activity is tagged

Airman MCP automatically sets the Postgres `application_name` session variable on every connection it opens:

```
SET application_name = airman:<purpose>/<session-short>;
```

Where:

- `purpose` — a static label configured per Airman instance through `AIRMAN_MCP_PURPOSE` (for example, `billing`, `support`, or `analytics`). Defaults to `_` when unset.
- `session-short` — the first eight hexadecimal characters of the MCP session token, used to group all the queries in one interaction.

The tag is set at the Postgres session level and persists for every subsequent statement on that connection.

3.4 Log entry format

With JSON logging enabled, Postgres produces entries like the following, which Loki ingests:

```
{
  "ts": "2026-05-22T14:30:45.123Z",
  "record": {
    "application_name": "airman:billing/a1b2c3d4",
    "message": "Duration: 45.2 ms  statement: SELECT customer_id, email FROM_
↪customers WHERE active = true",
    "database_name": "banking_db",
    "user_name": "agent_read",
    "command_tag": "SELECT",
    "error_severity": "",
    "log_time": "2026-05-22 14:30:45.123 UTC",
    "process_id": "12345",
    "session_id": "789",
    "sql_state_code": "",
    "query_id": "0"
  }
}
```

The backend extracts the SQL text and duration from the `message` field and maps the remaining fields directly to step metadata.

3.5 Query paths

The viewer issues different Loki queries depending on what you're viewing.

Sessions list: When you open a cluster, the backend reads its sync watermark, backfills any uncached history from Loki, loads the cached session summaries, appends a live query for the most recent day, and merges the results — sorting sessions by most recently seen.

Session detail: When you open a session, the backend issues a LogQL query scoped to that cluster and filtered to the session short, then parses each returned log line into an ordered list of steps.

The backend uses LogQL queries of this shape:

```
{container="postgres", cnp_cluster_id=~"<clusterId>.*"} |= "airman:"
```

For session detail, the session short replaces `airman:` as the line filter.

3.6 Background sync

To avoid expensive full-range Loki queries on every page load, the backend caches session summaries and keeps them current with a watermark-based incremental sync.

3.6.1 Sync pipeline

When a sessions list is requested for a cluster, the backend:

1. **Reads the watermark** — the per-instance, per-cluster record of the earliest known entry and how far the cache has been populated. On a first sync, both are zero.
2. **Finds the earliest entry** — on a first sync, searches backward over 30 days, then 90, then 365, to establish the full time range to cache. If none is found, it treats today as the earliest.
3. **Backfills history** — fills the cache from the earliest entry to today in one-day chunks, upserting session summaries and advancing the watermark as it goes. Progress is reported through the status endpoint.
4. **Loads cached sessions** — reads all cached summaries for the cluster.
5. **Fetches today's live data** — runs a single query for the last 24 hours. This data is always fresh and isn't cached.

The backend then merges cached and live sessions by session short, sums step counts, adjusts the first- and last-seen timestamps, and sorts by last seen.

3.6.2 Adaptive chunk splitting

High-verbosity clusters can exceed Loki's per-query limits. When Loki reports that a one-day chunk is too expensive, the backend splits the window in half and retries each half independently, down to a minimum chunk size of one hour. If a one-hour chunk still fails, it logs an error and skips that window.

3.6.3 Retry policy

Upstream Loki calls are wrapped in a retry policy: up to five retries with exponential backoff (starting at 2 seconds, doubling, capped at 30 seconds) for transient errors (502, 503, 504). Non-retryable errors (400, 401, 404, 500) fail immediately.

3.6.4 Resync

When you trigger a resync from the Configure page, the backend clears the cached sessions and watermarks for the instance, then starts a background job: for a Loki instance it re-syncs the single stub cluster; for an HM instance it discovers all projects and clusters and syncs them in parallel. The request returns immediately, and the instance's status advances from **Syncing** to **Done** (or **Error**) as the job runs. See [Connecting data sources](#) .

Deploying and configuring

Deploy and configure the EDB Agent Governance audit log viewer. For the authentication model and credential handling behind these settings, see [Securing access and handling credentials](#) .

4.1 Understanding the container stack

The audit log viewer deploys as a Dex-fronted stack of four containers:

Service	Role
<i>nginx</i>	Terminates HTTPS, serves the viewer's static assets, and routes API and single-sign-on requests to the right service so the browser talks to a single origin.
<i>bff</i>	The backend — OIDC client of Dex, instance management, and the HM/Loki proxy that parses logs and caches sessions.
<i>dex</i>	Identity federation — brokers login to your upstream identity providers.
<i>bootstrap</i>	A one-shot helper that generates secrets and the internal certificates the stack needs on first run.

A shared Postgres service backs the stack, hosting the backend's database for instance configuration and sessions, and the Dex database for identity provider connectors.

4.2 Prerequisites

- A container host with Docker 24 or later and Compose v2.
- A TLS certificate and key for your domain, from any certificate authority your users trust. A self-signed certificate is acceptable for local testing.
- At least one upstream data source to audit: an [HM instance with a machine user API key](#) , or a standalone Loki instance receiving Postgres logs from AI agent workloads.

4.3 Pre-deployment configuration

Set the following values before running the bootstrap helper. The bootstrap helper generates and persists everything else automatically.

4.3.1 Required

Setting	Required	Description
Public base URL	Yes	The public origin — scheme and host, no path and no trailing slash, for example <i>https://governance.example.com</i> . The OIDC issuer and redirect URIs derive from it.
Bootstrap admin email	Yes	The email of the initial break-glass administrator. Also the default admin allowlist.
Postgres password	Production	The password for the shared Postgres service.
Image registry	Production	The registry for the pre-built images.
Image tag	Production	The image tag — pin to a released version.

The values the bootstrap helper generates — the shared client secret, the session signing key, the token encryption key, and the bootstrap admin’s identifier and password hash — are written for you and don’t set by hand.

4.3.2 Optional

These settings are recommended for production deployments:

Setting	Default	Description
Database connection	Points at the shared Postgres service	The connection string for the backend's database.
Upstream egress to private ranges	Blocked	Whether registered instances may reach private network ranges. Allow only for on-premises deployments whose HM or Loki upstreams live on private IPs. See <i>Securing access and handling credentials</i> .
OIDC CA bundle	—	A CA bundle to trust a self-signed or private-CA OIDC issuer. OIDC TLS verification stays on; this adds trust rather than bypassing it.
Session idle timeout	2h	How long an idle session lasts before it expires.
Session absolute lifetime	12h	The maximum lifetime of a session regardless of activity.
Admin allowlist	The bootstrap admin email	The emails granted administrator access, in addition to anyone whose <i>groups</i> claim includes <i>governance-admin</i> .

4.4 Deploying with pre-built images

Use the pre-built EDB-published images — no need to build from source.

1. **Provide your TLS certificate.** Place your certificate and private key where the stack expects them (the full chain and the private key).
2. **Configure the deployment.** Copy the example environment file and set the required values — at minimum the public base URL and the bootstrap admin email, plus the image registry and tag (see *Required*).

3. **Generate secrets and start.** Run the production start script. It runs the bootstrap helper — which generates the shared secrets, session and encryption keys, the bootstrap admin’s identifier and password, and the internal certificates — and then brings the stack up.
4. **Sign in and federate.** Open the deployment in a browser and sign in as the bootstrap admin, using the password the bootstrap helper recorded. Then add your upstream identity provider under **Settings** → **Identity Providers** and switch day-to-day login to it.

Note

Pin the image tag to a released version rather than running *latest*, so deployments are reproducible.

4.5 Managing identity providers

Configure the identity providers your users sign in through in the audit log viewer under **Settings** → **Identity Providers**:

1. Add an OIDC, SAML, or LDAP connector for your organization’s identity provider.
2. After federating, switch day-to-day login away from the bootstrap admin to the federated provider, keeping the bootstrap admin as a break-glass account.
3. Restrict the admin allowlist to the people who hold administrator access.

4.6 Securing and monitoring your deployment

Keep the following in mind once the deployment is running:

- **Encrypt the data volume.** The shared Postgres volume holds instance access keys, sessions, and identity provider connector secrets. Keep it on encrypted-at-rest storage. If the volume is lost, the databases are recreated empty — instances and connectors must be re-added or restored from a backup. See [Securing access and handling credentials](#).
- **Protect the bootstrap credentials.** Treat the recorded bootstrap admin password as sensitive and remove the file once you’ve stored the password securely.
- **Keep upstream TLS verification on.** Don’t disable TLS verification for upstream HM or Loki sources in production.
- **Monitor liveness.** Dex exposes a health endpoint that gates startup of the backend and nginx. Observe overall liveness through the container status and nginx upstream errors.

Connecting data sources

Register one or more upstream data sources to make their AI agent sessions available in the audit log viewer. An instance is a registered upstream data source — either an EDB Hybrid Manager (HM) deployment or a standalone Grafana Loki endpoint. The viewer can connect to multiple instances at the same time, aggregating clusters and sessions across all of them. Instance configuration, including access keys, is stored server-side and never sent to the browser.

Manage instances on the **Configure** page, accessible from the home page's **Configure Instances** button or from **Settings** → **Configure instance** in the header.

5.1 Instance types

Type	Description	Authentication	Discovery
HM	EDB Hybrid Manager dataplane	Machine user API key, sent in the <i>x-access-key</i> header	Auto-discovers projects and clusters
Loki	Direct Loki log store	Optional <i>Authorization</i> header	Creates a single stub project and cluster automatically

5.2 Adding an HM instance

1. Open the **Configure Instances** page.
2. Select **Add New**.
3. Complete the form:
 - **Type** — HM.
 - **Name** — a descriptive label, for example `Production EU`.
 - **Base URL** — the HM API endpoint, for example `https://hm.example.com:8443`.
 - **Access Key** — the machine user API key. These keys typically start with `hmak_` or `baak_`.

4. Select **Add Instance**.

On success, the viewer's backend discovers projects and clusters from the HM instance, and those clusters appear on the home page within seconds.

5.2.1 Requirements for an HM instance to show session data

For an HM instance to surface AI agent sessions, the following must be true:

- Postgres statement logging is enabled on the target clusters (`log_statement = 'all'` or `log_min_duration_statement = 0`).
- The cluster is reachable over the network from the viewer's backend.
- AI agent workloads run through Airman MCP with tracing enabled, so that activity is tagged and discoverable.

You create the machine user API key in the HM admin portal. The viewer stores it server-side and never returns it to the browser. For more on credential handling, see [Securing access and handling credentials](#).

5.3 Adding a Loki instance

1. Open the **Configure Instances** page.
2. Select **Add New**.
3. Complete the form:
 - **Type** — Loki.
 - **Name** — a descriptive label, for example `Dev Loki`.
 - **Base URL** — the Loki HTTP endpoint, for example `http://loki.internal:3100`.
 - **Authorization** — optional. A bearer token or other authorization header value, if your Loki endpoint requires one.
4. Select **Add Instance**.

For Loki instances, the viewer creates a single stub project and cluster, and all Loki sessions appear under it.

5.4 Removing an instance

Each instance row on the **Configure** page has a **Delete** button. Selecting it shows a confirmation popover. On confirmation, the viewer removes the instance record, its cached session data, and its sync watermarks.

Removal is irreversible. Re-adding the instance requires re-syncing its history from upstream.

5.5 Resyncing an instance

The backend caches session summaries for fast retrieval. Force a full resync when upstream data was corrected or backfilled, when the cache looks inconsistent, or when the cache database was lost.

1. On the **Configure** page, locate the instance row.
2. Select the **Resync** button (the sync icon).
3. The backend clears the instance's cached data and begins a full background sync.

The instance shows its sync status as **Syncing** (with a spinner), **Done** (with a session count), or **Error** (with a message). The resync runs in the background, so you can continue using the viewer while it completes. For details on how syncing works, see [Architecture and data flow](#) .

Note

The **Resync** button is only visible when developer tools are enabled. Otherwise the backend syncs automatically on first access to a cluster.

Browsing sessions

Use the **Sessions List** to audit all AI agent sessions detected for a cluster. Each session represents a complete interaction — all the SQL statements an AI agent executed within a single Airman MCP session.

6.1 Finding the sessions list

The **Home** page lists every cluster across all configured instances. Each cluster card shows the cluster name, its status, and its project, and a search box filters the list. Select a cluster card to open its **Sessions List**.

If a configured instance fails to load, the **Home** page shows a per-instance error alert. If no instances are configured yet, it shows a zero state with a **Configure Instances** button. See *Connecting data sources* .

When you open a cluster, the **Sessions List** loads automatically and triggers a background sync if needed, so you see the latest data from the upstream source.

6.2 Reading the sessions table

The sessions table supports customizable column visibility:

Column	Default	Sortable	Description
Session ID	Always visible	Yes	Monospace identifier. Select it to open the Session Detail page.
Purpose	Visible	Yes	Declared purpose from the Airman MCP configuration.
Started	Visible	Yes	Timestamp of the first statement in the session.
Commands	Hidden	No	Tags showing the SQL command types in the session (SELECT, INSERT, UPDATE, and so on).
Steps	Visible	Yes	Total number of SQL statements executed.

continues on next page

Table 1 – continued from previous page

Column	Default	Sortable	Description
Errors	Visible	Yes	Count of statements with error severity. Shown as a red badge when greater than zero.

To change which columns appear, select the column visibility button (the gear icon) in the toolbar and use the checkbox list. Session ID is always visible and can't be hidden.

6.3 Searching sessions

The search field in the toolbar performs full-text matching across the session ID and purpose. Type a term and press **Enter**, or wait for auto-filtering. Use the clear button to reset the search.

6.4 Filtering by outcome

The toolbar's filter lets you narrow the list by outcome:

Filter	Options	Description
Outcome	OK, Issues	Show only sessions that completed cleanly, or only those that contain at least one error.

The filter is single-select — selecting a different option replaces the current one. Clear it to show all sessions again.

6.5 Configuring page size

The table shows 10 sessions per page by default, configurable to 10, 20, or 50, with standard previous and next navigation.

6.6 Opening a session

Select a session ID to open the *Inspecting a session* page, where you can inspect each SQL statement step by step and read the full audit log for that session.

Use the **Session Detail** page to inspect every SQL statement an AI agent executed, along with its execution metadata, in two views: a step-by-step reconstruction and a flat audit log.

7.1 Inspecting the page header

The header shows:

- **Session ID** — the session identifier, as a large monospace title.
- **Started** — the formatted timestamp of the first statement.
- **Purpose** — a tag showing the declared purpose, when one is available.
- **Steps** — a badge showing the total statement count.

7.2 Viewing the step-by-step reconstruction

Use the **Reconstruction** tab to step through the session chronologically. Each step is a single SQL statement with its full execution context, presented as an expandable accordion.

7.2.1 Collapsed step

Each collapsed step shows, at a glance:

- The step number, for example Step 1.
- A command tag for the SQL command type (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, and so on).
- An error severity tag, shown only when the statement errored (red for `ERROR`, `FATAL`, or `PANIC`).
- The timestamp and duration, right-aligned — for example `14:30:45 · 45ms`.

7.2.2 Expanded step

Selecting a step expands it to reveal the request panel, which is shown by default and contains:

- **SQL statement** — the statement, in a syntax-highlighted code block.
- **SQLSTATE code** — shown as a tag when it isn't 00000 (success).
- **Properties** — a grid of execution metadata:

Property	Description
Database	Target database name
User	Postgres role used
PID	Backend process ID
PG Session ID	Postgres session identifier
Query ID	Internal query plan ID
Duration	Execution time in milliseconds
application_name	The full Airman tag, for example <i>airman:billing/a1b2c3d4</i>

7.2.3 Accordion mode

By default, you can expand multiple steps at once. When **Accordion steps** is enabled in settings, expanding a step automatically collapses the previously open one — useful for sessions with many steps where screen space is limited.

7.3 Viewing the flat audit log

The **Audit Log** tab presents the same session data as a flat table, optimized for scanning and filtering across all steps at once.

7.3.1 Audit log columns

Column	Description
Time	Timestamp, monospace and right-aligned
SQL	Statement text, monospace, truncated with an ellipsis
Purpose	Declared purpose

7.3.2 Searching the audit log

The toolbar's search field matches across the SQL text, purpose, and timestamps. Type a term to filter the table, and clear it to show all steps again.

7.3.3 Configuring page size

The table shows 10 rows per page by default, configurable to 10, 20, or 50.

Securing access and handling credentials

Audit AI agents operating on Postgres clusters managed by Hybrid Manager (HM) or monitored through a standalone Loki instance. The audit log viewer reads Postgres query logs that may contain sensitive data and holds credentials for the upstream sources it audits. There are two distinct authentication layers, and all credentials live server-side — none are exposed to the browser.

Layer	Question it answers	Mechanism
User authentication	Who may sign in to the viewer?	Identity federation through Dex (OIDC)
Upstream authentication	How does the backend reach HM or Loki?	Machine user API keys

8.1 Authenticating users

Sign in through Dex, which federates to your upstream identity providers:

1. The backend is an OIDC client of Dex. Dex brokers login to the configured connector — OIDC, SAML, or LDAP — and reports which identity provider authenticated the user.
2. On success, the backend mints a server-side session: a signed cookie, with refresh and ID tokens encrypted at rest. Tokens never reach the browser.
3. A bootstrap admin exists before any identity provider is configured. Use it for first login, then add a federated identity provider and switch day-to-day login to it.
4. **Admin gating** — a user is an administrator if their email is in the admin allowlist, or if their `groups` claim includes `governance-admin`. Admin-only surfaces, such as identity provider management, are gated on this.

For configuring identity providers, see [Deploying and configuring](#).

8.2 Authenticating upstream sources

Authenticate to upstream sources using machine user API keys — shared credentials that grant access to an entire HM dataplane or Loki instance. These keys are:

- Created in the HM admin portal (for HM instances) or configured externally (for Loki).
- Stored in the backend's Postgres database.
- Injected server-side on every proxied request.
- Never included in responses to the browser — they're cleared on the server and omitted.

The backend adds the appropriate header based on the instance type:

Instance type	Header	Format
HM	<i>x-access-key</i>	The raw key value, for example <i>hmac_abc123...</i>
Loki	<i>Authorization</i>	The full header value, for example <i>Bearer token123...</i>

8.3 Storing credentials at rest

The backend maintains two credential stores in its Postgres databases:

- The viewer's database holds instance access keys and user sessions.
- The Dex database holds identity provider connector secrets, such as OIDC client secrets and LDAP bind passwords, in cleartext.

Keep the underlying storage volume on encrypted-at-rest storage — LUKS or dm-crypt, cloud disk encryption, or an equivalent. The Dex connector credentials in particular have no application-layer encryption, so volume-level encryption is the protection that matters.

8.4 Protecting the network

The viewer enforces several network-level protections.

TLS verification. OIDC TLS verification — for discovery, JWKS, and token exchange — is always on and can't be disabled. To trust a self-signed or private-CA issuer, supply a CA bundle, which is added to the system trust pool rather than bypassing verification.

Upstream egress controls. By default, the backend blocks instance egress to private network ranges so that a registered instance can't be used to probe your internal network. Cloud metadata, loopback, and link-local addresses stay blocked regardless. Allow private ranges only for on-premises deployments whose HM or Loki upstreams legitimately live on private IPs.

Cross-origin requests. The viewer and its backend are served from a single origin in normal operation, so cross-origin access to the backend API is restricted to the origins you explicitly allow.

8.5 Handling data

Review Postgres query logs that include the SQL text agents executed, along with execution metadata such as the target database, Postgres role, and duration.