ROKKS

SOLID INTELLIGENCE

ROKKS End User Software Manual

English edition · Documented from the configured beta product

12 September 2026

DRAFT / Review edition

Document information

DOCUMENTED SOFTWARE

ROKKS beta

EXACT WEB PRODUCT COMMIT

`f7b7740e7371bad8317a547dd2608ac2706fefd7`

EXACT ANDROID BETA COMMIT

`ed5d9494be9e5d3919d0c34d9d45440457e96ca3`

DOCUMENTATION DATE

12 September 2026

This is a draft review artifact. Product captures and source-authentic mockups are labelled throughout the manual.

CONTENTS

Table of contents

1. Getting Started — 7 topics

1.1 Platform Overview
1.2 Quick Start
1.3 First Login
1.4 Navigation
1.5 Environments
1.6 Tenants
1.7 Glossary

2. Data Fabric — 11 topics

2.1 File Areas
2.2 Uploads
2.3 Transform Plans
2.4 Schema Editor
2.5 Web API Connections
2.6 Web API Streams
2.7 App Connect
2.8 Streaming Connectors
2.9 Local Tables
2.10 Analytic Warehouse
2.11 Lost and Found

3. Dashboards — 13 topics

3.1 Starred and Attention
3.2 Workspaces
3.3 Widget Collections
3.4 Blueprints
3.5 Reports
3.6 Widget Builder
3.7 Data Source Connector
3.8 Charts
3.9 Tables
3.10 Dashboard Filters
3.11 Drilldowns
3.12 Version History
3.13 Styles

4. AI Assistance — 2 topics

4.1 Sidekick
4.2 AI Control

5. Operations — 6 topics

5.1 Job Queue
5.2 Alerts
5.3 Service Stack
5.4 Monitoring and Logs
5.5 Backup & Restore
5.6 Updates

6. Administration — 6 topics

6.1 Access Inventory
6.2 Account Settings
6.3 Companies and Teams
6.4 Taxonomy
6.5 Preferences
6.6 Advanced Platform Tools

7. Mobile — 4 topics

7.1 Android App
7.2 Mobile Dashboards
7.3 Mobile Alerts
7.4 Mobile Settings

8. Tutorials — 5 topics

8.1 Pagila Maiden Flight
8.2 CSV to Dashboard
8.3 Web API Stream to Alert
8.4 App Connect Dashboard
8.5 Tenant Onboarding

9. End User Reference — 6 topics

9.1 Feature Matrix
9.2 Worker Job Types
9.3 Widget Capabilities
9.4 Mobile Capabilities
9.5 Data Types
9.6 Errors

SECTION 1

Getting Started

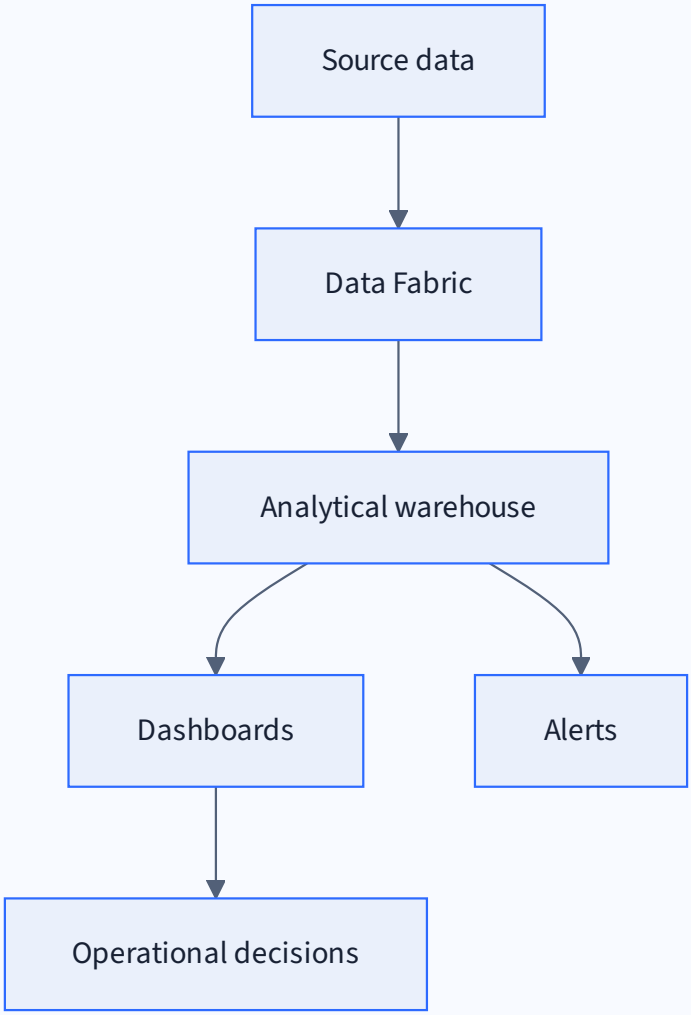
7 topics

Platform Overview

Overview

Rokks connects data sources, transforms them into governed tables, and presents them as dashboards, alerts, and reusable analytical workflows. Users work in the browser. Integrators operate the platform through documented deployment, authentication, scoping, API, and job concepts.

How it works



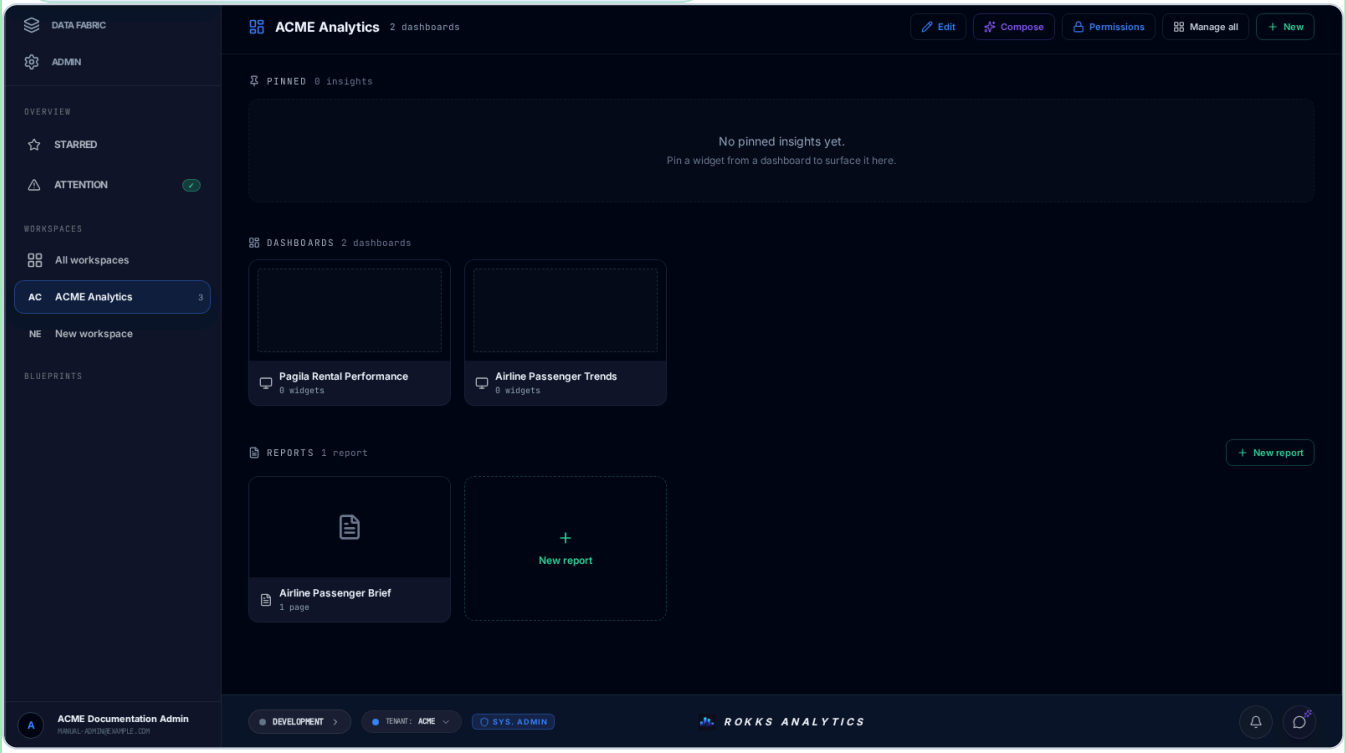
The Data Fabric is where data enters the system. A source can be a file, a web endpoint, an application connector, or an existing database table. Rokks profiles the source, lets you define or review metadata, and loads the result into a governed analytical layer.

Dashboards are the primary user-facing output. A dashboard can contain charts, tables, scorecards, maps, and other widgets. The Widget Builder helps users select data, choose fields, apply filters, preview results, wait for the instant save, and close the editor without writing code.

The operations area keeps long-running work visible. Jobs, alert evaluations, tenant operations, and service health checks are tracked so users do not have to guess whether the platform is still working.

The Android app is the mobile companion for read-focused work. It lets users view dashboards, review alert events, and manage mobile session settings while heavier authoring and administration stays in the web app.

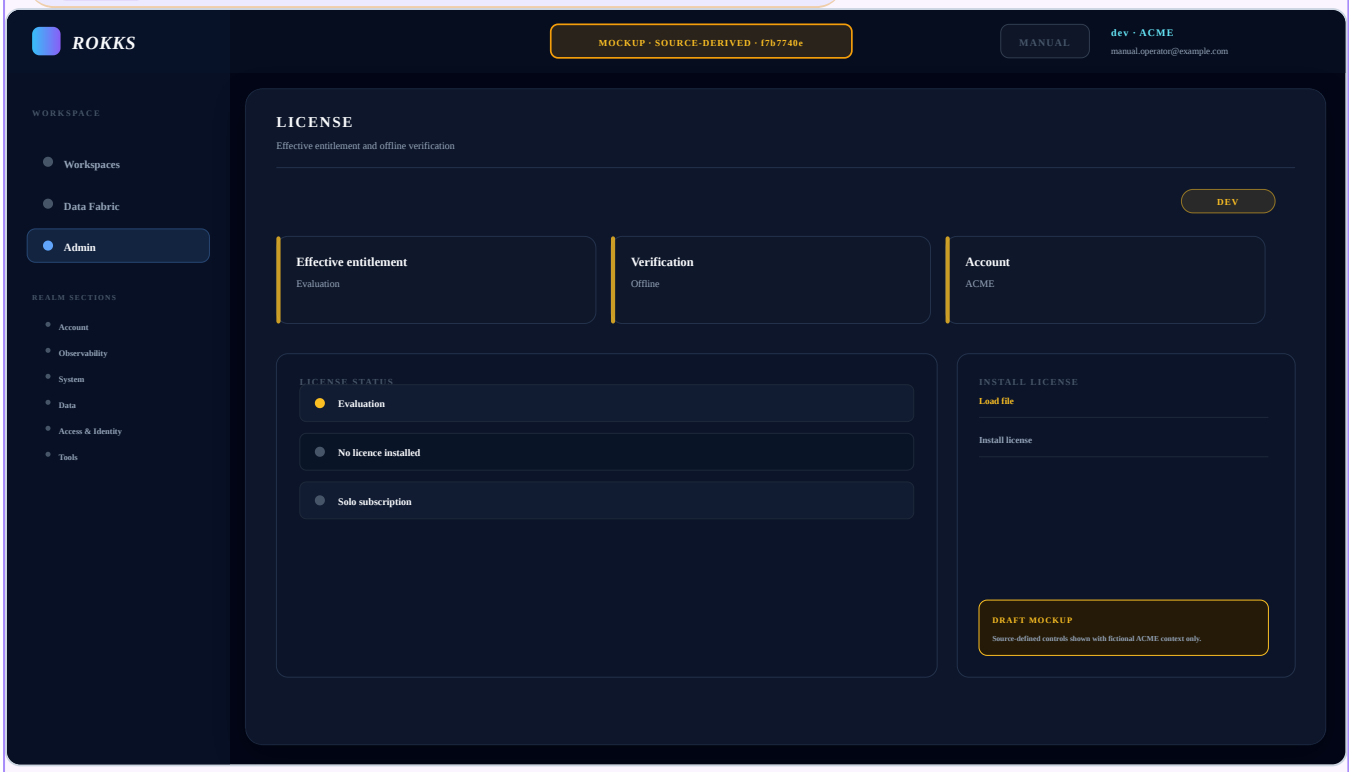
BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



ACME Analytics workspace with two starter dashboards and a one-page report. Dashboard widgets and pinned insights have not been added yet.

Captured 12 Sept 2026 UTC [Open documented view ↗]

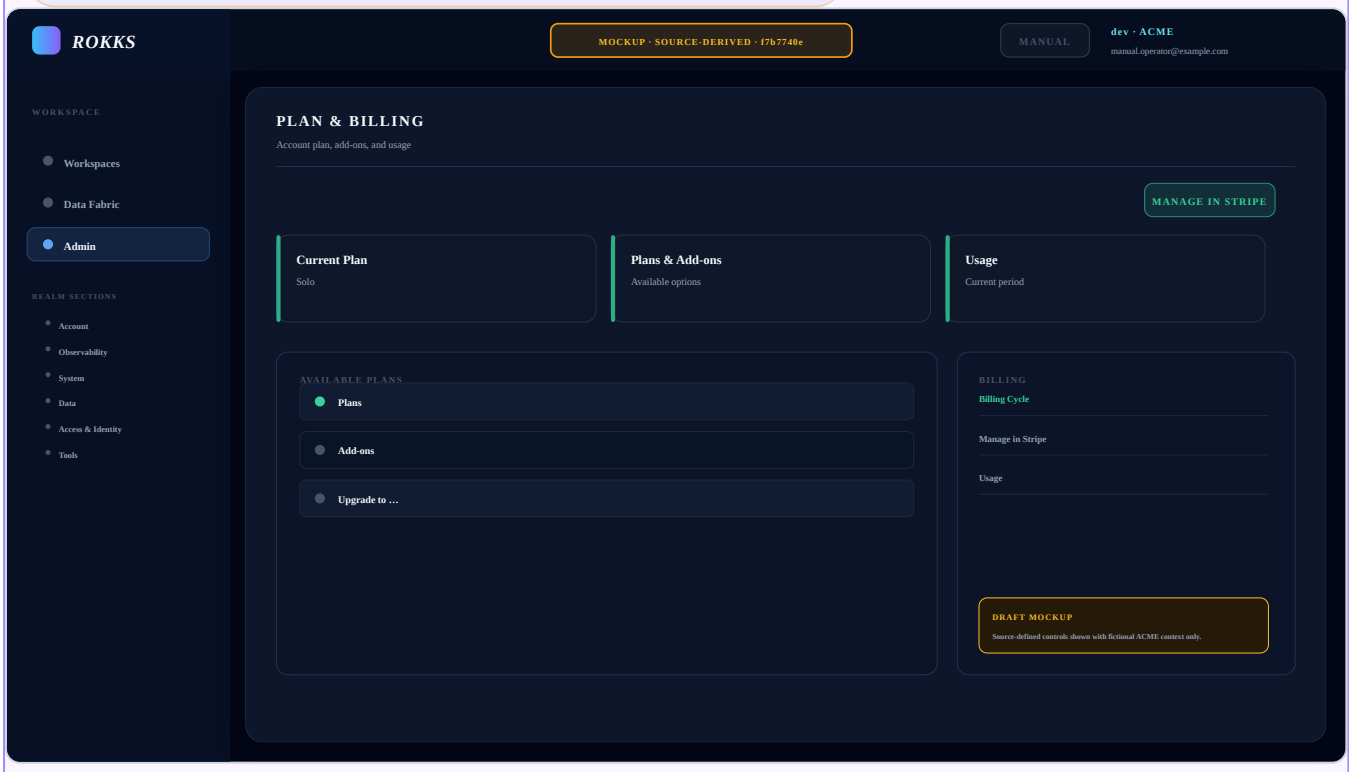
MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



License illustration showing effective entitlement and offline installation controls.

Source baseline: beta f7b7740e

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Plan and Billing illustration showing plan, add-on, cycle, and usage areas.

Source baseline: beta f7b7740e

Step-by-step

1. Choose the active environment and tenant.
2. Bring data into the Data Fabric.
3. Review the generated schema and metadata.
4. Build a dashboard from the loaded data.
5. Monitor jobs and alerts after the workflow is running.

Common mistakes

The most common mistake is treating a dashboard widget as the source of truth. Widgets are views over governed data. Fix source data, metadata, schedules, and filters in the relevant Data Fabric or operations page, then let dashboards update from that source.

Another common mistake is switching environments without checking the tenant selector. Environment and tenant together define the scope of data, jobs, dashboards, and configuration.

Related

- Quick Start
- Environments
- Tenants
- Android App
- Architecture

Quick Start

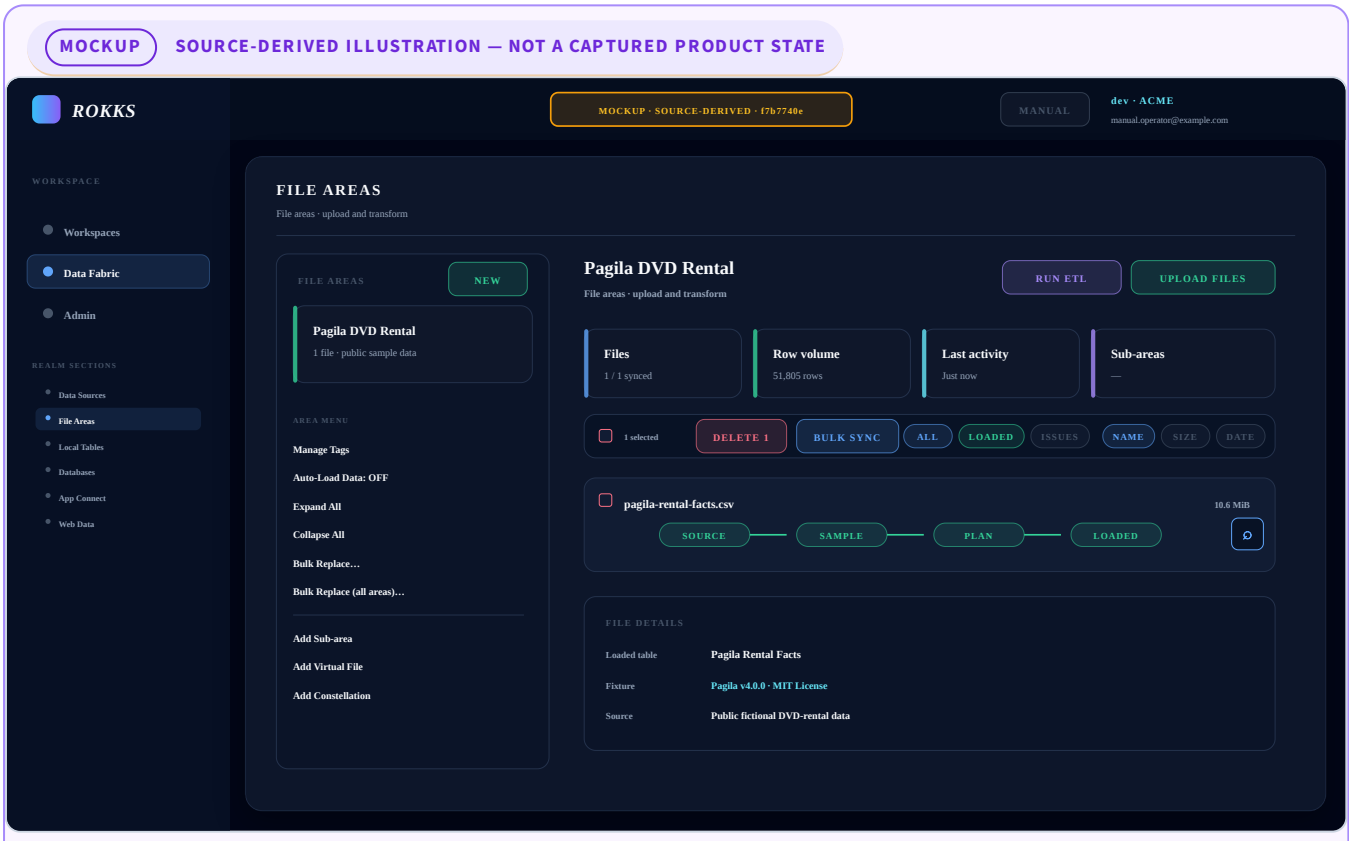
Overview

This quick start summarizes the smallest useful Rokks workflow: load the public fictional six-table Pagila sample, review its relationships, and build a dashboard widget. It assumes your platform is already available at your organization's Rokks URL. Follow the Pagila maiden flight for exact checksums, mappings, acceptance totals, and illustrated checkpoints.

How it works

A regular File Area groups related uploads. Its **Constellation** child accepts one JSON source and turns that source into several related governed tables. The Pagila source produces six nodes and five declared relationships. After the data is loaded, the Widget Builder reads the catalog and lets you follow those relationships across the available tables and columns. The dashboard never stores copied rows; it queries the governed data whenever it renders.

File Areas owns the imported outputs. **Local Tables** provisions tables created directly in Rokks and does not list Constellation outputs.



Data Fabric illustration based on the File Areas surface. Source baseline: beta f7b7740e

Step-by-step

1. Download the Pagila Rental Constellation JSON and its source and derivation record.

2. Open **Data Fabric → File Areas**, click **New**, and name the regular area `Pagila Samples`.
3. Open that area's action menu, choose **Add Constellation**, and name the child `Pagila Rental Constellation`.
4. Select the child, click **Upload Source**, and choose `pagila-rental-constellation.json`.
5. Wait for sampling, choose **Generate Plan**, and confirm six nodes named `categories`, `stores`, `customers`, `films`, `rentals`, and `payments` plus five relationships. Click **Save Constellation**.
6. Click **Run** and wait until the Constellation reports **6/6 loaded**. If you can access **Job Queue**, also confirm its child `FILE_ETL` job is **DONE**.
7. Open **Workspaces**, enter or create `ACME Analytics`, create the dashboard `Pagila — Rental Performance`, and choose **Build it by hand**.
8. Choose **Add column** to create the first empty lane, then use the slot labelled **Add or drag widget here**.
9. In the schema canvas, follow the catalog relationship `payments.rental_id → rentals.rental_id`. Configure **Monthly Revenue** as `SUM("payments"."amount")` by month of `"rentals"."rental_date"`, then click **Run Query**.
10. When the preview is correct, wait for the instant save to finish, then click **Close**.

If you prefer a chat-driven path, the Sidekick task cookbook shows how to request the workspace, dashboard, and first widget while keeping each result reviewable.

Common mistakes

Do not close the job queue because a job is running. Jobs continue in the background, but the queue is the best place to see progress, errors, and completion status.

Do not skip the schema review. Good display names, types, units, and constraints make dashboards easier to build and easier for Sidekick to understand.

Related

- File Areas
- Widget Builder
- Job Queue
- Pagila Maiden Flight

First Login

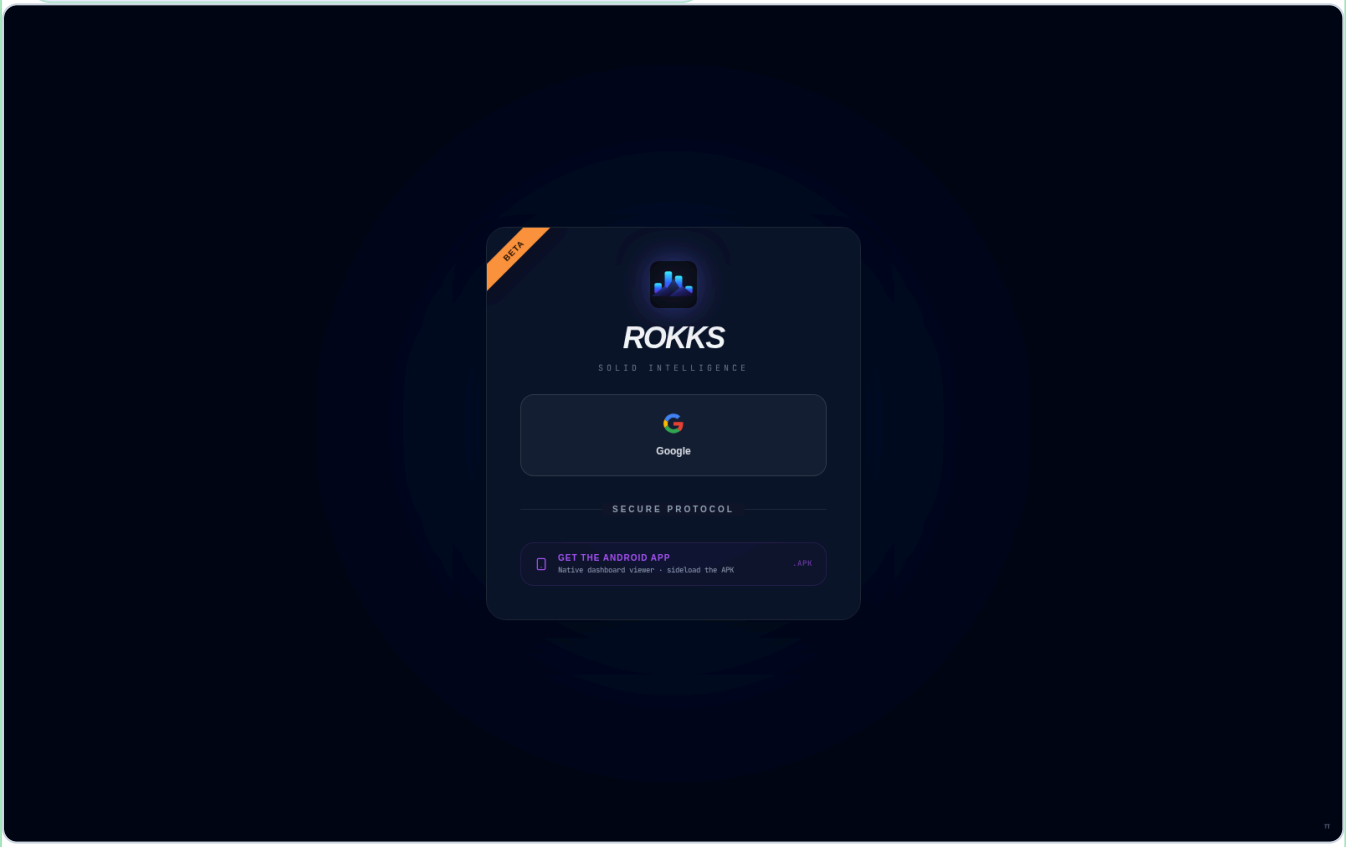
Overview

Your first login establishes who you are, which environments you can access, and which tenant workspace is active. Rokks uses that context for documents, dashboards, jobs, alerts, and analytical queries.

How it works

After authentication, the app loads your profile, available environments, tenant choices, and current preferences. The header shows the active scope. If the scope is missing or wrong, stop before uploading data or editing dashboards.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Configured beta login screen. No customer or account data is visible. Captured 12 Sept 2026 UTC Open documented view ↗

This verified beta deployment offers Google sign-in and an Android app download. Your organisation may enable different sign-in methods, and the download appears only when that deployment serves it.

Step-by-step

1. Open the Rokks application URL provided by your administrator.
2. Sign in with the configured identity provider.
3. Confirm your name or account indicator in the header.

4. Select the intended environment.
5. Select the intended tenant.
6. Open **Workspaces** or **Data Fabric** and confirm that the active scope contains the expected content.
7. Open the **Manual** drawer if you need context for the current page.

Common mistakes

Do not assume that a blank workspace means data is missing. You may be in a different tenant or environment. Check the scope selector first.

Do not share screenshots that show real tenant names, customer records, or account identifiers unless your organization allows it.

Related

- Navigation
- Environments
- Tenants

Navigation

Overview

Rokks groups the app by workflow. The left navigation has three top-level realms: **Workspaces**, **Data Fabric**, and **Admin**. The profile menu contains **User Settings** and **Preferences**. The header shows scope, assistant access, the Manual drawer, and page-level actions.

How it works

The active view controls the center workspace. Dashboards and collections are resource pages, while administration pages are fixed tools. The Manual drawer follows the active view and opens the most relevant documentation page when one exists.

Step-by-step

1. Open **Workspaces** to browse workspace groups, dashboards, reports, and blueprints.
2. Open **Data Fabric** to connect, load, repair, transform, or inspect data.
3. Open **Admin** for jobs, backups, monitoring, service health, identity, tenants, taxonomy, and system settings.
4. For alert rules, open **Workspaces** → **Attention** → **Manage rules**.
5. Open the profile menu for **User Settings** or **Preferences**.
6. Click **Manual** in the header for contextual documentation.

Common mistakes

Do not configure a data source from a dashboard widget. Widgets consume prepared sources; source ownership lives in the Data Fabric.

Do not troubleshoot a stuck spinner from the dashboard alone. Open the Job Queue or Monitoring pages to see whether background work is still active.

Related

- Quick Start
- Job Queue
- Service Stack

Environments

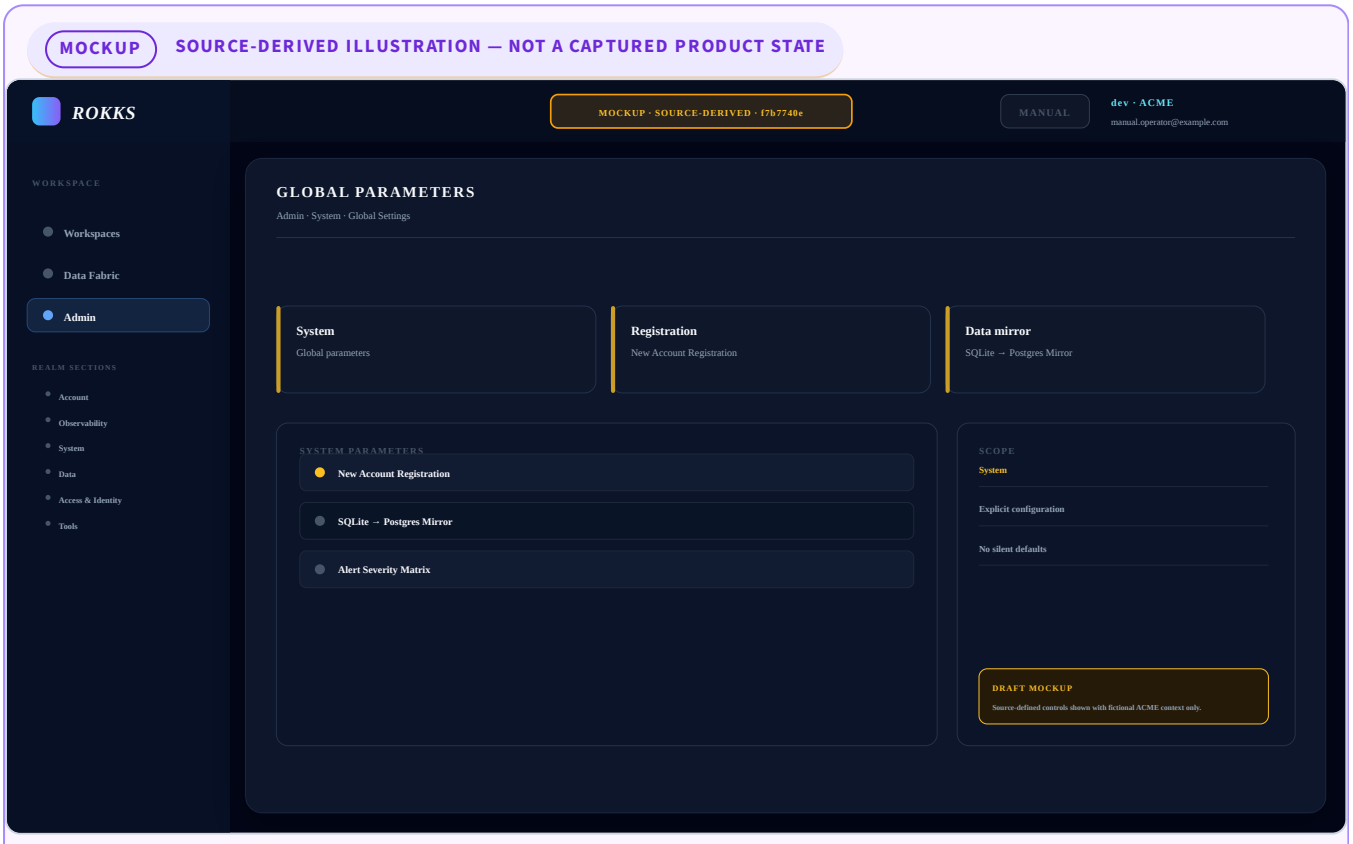
Overview

An environment is a platform-level workspace such as production, staging, or a demo area. It controls which backend configuration, storage, database, jobs, and service endpoints are used.

How it works

Every data operation runs inside one active environment. The same user may have access to several environments, but Rokks never treats them as interchangeable. Data loaded in one environment does not automatically appear in another.

Integrators use environments to separate lifecycle stages and infrastructure targets. Users use the environment selector to avoid editing the wrong workspace.



Global Parameters illustration showing representative system configuration cards. Source baseline: beta f7b7740e

Service and persistence scope

Shared application services serve every enabled environment. Storage is the explicit exception: each environment selects its PostgreSQL and document-store instance in Service Stack configuration. The environment record is the only owner of those bindings; there is no second service-assignment matrix or instance-level environment list.

Underlying database identifiers are implementation-managed and are not a stable end-user contract. Registry connection secrets are authoritative. The environment's data and platform PostgreSQL DSNs must target the exact database assigned by the platform; do not derive, substitute, or repair a database name from the environment display name or ID. Runtime services use separate least-privilege data and platform identities; the privileged database identity is reserved for provisioning rather than ordinary dashboards and imports.

Display identity and storage routing

An environment has two deliberately separate identifiers:

Field	Purpose	Example
Display name	Human-facing label in selectors and screens	`Development`
Document routing key (`firestoreNamespace`)	Technical deployment configuration	`TEST`

ROKKS does not elect a default environment and does not invent a `default` document namespace. If routing configuration is absent, the environment remains explicitly unconfigured instead of reading from a plausible fallback. Display names never alter storage routing. Changing a technical routing key is therefore an explicit data migration, not a rename shortcut.

Step-by-step

1. Check the environment selector before starting work.
2. Use production for live business workflows.
3. Use staging or demo environments for testing sources, dashboards, and alert rules.
4. Confirm that jobs and dashboards you expect are visible after switching.
5. Ask an administrator if an environment is missing from your selector.

Common mistakes

Do not copy values from one environment into another without understanding the target tenant. Environment and tenant together define the scope.

Do not test destructive operations in production. Use a non-production environment when validating imports, purge flows, or schema changes.

Related

- Tenants
- Deployment Topologies
- Environment and Tenant Scope

Tenants

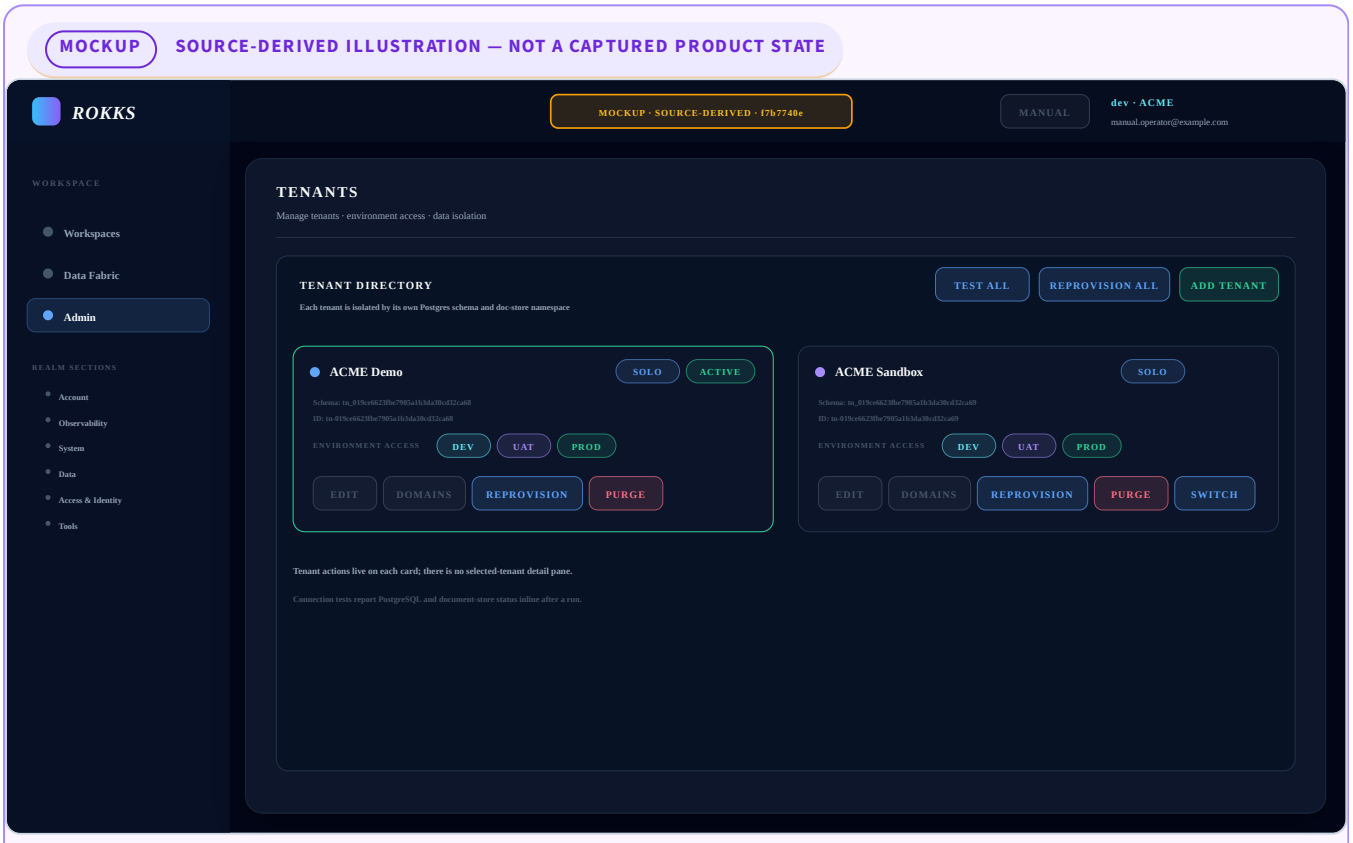
Overview

A tenant is a business workspace inside an environment. Tenants separate dashboards, files, metadata, jobs, alerts, preferences, and most user-facing data. Use tenants for customers, business units, or isolated teams.

How it works

Rokks always evaluates tenant-aware work in the selected tenant. A file upload, dashboard save, alert rule, or accepted job submission belongs to that tenant. Accepted tenant operations such as purge run as tracked jobs so administrators can see progress and failures.

Tenant export, archive import, and browser-to-browser transfer are currently paused. The former export sent tenant records and blobs through the general job-status channel. The former browser import inflated and retained an unrestricted ZIP, then duplicated it before backend limits applied. A dedicated, authorized streaming replacement must enforce snapshot correctness, byte and memory budgets, backpressure, and cancellation in both directions.



Tenant administration illustration based on the documented beta source.

Source baseline: beta f7b7740e

Step-by-step

1. Select the active tenant before editing data.
2. Create or request a tenant for each isolated customer or workspace.

3. Assign users only to tenants they should access.
4. Do not use Tenant Transfer as a migration procedure while its export, import, stream, and receiver paths are paused.
5. Use purge workflows only when the tenant should be permanently removed.

Common mistakes

Do not use one tenant for unrelated customers just because their dashboard designs look similar. Tenant boundaries preserve data isolation; visual similarity is not a reason to combine data scopes.

Do not purge a tenant to work around a migration problem. Fix or delete the affected source first.

Do not treat a transfer error as a queued background job. Paused export and import submissions are rejected before a job record is created.

Related

- Tenant Onboarding
- Environment and Tenant Scope
- Backup and Restore

Glossary

Summary

This glossary defines product terms that appear across the user manual and integrator guide.

Fields / Parameters

Term	Type	Description
Environment	Scope	Platform-level deployment or lifecycle workspace.
Tenant	Scope	Isolated business workspace inside an environment.
Data Fabric	Feature area	Where files, APIs, apps, and tables become governed sources.
File Area	Source	A workspace for related uploads. In a regular area, each file can load to its own physical table; a Virtual File combines several files into one table, while a Constellation represents several related tables.
Transform Plan	Configuration	The mapping from raw source fields to governed output fields.
Catalog	Metadata	The current list of tables, fields, display names, types, and units.
Dashboard	Workspace	A canvas containing widgets for analysis and monitoring.
Widget	Dashboard item	A chart, table, map, scorecard, or other visual component.
Sidekick	Assistant	The AI assistant that helps build and refine dashboards.
Worker Job	Background work	A tracked operation such as sync, load, generation, compile, or purge.

Example

```
Choose an environment and tenant, load a source in the Data Fabric, then build
widgets from the catalog.
```

Related

- Platform Overview
- Feature Matrix
- Worker Jobs

SECTION 2

Data Fabric

11 topics

File Areas

Overview

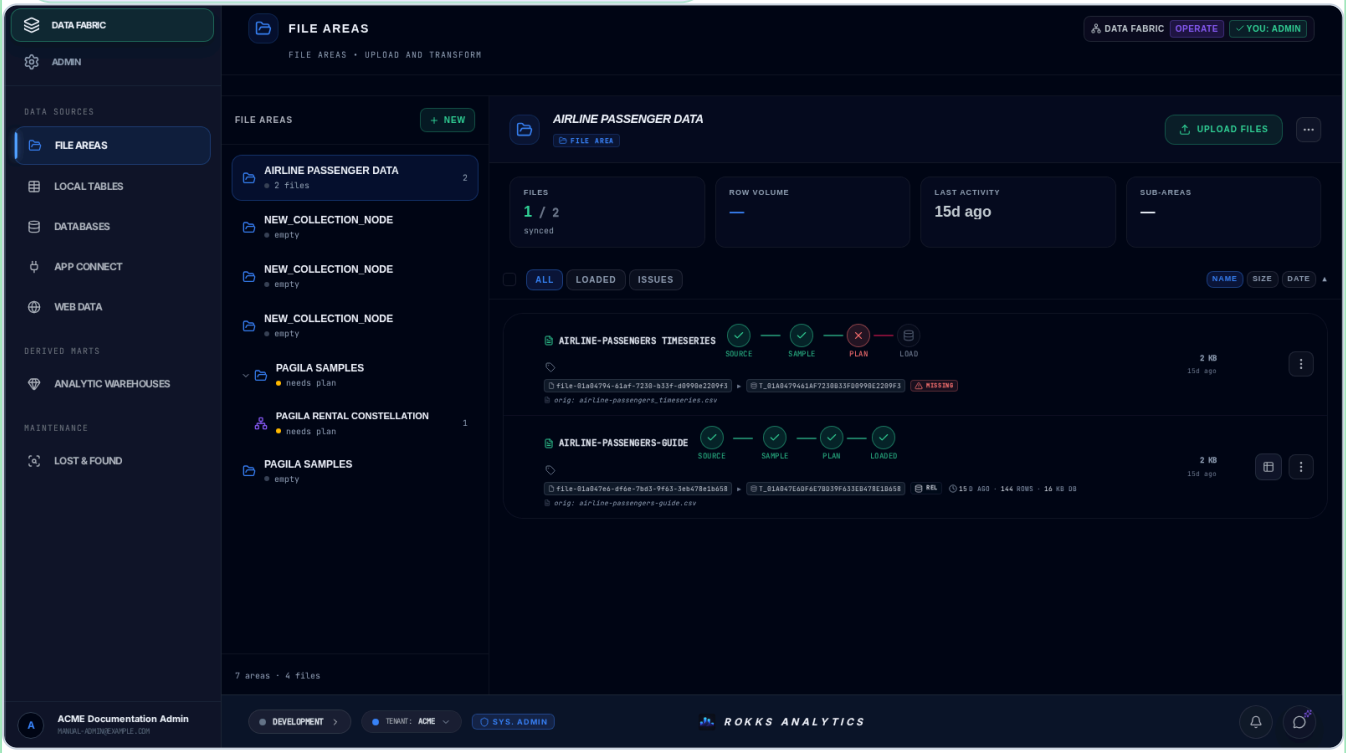
A file area is a workspace for related uploads. Use it for recurring CSV or delimited text, XLS/XLSX spreadsheet, or JSON files that belong together. Parquet is not decoded by the current File Areas parser.

How it works

When you upload a file, Rokks samples the data and proposes a transform plan. The plan describes display names, data types, mappings, units, and constraints. In a regular **AREA**, each file can load to its own physical table. Use a **Virtual File** when several files must combine into one table, or a **Constellation** when the source represents several related tables. After a successful load, the resulting table or tables become available in the catalog and can be selected in the Widget Builder.

A Constellation is a leaf container inside a regular File Area. It accepts one JSON source, owns one multi-table plan, and displays each governed result under **Output tables** with its own **Inspect** action. The same tables are available through **Analytic Warehouse** → **Data Source Tables** and **Relationship Map**. They are File Areas outputs, so **Local Tables** does not list them.

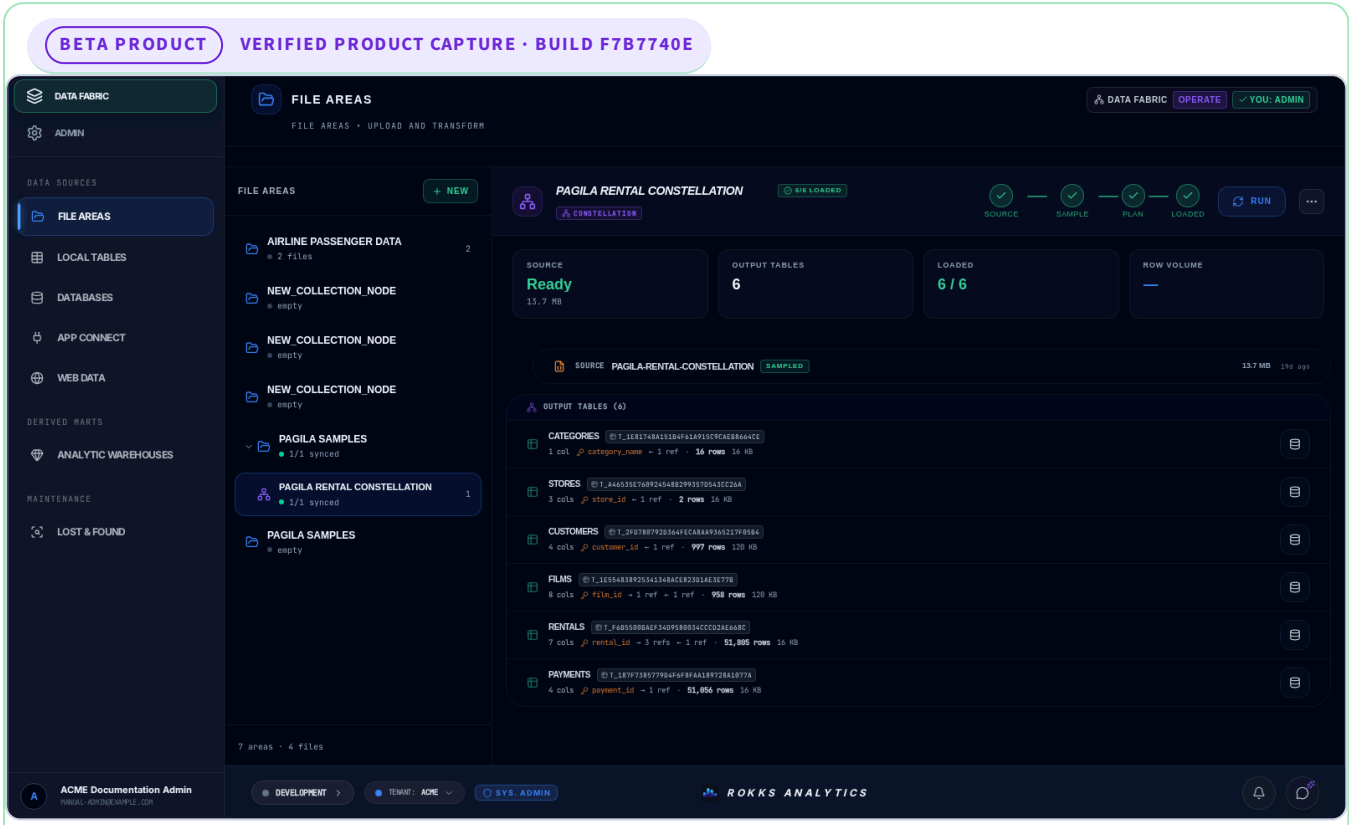
BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



ACME File Areas shows the airline guide loaded with 144 rows. The other airline file and Pagila constellation still need a plan; this is not a completed Pagila import.

Captured 12 Sept 2026 UTC [Open documented view ↗]

The first capture contrasts a loaded airline sample with files still awaiting a plan. The second shows the real six-table Pagila rental projection after loading: all output tables are ready, with 51,805 rental rows and 51,056 rental-level payment aggregates.



Real beta File Areas output: six loaded tables, including 51,805 rentals and 51,056 rental-level payment aggregates.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open **Data Fabric**.
2. Create a file area with a clear business name.
3. Upload one or more related files. For one-table output from several files, create a Virtual File instead of relying on ordinary area grouping.
4. Review each proposed transform plan, or configure the Constellation when the source contains related tables.
5. Adjust types, labels, units, and mappings.
6. Run the load and monitor the job.
7. Build dashboards from the resulting catalog entries.

Sidekick can perform the same attachment-to-plan sequence in **Agent** mode. Use the File Area and constellation cookbook when you want a chat-driven path with an explicit plan-review stop before loading.

Common mistakes

Do not assume that putting several files in one regular area merges them. Ordinary files retain separate table outcomes; use a Virtual File for N-to-one output or a Constellation for a related multi-table model.

Do not rename fields only inside a dashboard. Fix display names in the source metadata so every widget and assistant workflow sees the same names.

Related

- Uploads
- Transform Plans
- Local Tables
- Widget Builder

Uploads

Overview

Uploads bring file-based data into a file area. A file can be staged, profiled, transformed, loaded, and monitored without making the browser responsible for the heavy work.

How it works

The browser sends the file to the platform, then background jobs handle parsing and loading. Progress appears in the UI as a status badge and in the Job Queue as a detailed job record. If a load fails, the file keeps an error state so you can inspect and retry it.

Step-by-step

1. Open the target file area.
2. Drop files into the upload zone or choose them from your file picker.
3. Wait for the upload state to finish.
4. Review the transform plan before loading.
5. Start the load only after the plan matches the data.
6. Open the Job Queue if progress looks unclear.

Common mistakes

Do not upload a corrected replacement file into a file area without checking whether the old file should stay. If the data represents a replacement rather than an addition, use the appropriate refresh workflow.

Do not ignore type warnings. A date read as text or a number read as text will limit charting, sorting, filtering, and alerting later.

Related

- File Areas
- Transform Plans
- Job Queue

Transform Plans

Overview

A transform plan explains how source fields become governed output fields. It is the review step between raw data and reusable analytics.

How it works

Each target field has a display name, type, mapping rule, optional unit, and optional constraints. Rokks can suggest a plan from sample data, but the user remains responsible for confirming the business meaning. Once loaded, downstream pages read the resulting catalog metadata rather than the raw plan.

Step-by-step

1. Open a file area or source that has sampled data.
2. Review every proposed target field.
3. Rename fields to business-friendly labels.
4. Confirm date, number, boolean, enum, and text types.
5. Set units where measures need interpretation.
6. Mark key fields when they identify records.
7. Save or run the load.

Common mistakes

Do not accept vague names such as `Column 1` or `value` when the field has a business meaning. Good names improve dashboards and assistant suggestions.

Do not use a text type for values that need sorting, aggregation, time filtering, or thresholds.

Related

- Schema Editor
- Uploads
- Data Types

Schema Editor

Overview

The Schema Editor lets you correct and enrich table metadata after a source has been loaded. Use it when a field name, type, unit, relationship, or constraint needs to be improved for future analysis.

How it works

Dashboards, filters, Sidekick, alerts, and integrator-facing metadata all rely on the catalog. The Schema Editor updates that catalog so the change is visible everywhere. A display rename changes what users see; it does not require reloading the source data.

Step-by-step

1. Open the loaded source in Data Fabric.
2. Open the schema or metadata panel.
3. Review field labels, types, units, and key flags.
4. Update incorrect or unclear metadata.
5. Existing metadata edits persist automatically; wait for the persistence state to settle. When creating a new field, use **Commit** to create that field.
6. Reopen a dashboard widget or source selector to confirm the improved labels.

Common mistakes

Do not create a new upload just to rename fields. Use metadata editing when the data itself is correct.

Do not leave important measures without units. A revenue, duration, count, or percentage field is easier to trust when the unit is explicit.

Related

- Transform Plans
- Data Source Connector
- Data Types

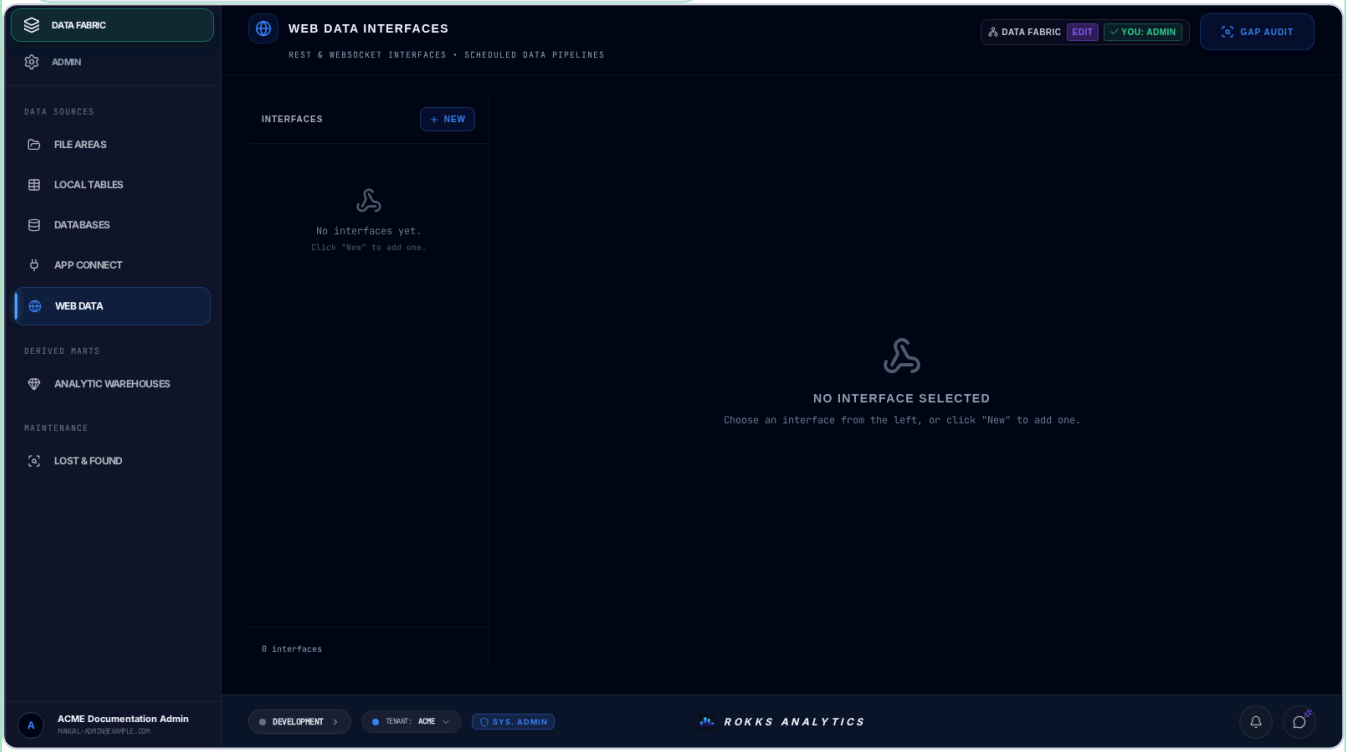
Web API Connections

Overview

A Web API connection stores the reusable connection details for one external data provider. Streams under the connection define which endpoints or resources to sync.

How it works

The connection describes how Rokks can reach and authenticate to the provider. A stream describes the data shape, extraction strategy, sync mode, and target table. Heavy extraction and loading runs as Dispatcher-scheduled background jobs so a browser tab can close without corrupting the source state.



Web Data Interfaces in ACME before a connection has been added. Use New to begin; this capture does not show a configured API or a completed sync.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open **Web Data** in the Data Fabric.
2. Create a connection and name the provider clearly.
3. Enter the base request details supplied by the provider.
4. Test discovery or probe the endpoint if available.
5. Add one or more streams.
6. Configure each stream’s transform plan and sync mode.
7. Configure a schedule, then validate the first Dispatcher-submitted sync in the Job Queue.

Manual Web API **Sync** and **Full Sync** actions are currently unavailable in the UI and Sidekick. The supported execution path is the schedule owned by Dispatcher; the Worker does not accept a browser-rich `WEBAPI_SYNC` payload.

Common mistakes

Do not put several unrelated providers into one connection. Keep connection names aligned with the provider or account.

Do not assume a successful endpoint probe proves that the complete stream can load. Review the first scheduled job and its output before relying on the stream in a dashboard or alert.

Related

- Web API Streams
- Worker Jobs
- Connector Lifecycle

Web API Streams

Overview

A Web API stream is a specific dataset retrieved from a connection. Use streams for resources such as orders, tickets, events, measurements, or inventory snapshots.

How it works

Each stream has extraction settings, a transform plan, a target table, and a sync mode. A full refresh replaces the source's table contents. An incremental stream uses a cursor or timestamp strategy to fetch only new or changed records when the provider supports it.

Step-by-step

1. Open the parent Web API connection.
2. Create a stream for one endpoint or resource.
3. Test a sample response.
4. Review the generated transform plan.
5. Choose full refresh or incremental cursor mode.
6. Configure the recurring schedule.
7. Observe the first Dispatcher-submitted sync in the Job Queue and validate its output before depending on the stream.

Current execution path

Web API syncs currently enter the Worker only through the scheduled Dispatcher path. The manual **Sync** and **Full Sync** actions in the UI and Sidekick are unavailable while the Worker contract is hardened to resolve the canonical connection and stream server-side.

Use the first scheduled run as the end-to-end validation of the endpoint, transform plan, and target table. A successful sample-response test validates discovery and mapping, but it does not prove that a complete background sync can load the target.

Common mistakes

Do not use incremental mode unless the source provides a reliable cursor, timestamp, or comparable change marker.

Do not treat a failed scheduled sync as only a dashboard problem. Open the stream and the Job Queue to see whether extraction, transform, or load failed.

Related

- Web API Connections
- Job Queue
- API Patterns

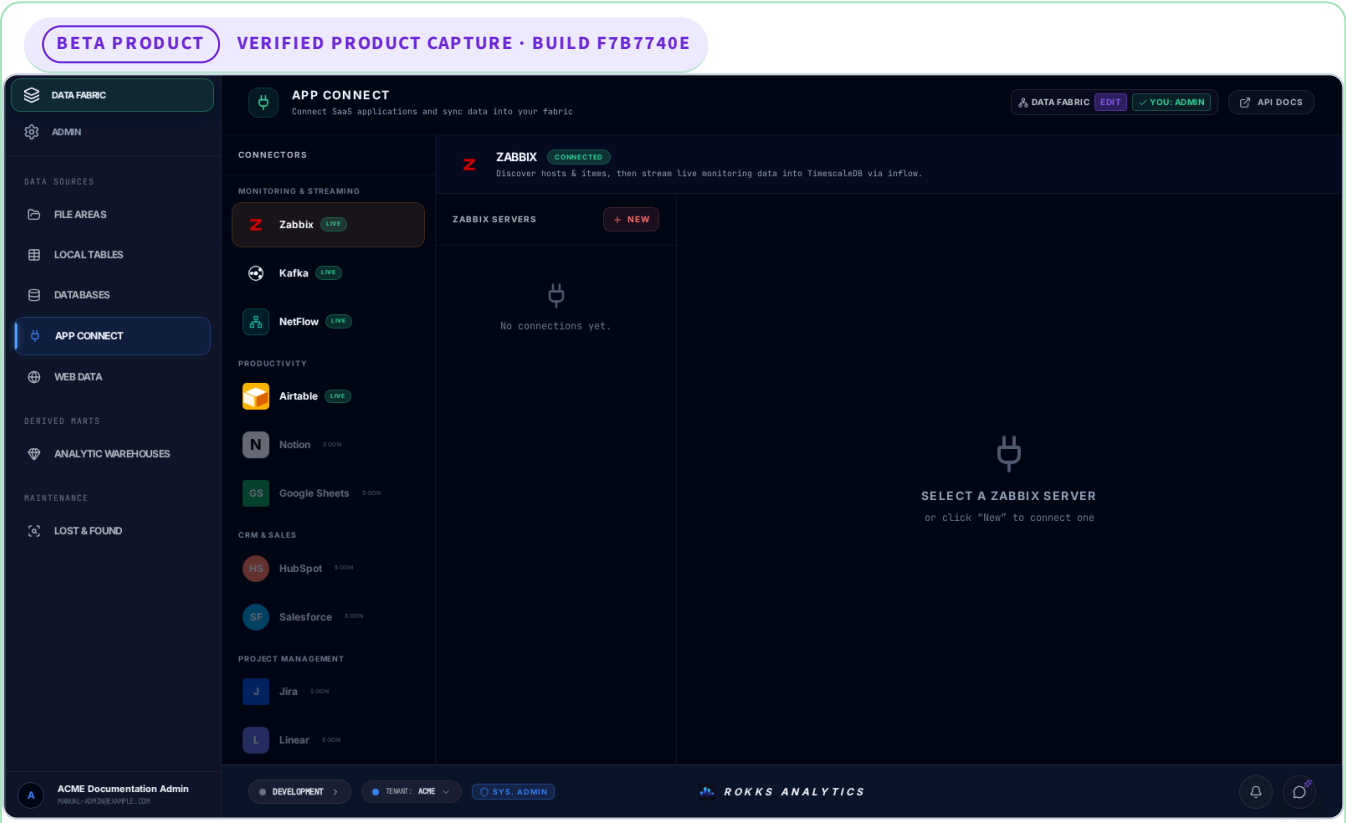
App Connect

Overview

App Connect is the connector catalog for Airtable, Zabbix, Kafka, and NetFlow. Each provider exposes its own setup and execution path; the page does not impose one universal discovery-and-mapping wizard.

How it works

For Airtable, create a connection with a name and Personal Access Token, then choose the tables and fact roles for an Airtable constellation. Airtable fields and types are snapshotted automatically; use **Save constellation** to retain the selection or **Save & Sync** to retain it and start the supported sync. Zabbix has a configuration-sync workflow, while Kafka and NetFlow use the streaming connector surfaces described separately.



The real App Connect catalogue with Zabbix selected. No server connections exist in this ACME fixture; the header badge is not proof of a configured or healthy connection.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open **App Connect** and select one of the connectors marked **Live**.
2. Follow that provider's form and execution path; do not transfer labels or steps from another provider.
3. For Airtable, choose **New**, enter the connection name and Personal Access Token, then choose **Save & Test**.
4. Select the Airtable tables and their fact roles in the constellation panel.
5. Choose **Save constellation** or **Save & Sync**, then monitor the resulting work.

6. For Zabbix, Kafka, or NetFlow, use the provider-specific controls and the Streaming Connectors guidance.

Sidekick can inspect and sync an existing Airtable Constellation, but it does not replace the credential form or table-selection setup above. See the catalog and connection cookbook.

Common mistakes

Do not look for a manual Airtable field-type or key editor on this page. The current Airtable workflow snapshots provider fields and types automatically and asks you to choose tables and fact roles.

Do not treat a connector card's **Live** or **Connected** label as proof that your own connection passed a test. Those labels describe connector availability; validate the configured connection through its provider-specific controls.

Related

- Connector Lifecycle
- Data Source Connector
- App Connect Dashboard Tutorial
- Sidekick task cookbook

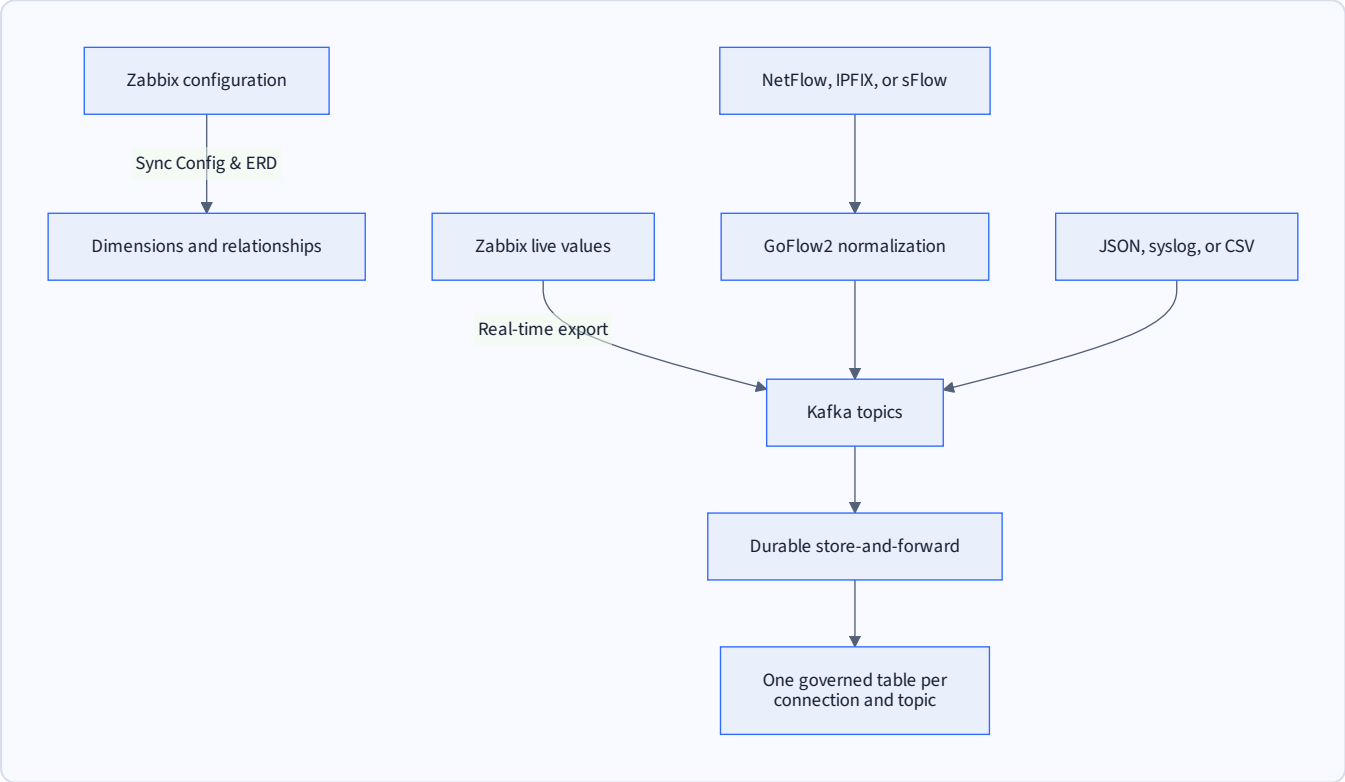
Streaming Connectors

Overview

The streaming connectors in **App Connect** bring continuous operational data into Rokks. Kafka is the streaming transport; Zabbix uses it for live values while synchronizing configuration separately, and NetFlow uses GoFlow2 to normalize flows before Kafka carries them into Rokks.

The illustrations on this page are source-derived **mockups**, not captures of operating Zabbix, Kafka, or NetFlow systems. They use only fictional ACME names and `example.com` endpoints.

How it works



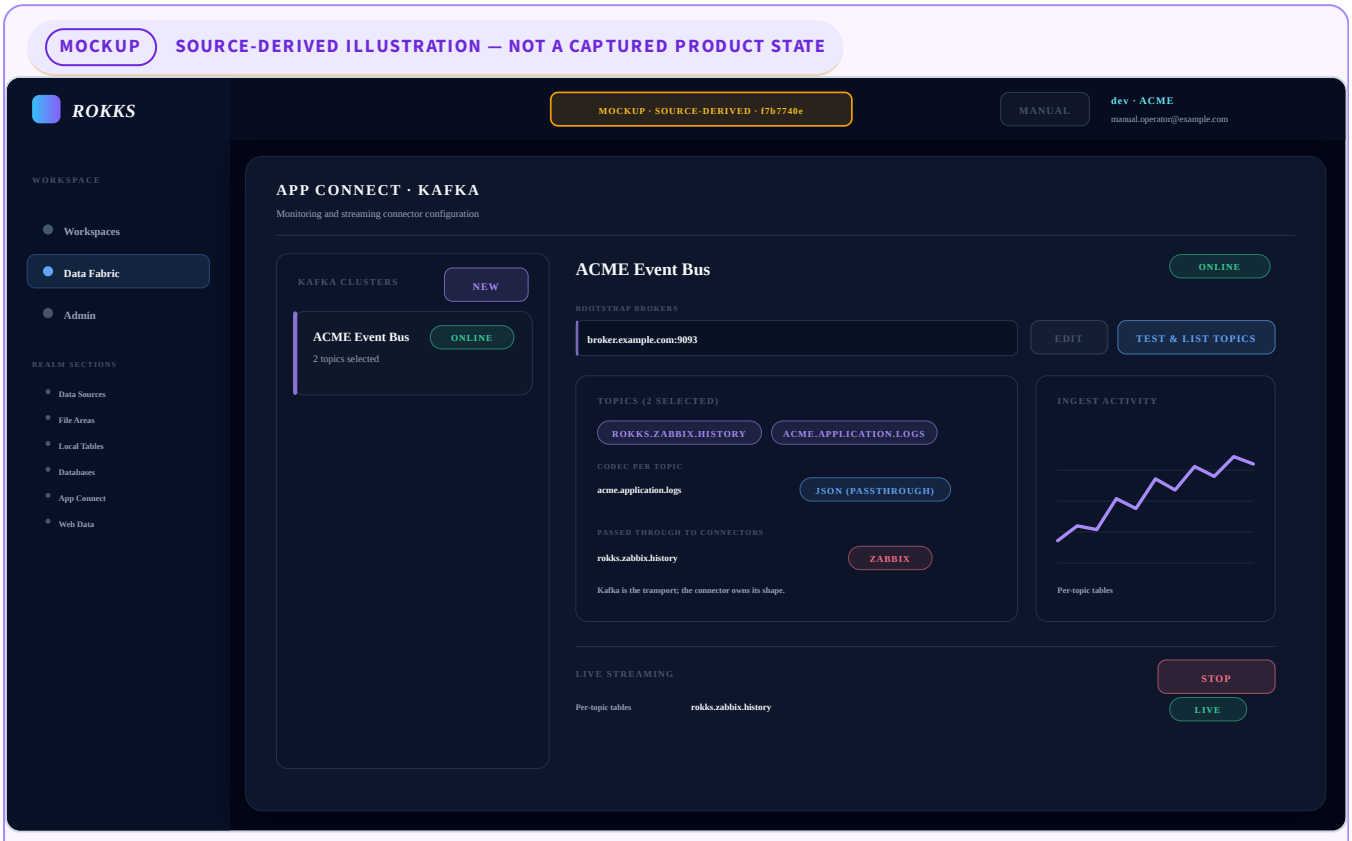
Each selected Kafka topic has a codec. Generic topics use **JSON (passthrough)**, **Syslog**, or **CSV**. A topic linked from Zabbix or NetFlow uses a connector-owned shaped codec; the Kafka page shows its owner and locks the codec there. A running stream shows received and forwarded counts, throughput, spool backlog, the last-record time, errors, and the provisioned per-topic tables.

Step-by-step

Configure the Kafka transport

1. Open **Data Fabric** → **App Connect**, select **Kafka**, and click **New**.
2. Enter a name such as **ACME Event Bus** and one or more bootstrap brokers, such as `broker.example.com:9093`.

3. Enable **Use TLS** and **Use SASL authentication** when the cluster requires them, then click **Save & Discover**.
4. Select the topics to ingest. For a generic topic, choose the codec that matches its actual payload.
5. Click **Start** under **Live Streaming**. The control is available only after the connection is online and at least one topic is selected.
6. Check **Per-topic tables** and the live metrics before using the data in a dashboard.

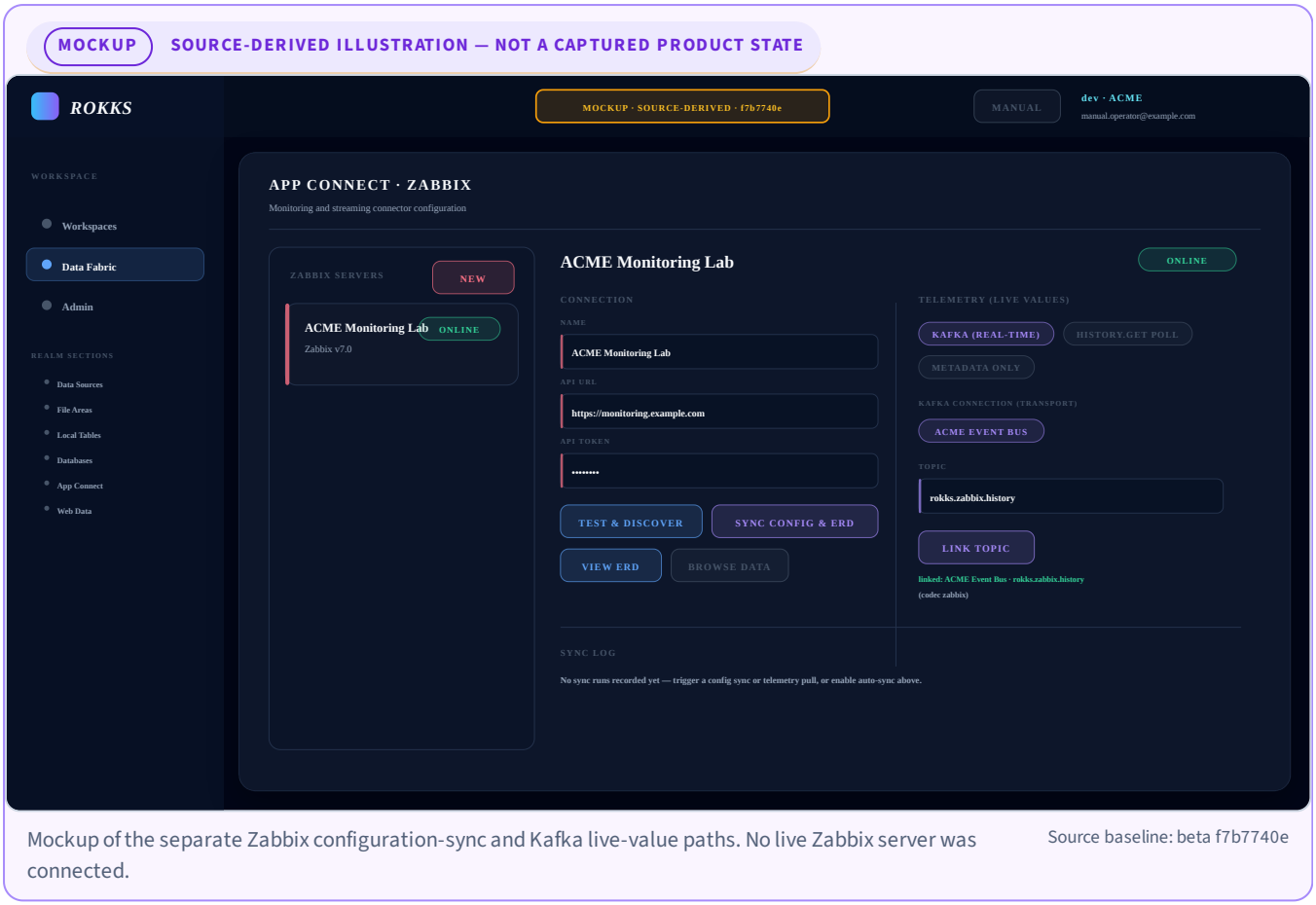


Mockup of an ACME Kafka connection and topic stream. No live cluster or messages were used. Source baseline: beta f7b7740e

Synchronize Zabbix configuration and values

1. Select **Zabbix**, click **New**, and enter a name, the API URL, and an API token. Use an endpoint such as `https://monitoring.example.com` in test documentation.
2. Click **Save & Discover**. Use **Test & Discover** later to repeat the connection check and refresh the host and item preview.
3. When the connection is **Online**, click **Sync Config & ERD**. Rokks tracks the job inline and materializes the configuration dimensions and their relationships. Use **View ERD** or **Browse data** after tables exist.
4. Under **Telemetry (live values)**, keep **Kafka (real-time)** selected for the canonical streaming path.
5. Choose the Kafka connection and topic that receives the Zabbix real-time export, then click **Link topic**. Rokks assigns the connector-owned `zabbix` codec so values use the item-values shape.
6. If the Kafka stream was already running, stop and restart it after linking so the shaped destination table is provisioned.

The UI also offers **history.get poll** for an incremental, on-demand or scheduled **Pull Telemetry** path when Kafka is unavailable. **Metadata only** synchronizes the configuration model and does not ingest values. Configuration sync and telemetry pull have separate schedule controls.



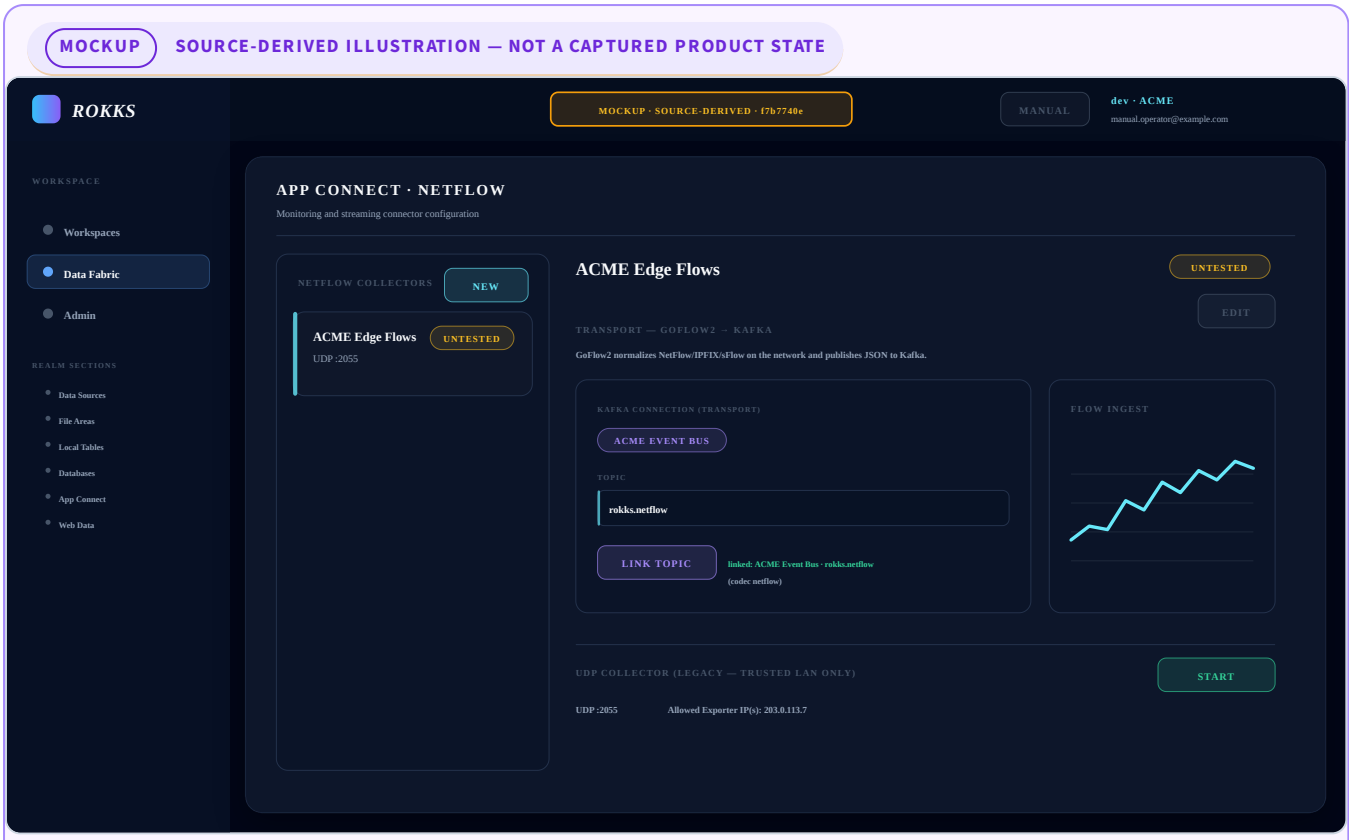
Mockup of the separate Zabbix configuration-sync and Kafka live-value paths. No live Zabbix server was connected.

Source baseline: beta f7b7740e

Link NetFlow through GoFlow2 and Kafka

1. Configure GoFlow2 near the exporters so it accepts NetFlow v5/v9, IPFIX, or sFlow and publishes normalized JSON records to your Kafka cluster.
2. In **App Connect**, select **NetFlow**, click **New**, give the connection a name such as **ACME Edge Flows**, and click **Save**.
3. Under **Transport — GoFlow2 → Kafka**, choose the Kafka connection and normalized flow topic, then click **Link topic**. The suggested topic is `rokks.netflow`.
4. Rokks assigns the connector-owned `netflow` codec. Start, or restart, the linked stream from the **Kafka** connector.
5. Confirm activity on the NetFlow page and verify the topic table and live status on the Kafka page.

The NetFlow page also displays **UDP Collector (legacy — trusted LAN only)**. This direct collector accepts NetFlow v5/v9 and is retained only for appliance or on-premises single-tenant use. If your installation explicitly uses it, restrict **Allowed Exporter IP(s)**, enforce the same restriction with a firewall, and never expose the UDP listener to an untrusted network. GoFlow2 to Kafka remains the canonical path, including for IPFIX and sFlow.



Mockup of the canonical GoFlow2-to-Kafka path. No live exporter, cluster, or flow records were used. Source baseline: beta f7b7740e

Common mistakes

- Do not treat **Sync Config & ERD** as a live-values sync. Select a telemetry source separately.
- Do not choose a codec by trial and error. A codec mismatch surfaces as an error; Rokks does not guess another format.
- Do not try to change a Zabbix- or NetFlow-owned codec on the Kafka page. Change the topic link from the owning connector.
- Do not forget to restart an already-running Kafka stream after linking a shaped topic.
- Do not use the legacy UDP collector for IPFIX or sFlow, or on an untrusted network.

Related

- App Connect
- Local Tables
- Analytic Warehouse
- Job Queue
- Monitoring

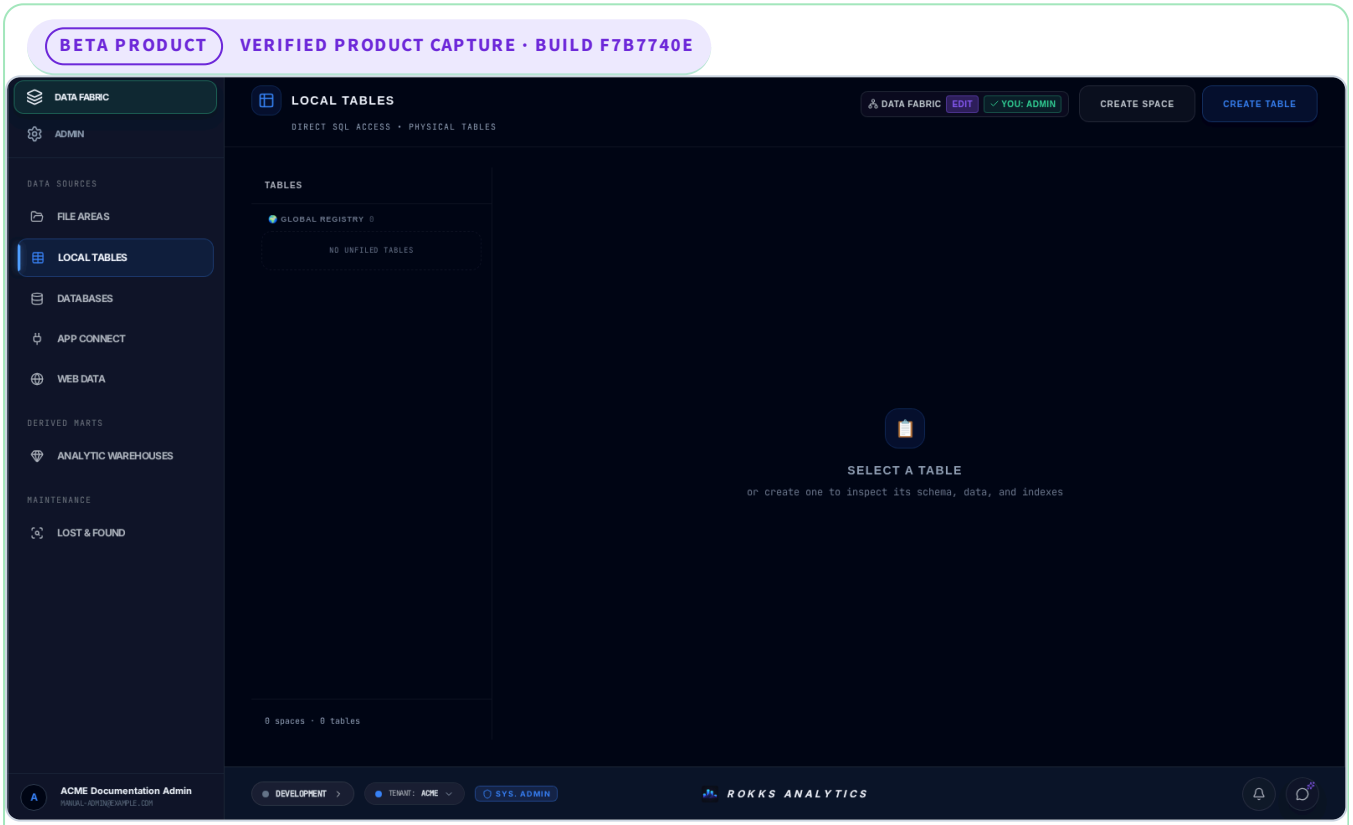
Local Tables

Overview

Local Tables creates governed physical tables directly in the active environment and tenant. Use it when you need a table managed in Rokks without first importing a file or configuring an external database connection.

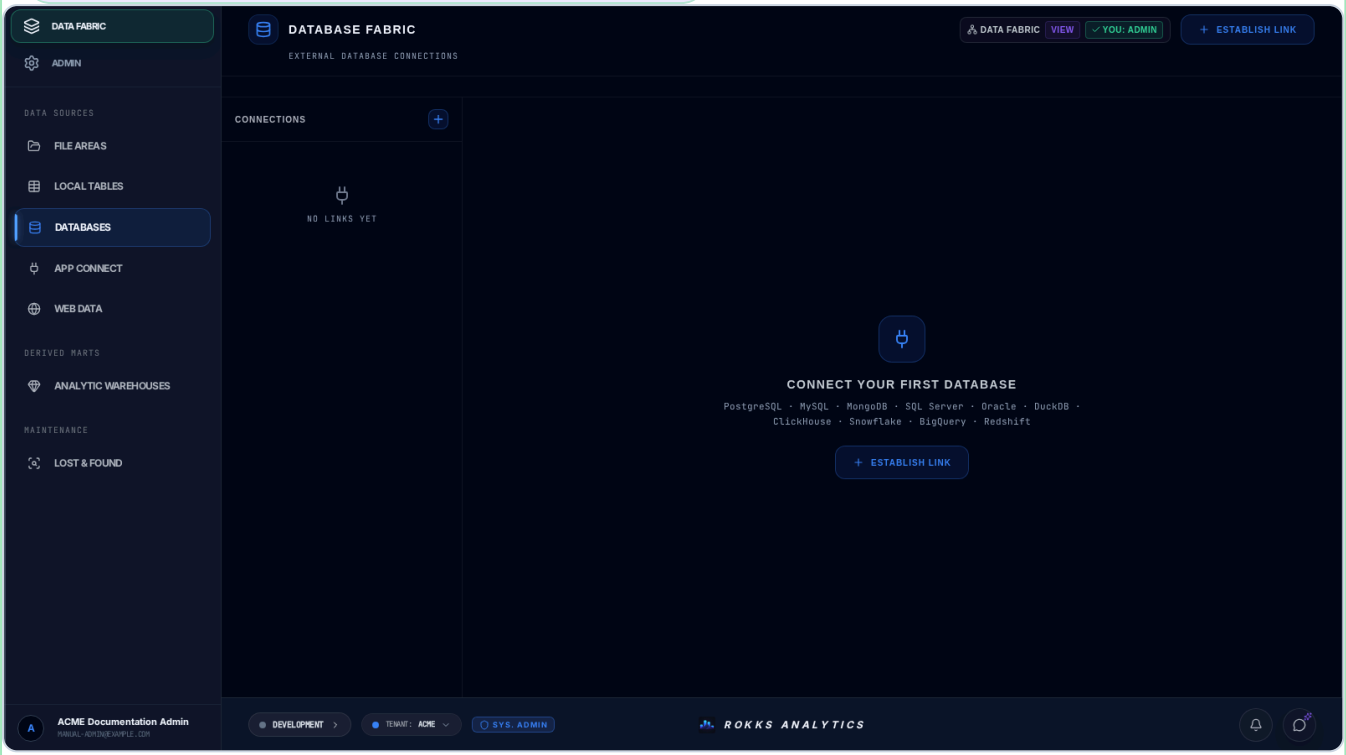
How it works

The left rail organizes tables into named spaces and a **Global Registry** for unfiled tables. **Create Space** opens **Establish Space**. **Create Table** opens **Provision Entity**, where you enter a logical name and select a UUID, INT4, or INT8 ID-column type before choosing **Commit Table**. Selecting a table opens its schema, data, and index inspection surface.



Local Tables in ACME before any local spaces or tables have been created. Create Space and Create Table are available in the toolbar.

Captured 12 Sept 2026 UTC [Open documented view ↗]



Reviewed ACME Database Fabric capture showing the supported connection engines and empty first-link state.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open **Data Fabric** → **Data Sources** → **Local Tables**.
2. Choose **Create Space** if the table should live in a named group, enter the space name, and choose **Commit Space**.
3. Choose **Create Table**.
4. In **Provision Entity**, enter the logical table name.
5. Select the ID column type: **UUID**, **INT4**, or **INT8**.
6. Choose **Commit Table**.
7. Select the new table in the rail to inspect its schema, data, and indexes.
8. Drag the table into a space when you want to organize it, or leave it in **Global Registry**.

Common mistakes

Do not use Local Tables to discover an existing warehouse database. Use the **Data Source Tables** area in Analytic Warehouse for database scanning and registration.

Do not expect file imports or Constellation outputs here. File Areas keeps those governed outputs on the source's **Output tables** list and exposes them through Analytic Warehouse and catalog-backed editors. Local Tables is for tables provisioned directly through **Create Table**.

Choose the ID type deliberately. INT4 is a 32-bit sequence, INT8 is a 64-bit sequence, and UUID is globally unique without a sequence.

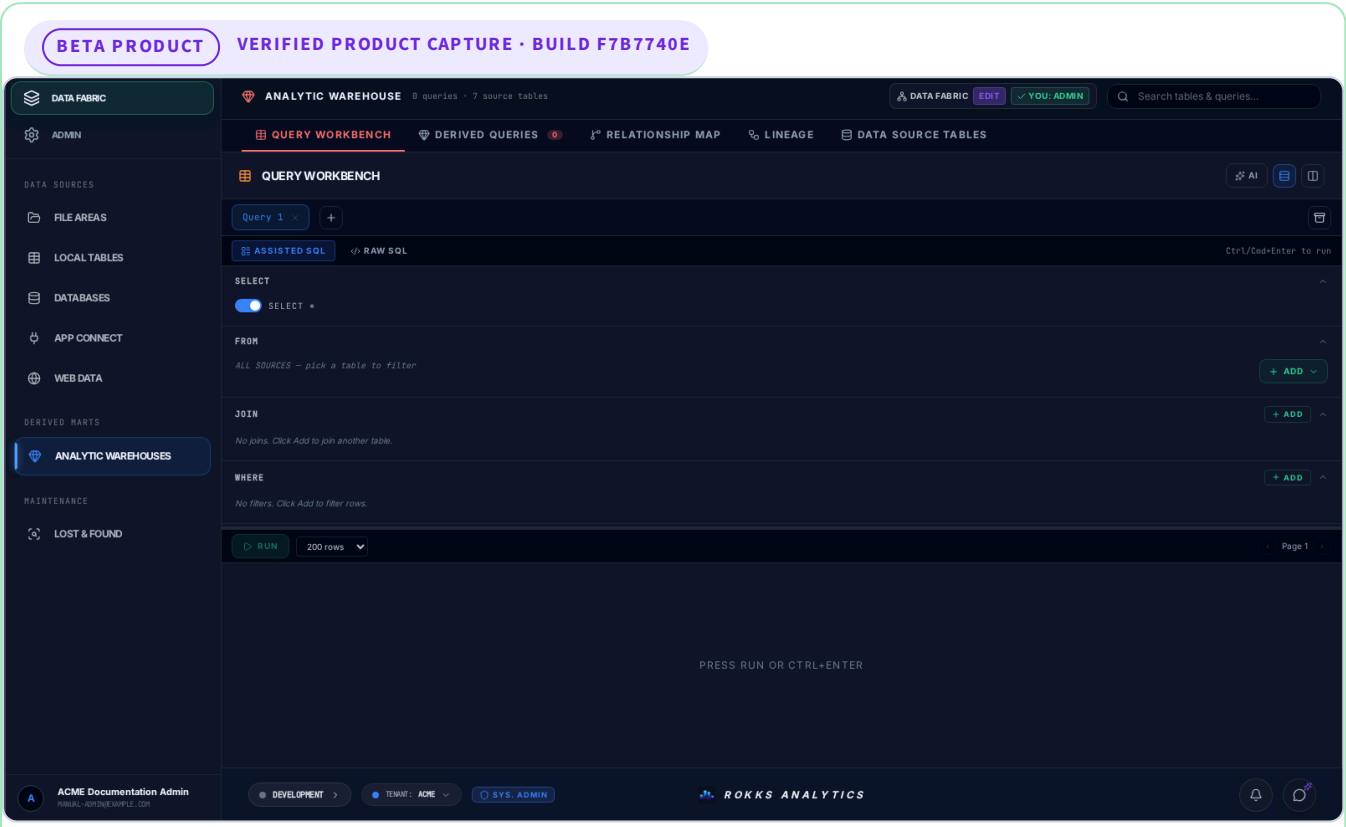
Related

- Schema Editor
- File Areas
- Analytic Warehouse
- Data Source Connector

Analytic Warehouse

Overview

The Analytic Warehouse is where reusable analytical outputs live. Use it for derived tables, curated views, materialized calculations, and shared analysis that should not be rebuilt inside every dashboard. Query Workbench provides a bounded preview before you save or materialize that work.



Reviewed ACME Analytic Warehouse capture showing Assisted SQL, Raw SQL, joins, filters, and the result area.

Captured 12 Sept 2026 UTC [Open documented view ↗]

How it works

A warehouse artifact reads governed source tables, applies a query or transformation, and registers the result for reuse. Some artifacts are refreshed manually or on a schedule. Others are used as source tables for dashboards and alerts.

The **Data Source Tables** area is also where administrators scan the active database, review discovered tables, and register suitable existing tables in the catalog. That discovery workflow is separate from **Local Tables**, which provisions new tables.

Query Workbench previews use a versioned, read-only result stream. The stream carries a schema, bounded row records, and one terminal record that says whether the preview is complete, truncated, or failed. Starting a newer preview cancels the older request; rows from a superseded request do not replace the current result.

A dedicated browser Worker performs frame assembly, UTF-8 decoding, JSON parsing, protocol validation, and row reconstruction. Its row batches remain provisional until the terminal is validated. A complete or truncated terminal commits the preview; an error, cancellation, malformed stream, or terminal-less EOF discards it. The main thread performs the final state integration and renders the grid. The implementation has landed; local and beta certification remain open, so this assignment is not a general responsiveness or enterprise-volume guarantee.

Step-by-step

1. Open **Data Fabric** → **Derived Marts** → **Analytic Warehouses**. **Query Workbench** is the initial tab.
2. Choose an existing query or create a new one.
3. Select governed sources from the catalog.
4. Define the derived output.
5. Run and validate the bounded preview. Check whether the result is complete or truncated before using its row count.
6. Let the workbench auto-save the draft, then choose **Publish** when the result is ready for reuse.
7. Use it from the Widget Builder or downstream workflows.

Common mistakes

Do not duplicate the same complex calculation across many widgets. Create a warehouse artifact when the logic is shared.

Do not materialize an output before validating field names, data types, and row counts.

Do not treat a truncated preview as the complete dataset. Reduce the query scope when you need to inspect a different slice, and use a purpose-built export workflow for bulk data.

Do not assume that provisional rows are valid after a connection closes unexpectedly. Query Workbench accepts them only after a valid complete or truncated terminal.

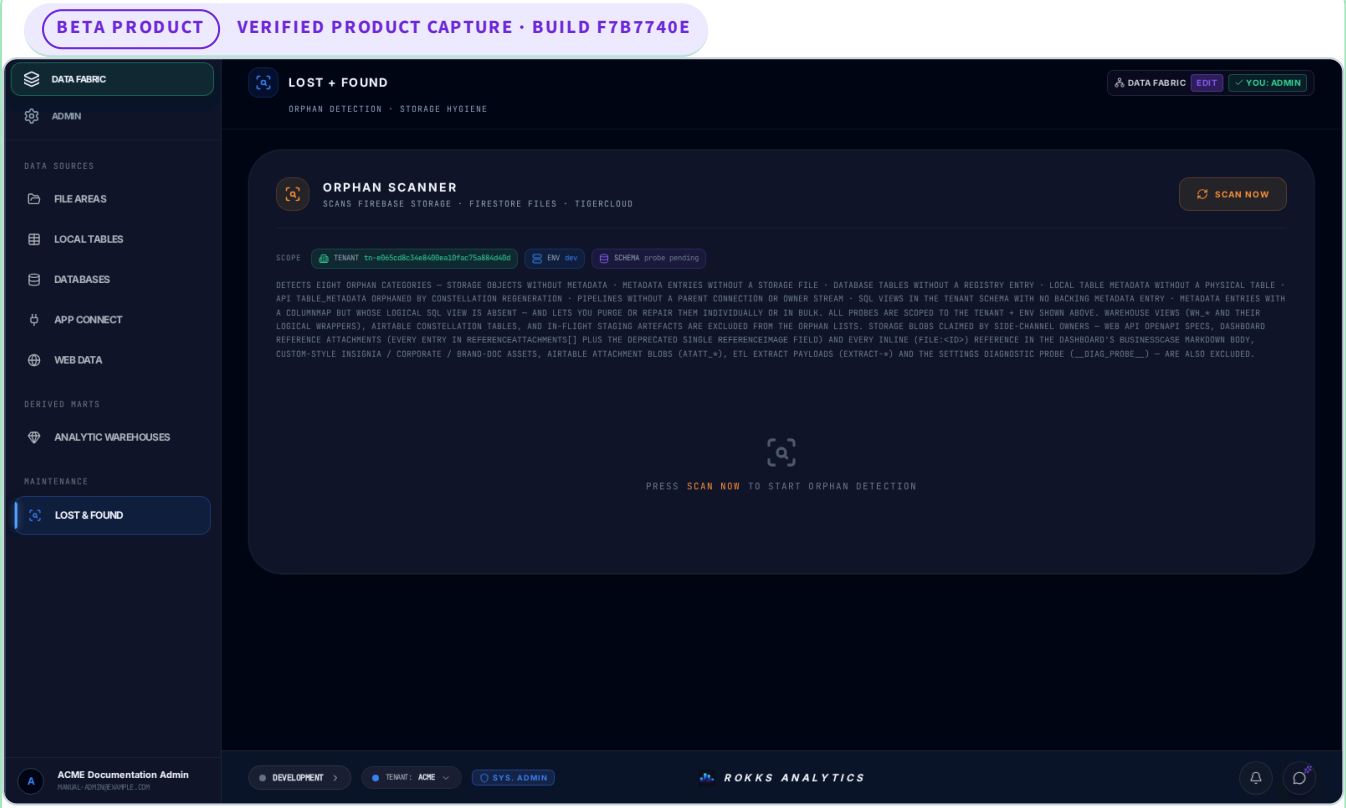
Related

- Widget Builder
- SQL Query Stream v1
- Worker Jobs
- Query Optimizer Troubleshooting

Lost and Found

Overview

Lost and Found helps administrators locate data assets, metadata, or records that no longer have an obvious owner in the UI. Use it when migrations, deletes, imports, or failed jobs leave something that needs review.



Reviewed ACME Lost and Found capture showing the tenant-scoped orphan scanner before a scan.

Captured 12 Sept 2026 UTC [Open documented view ↗]

How it works

The page compares registered entities, catalog metadata, and available backend artifacts across eight orphan categories. After a scan, each affected row exposes only the category-specific action the product supports: **Purge** for disposable residue or **Repair** where a relationship can be rebuilt. These are direct actions from this surface, not Worker jobs.

Step-by-step

1. Open **Data Fabric** → **Maintenance** → **Lost & Found** or **Admin** → **Tools** → **Lost & Found**.
2. Choose **Scan Now**.
3. Review each populated category and its diagnostics.
4. Confirm that no dashboard, connector, tenant, or retained dataset still needs the item.
5. Use **Repair** only where that row offers it, or **Purge** only where that row offers it.
6. Scan again and confirm the expected category is clear.

Common mistakes

Do not delete an item just because it looks old. Check whether a dashboard, connector, or tenant still references it.

Do not use Lost and Found as a normal source browser. It is for recovery and repair.

Related

- Backup and Restore
- Service Stack
- Troubleshooting

SECTION 3

Dashboards

13 topics

Starred and Attention

Overview

Starred and **Attention** are separate cross-workspace overview pages. Starred brings selected live widgets together. Attention lists saved compile failures, stale widgets, and alert violations already reported by rendered widgets.

How it works

Starred groups pinned widgets by workspace and dashboard. Supported data widgets render as live charts or tables, and each tile includes a location link back to its source dashboard. Narrative insight widgets are not included in this live feed.

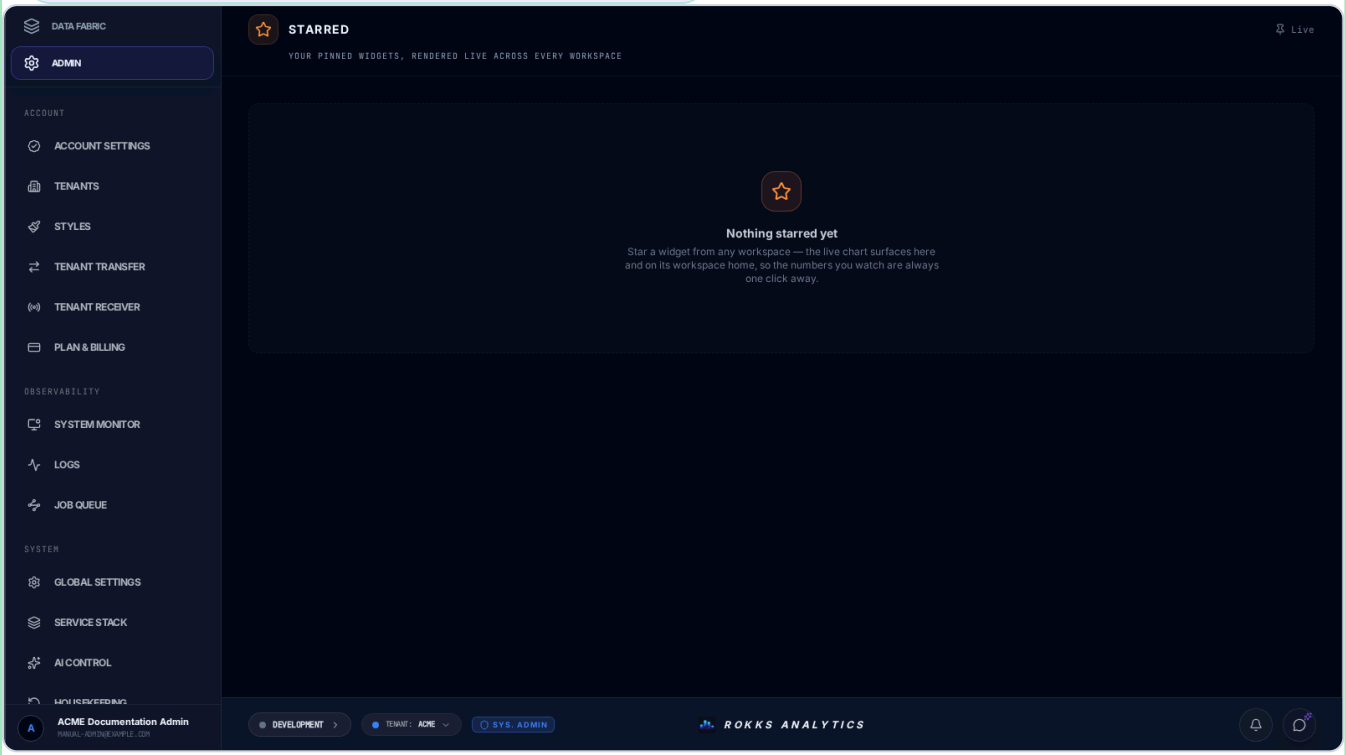
Attention is a triage view, not a second dashboard. It groups compact cards by workspace and dashboard and classifies each affected widget as:

- **Threshold breached** — a rendered widget has reported an alert violation. The summary counts these as **Observed alert breaches**; Attention does not independently confirm the current value.
- **Stale (last compile failed)** — the latest compile failed, but an earlier compiled result can still render.
- **Compile failed** — the compile failed and no earlier compiled result is available.

Opening Attention does not run threshold checks, issue new data queries, or start a periodic scan. The feed reflects saved widget configuration and existing live-widget observations. **Manage rules** opens the Sentinel alert-rule manager.

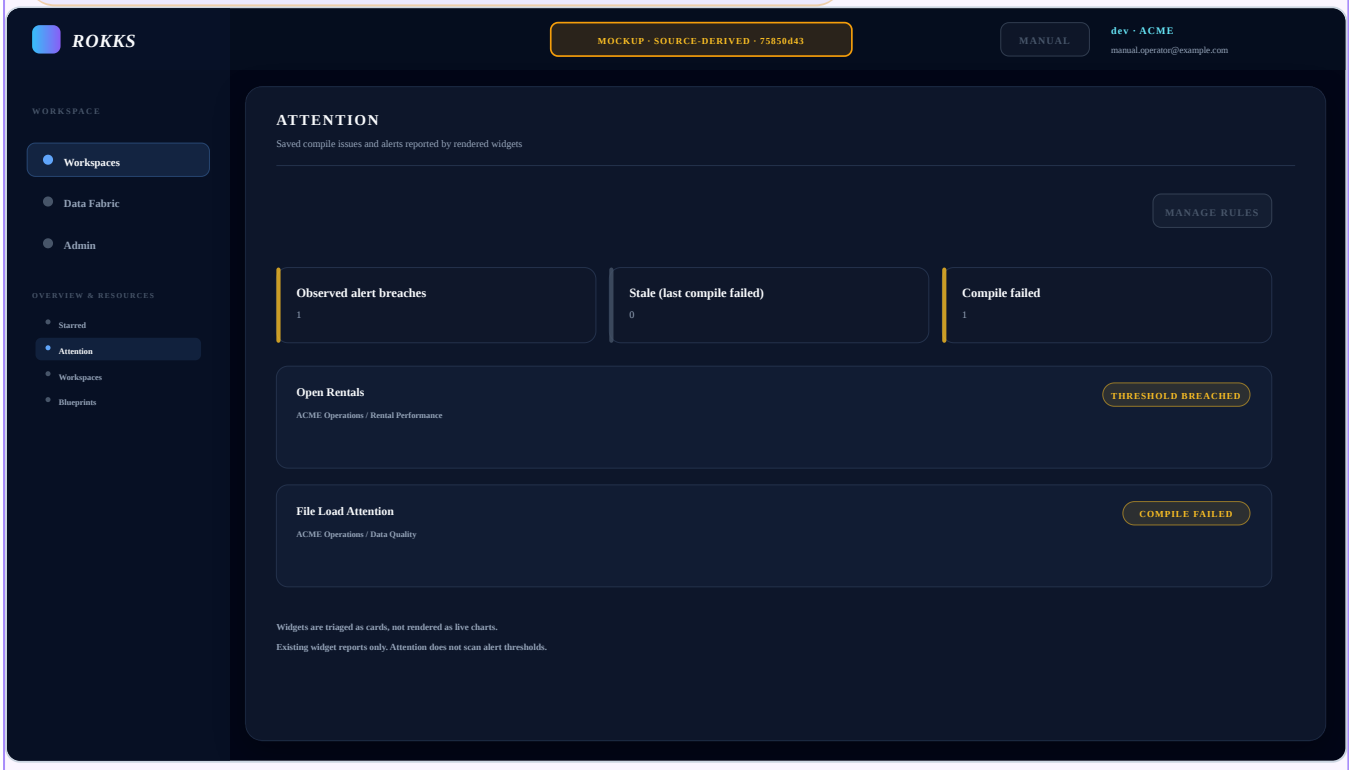
No reported issues means that this feed has no saved compile failures, stale widgets, or reported live-widget alerts. It does not prove that every widget is healthy, and it is not a whole-platform health scan.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Reviewed ACME Starred capture showing the empty pinned-widget state. Captured 12 Sept 2026 UTC [Open documented view ↗]

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Source-derived ACME Attention mockup showing saved compile issues and an already reported widget violation. This illustration reflects the newer source baseline shown below; the Starred capture retains its original beta baseline.

Source baseline: ROKKS web source 75850d43

Step-by-step

1. Open the **Workspaces** area.
2. Select **Attention**. When issues are reported, review the three status totals; otherwise read the **No reported issues** explanation.
3. Read a card's status, reason, workspace, and dashboard before deciding what to fix.
4. Select a card to open its source dashboard and focus the affected widget.
5. Click **Manage rules** when an alert-rule definition needs review. Reopening Attention does not trigger an alert evaluation.
6. Open **Starred** to view the current live render of widgets already present in the pinned set.
7. Use a Starred tile's location header to open the source dashboard.

Common mistakes

Do not read **Stale** as "no data." A stale widget still has a previous compiled result, but that result may not reflect the latest configuration.

Do not expect Attention to repair a widget in place. Its cards diagnose and navigate; make the correction in the source dashboard or alert-rule manager.

Do not treat an observed breach as a fresh measurement, or an empty Attention feed as evidence that every threshold has been checked. Attention displays existing reports rather than evaluating rules itself.

Do not expect the Starred page itself to manage the pinned set. This build exposes viewing and jump-to actions on that page, but no pin or unpin control there.

Related

- Workspaces
- Widget Builder
- Dashboard Filters
- Alerts

Workspaces

Overview

A workspace is the home for a related set of dashboards and reports. Use workspaces to keep a business or operational subject together, then use the workspace home to reach its pinned insights, dashboards, and reports.

How it works

The **Manage Workspaces** page shows every workspace and an **Ungrouped** bucket. Browse mode opens workspace, dashboard, and report cards. **Edit** mode exposes workspace and dashboard names, icons, ordering, movement, permissions, and workspace deletion.

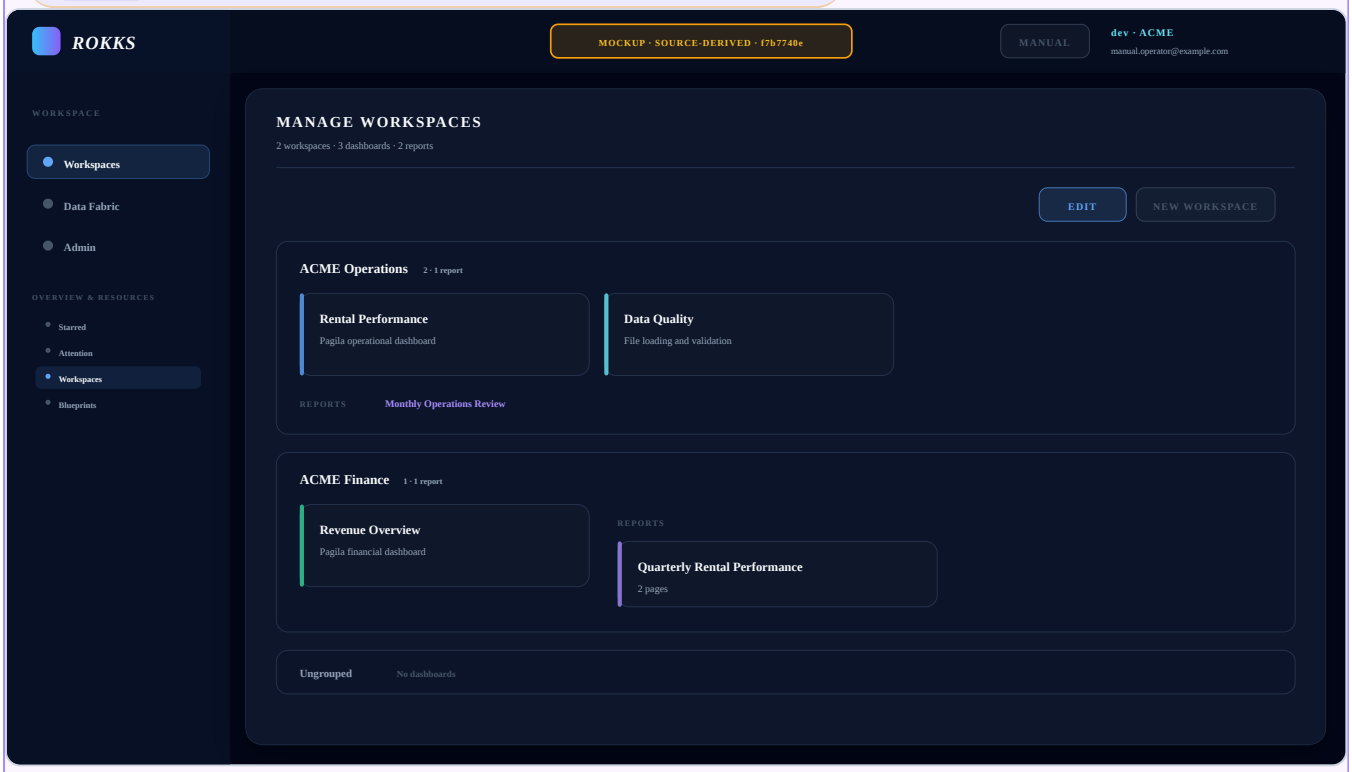
A single workspace home has three tiers:

1. **Pinned** — live widgets associated with that workspace.
2. **Dashboards** — dashboard preview cards.
3. **Reports** — report cards and **New report**. Reports belong to a real workspace, so this tier does not appear in Ungrouped.

The **Compose** action opens the workspace brief and source-material composer. **Permissions** opens the role-assignment editor. It separates grants made directly on this workspace from read-only inherited grants, and lets an authorized administrator choose a subject and an eligible role. Subjects can be users, groups, companies, or teams. **Held rank — no explicit cap** uses the subject's held role; an explicit role caps the grant rather than promoting that subject.

In this documented build, object role assignments govern metadata reads and writes for workspaces and their resources. Inherited access must be changed at its source; it cannot be subtracted on a child object. Do not treat object permissions as proof of complete SQL or data-source isolation: the SQL query paths still disclose incomplete data-source authorization. Confirm the data protection separately before using confidential information.

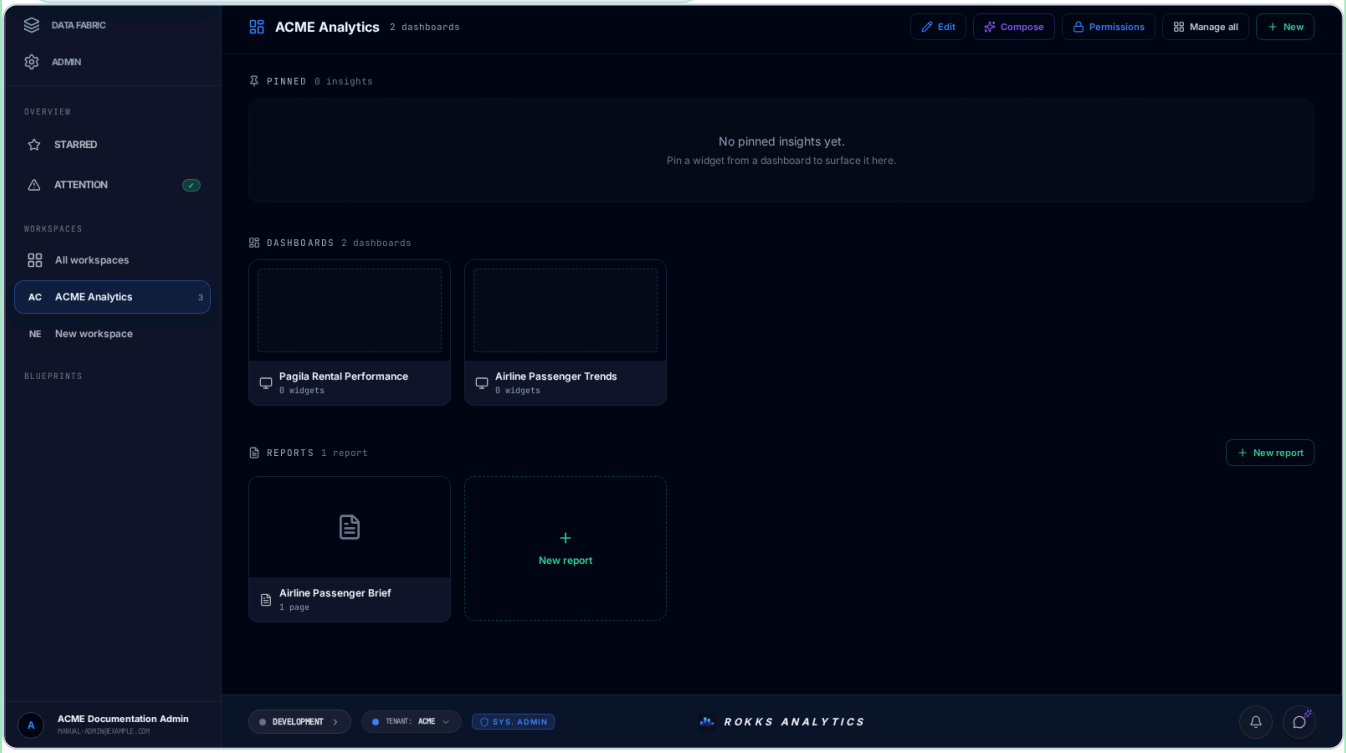
MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Source-derived ACME workspace mockup. It is an illustration, not a capture of a configured account.

Source baseline: beta f7b7740e

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



ACME Analytics workspace with two starter dashboards and a one-page report. Dashboard widgets and pinned insights have not been added yet.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open the **Workspaces** area and select **All workspaces**.
2. Click **New workspace**. The new section appears with the name **New workspace**.
3. Click **Edit** to expose management controls.
4. Rename the workspace inline and choose an icon if one helps users identify it.
5. Use the dashed **Add dashboard** card to create a dashboard inside that workspace. The new dashboard opens immediately.
6. Return to the workspace and drag dashboard cards to reorder them. On **Manage Workspaces**, drag a dashboard into another workspace to move it.
7. Click **Done** to return to browse mode.
8. Open the workspace header to see its Pinned, Dashboards, and Reports tiers.
9. Use **New** for another dashboard or **New report** for a report. Use **Manage all** to return to the tenant-wide view.

Sidekick can create and rename a workspace or dashboard in **Agent** mode. Use the dashboard authoring cookbook and stop after one reviewable result before adding the next dashboard component.

Common mistakes

Do not look for report cards while **Manage Workspaces** is in Edit mode. The current edit layout shows workspace and dashboard management, but its report rows are hidden until you return to browse mode.

Do not delete a workspace to delete its dashboards. Deleting the workspace removes the grouping; its dashboards remain and move to **Ungrouped**.

Do not confuse company or team membership with a workspace grant. Membership identifies the people a grant can reach; the assignment binds that subject to an object. Do not use **Permissions** alone as proof that a user cannot query the underlying data.

Related

- Starred and Attention
- Reports
- Companies and Teams
- Blueprints
- Dashboard Collections
- Sidekick task cookbook

Widget Collections

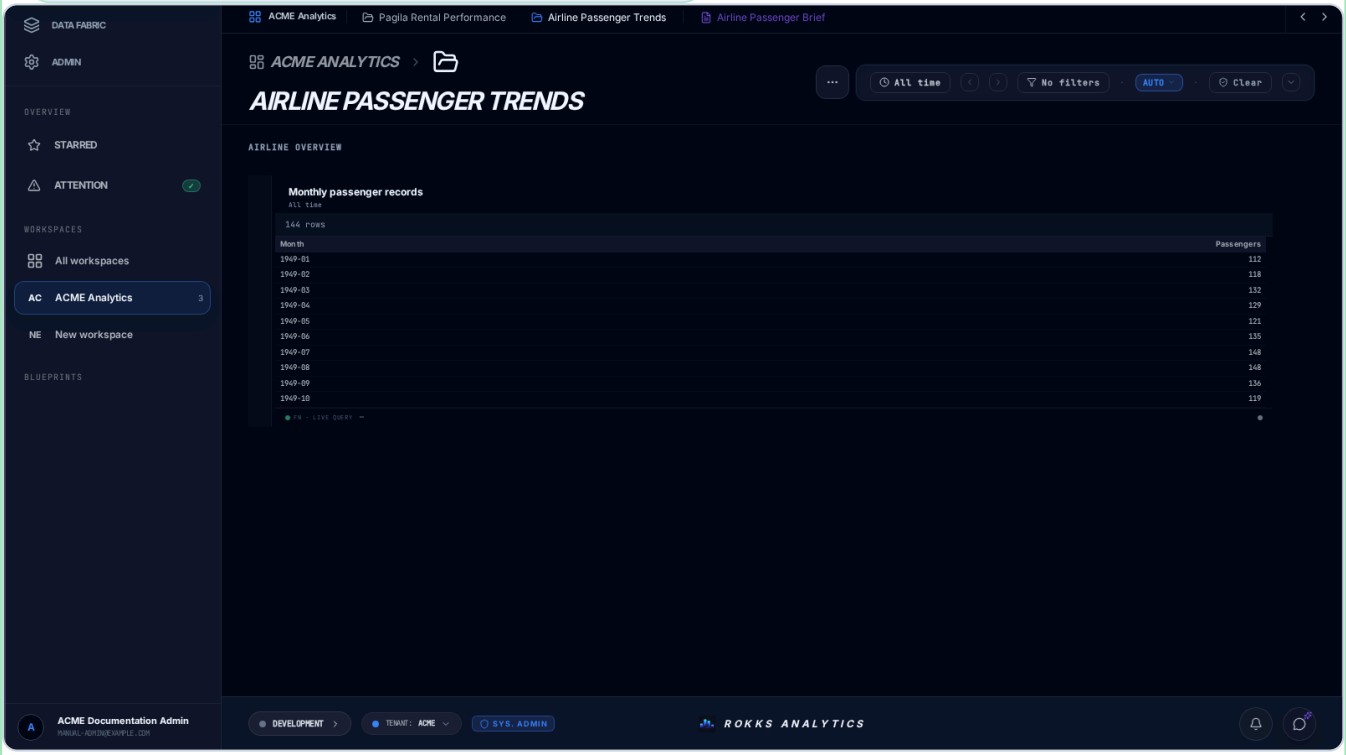
Overview

In this build, a collection is a widget feed, not a container for dashboards. It can hold copied widget configurations directly or select widgets dynamically by tags. A collection may be associated with a workspace and marked as pinned.

How it works

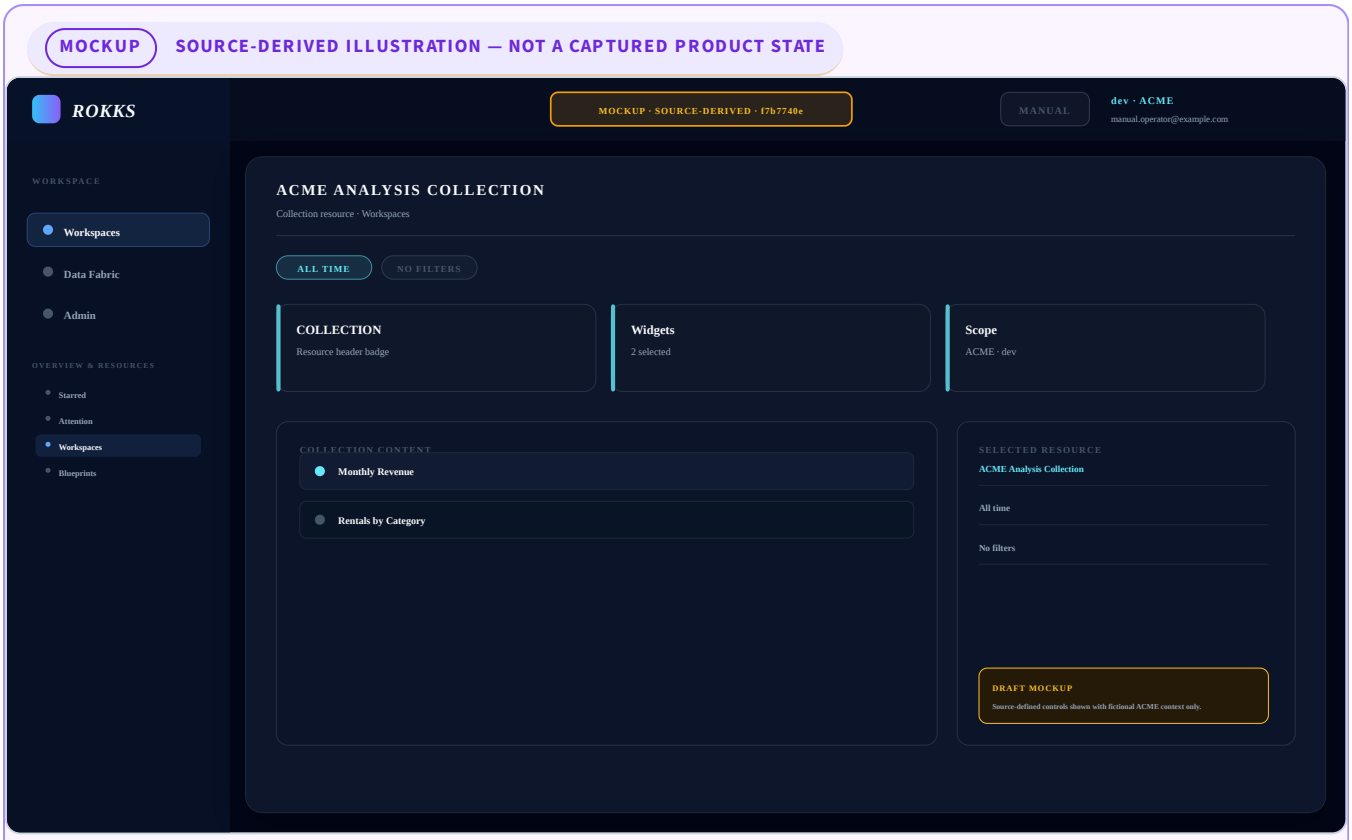
The standalone Collection view remains route-reachable, but Collections no longer have a navigation entry: the current navigation folds this concept into **Starred**. Dashboards and reports are organized by **Workspaces**, while a workspace's pre-existing pinned widget copies appear in its **Pinned** tier. This documented build can display stored pinned state, but exposes no user-facing control to add a widget to, or remove one from, that set.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Real beta dashboard with the compiled Monthly passenger records table using the public Airline Passengers sample.

Captured 12 Sept 2026 UTC [Open documented view ↗]



Collection resource illustration based on the shared analytics canvas and resource routing. Source baseline: beta f7b7740e

Step-by-step

1. Open **Workspaces** when you need to organize dashboards or reports.
2. Open **Starred** or a workspace's **Pinned** tier to view any widgets already present in stored pinned collections.
3. Use a tile's location header to return to its source dashboard.
4. Treat an old direct Collection link as a route-reachable widget collection, not as the place to create workspace structure.

Common mistakes

Do not try to add dashboards to a collection. Workspace membership owns dashboard and report organization.

Do not look for a widget star or pin action in this build. The underlying state actions are not connected to a user-facing control.

Do not describe a collection as an access-control boundary. It is a widget-feed model; environment, tenant, workspace, and underlying data authorization remain separate concerns.

Related

- Widget Builder
- Workspaces
- Version History
- Styles

Blueprints

Overview

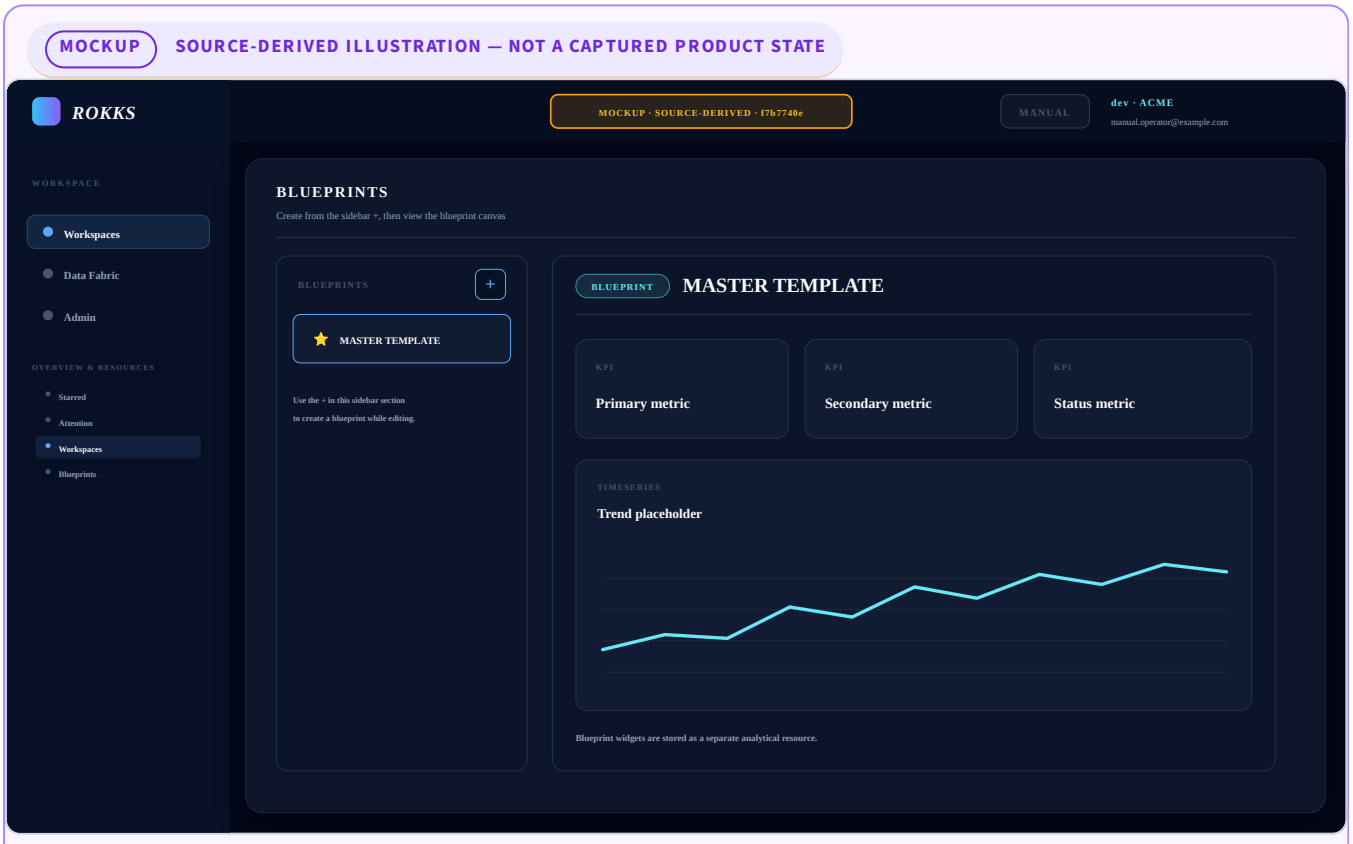
Blueprints are widget-bearing resources listed separately from workspaces and dashboards. Use the Blueprints section to keep named analytical structures available as their own resources while the product's blueprint workflow continues to evolve.

How it works

The pinned, expanded sidebar has its own **Blueprints** section. Sidebar edit mode exposes a plus control to create a blueprint and controls to rename, reorder, and delete existing entries. Opening an entry selects the Blueprint view and renders its stored widgets on the shared analytics canvas.

Blueprint management stays in that sidebar section. The name and icon identify an entry, and its position in the section is user-controlled. Creating an entry starts with the product-supplied **MASTER TEMPLATE** name, then immediately presents the inline rename field; the initial label is not a separate template choice.

This build does **not** expose a command that applies, instantiates, or converts a blueprint into a dashboard. It also does not expose the dashboard's **Edit Layout** action while the Blueprint view is active. A blueprint is therefore a separately stored resource in this build, not a complete end-user templating workflow.



Source-derived blueprint mockup for the draft manual. It illustrates the feature area and is not a captured product state.

Source baseline: beta f7b7740e

Step-by-step

1. Open the **Workspaces** area.
2. Pin and expand the sidebar; sidebar editing is unavailable while the sidebar is only a hover flyout.
3. Click the pencil icon to enter sidebar edit mode.
4. In **Blueprints**, click the plus control. ROKKS creates an entry named **MASTER TEMPLATE** and immediately opens its name for editing.
5. Enter a descriptive name and confirm it.
6. Click the blueprint entry to open its current canvas.
7. Return to sidebar edit mode when you need to rename, reorder, or delete the entry.

Common mistakes

Do not promise users that selecting a blueprint will create a dashboard. There is no apply or instantiate control in this documented build.

Do not confuse sidebar edit mode with dashboard layout editing. Sidebar edit mode manages the blueprint entry; the Blueprint view does not currently expose **Edit Layout**.

Deletion is final in the sidebar confirmation shown by this build. Verify the target name before you confirm **Delete Resource**.

Related

- Workspaces
- Widget Builder
- Dashboard Collections
- Version History

Reports

Overview

A report is a paginated, print-oriented document inside a workspace. Use reports when the output needs fixed page geometry, a narrative, repeatable page furniture, and a browser-generated PDF rather than an interactive dashboard canvas.

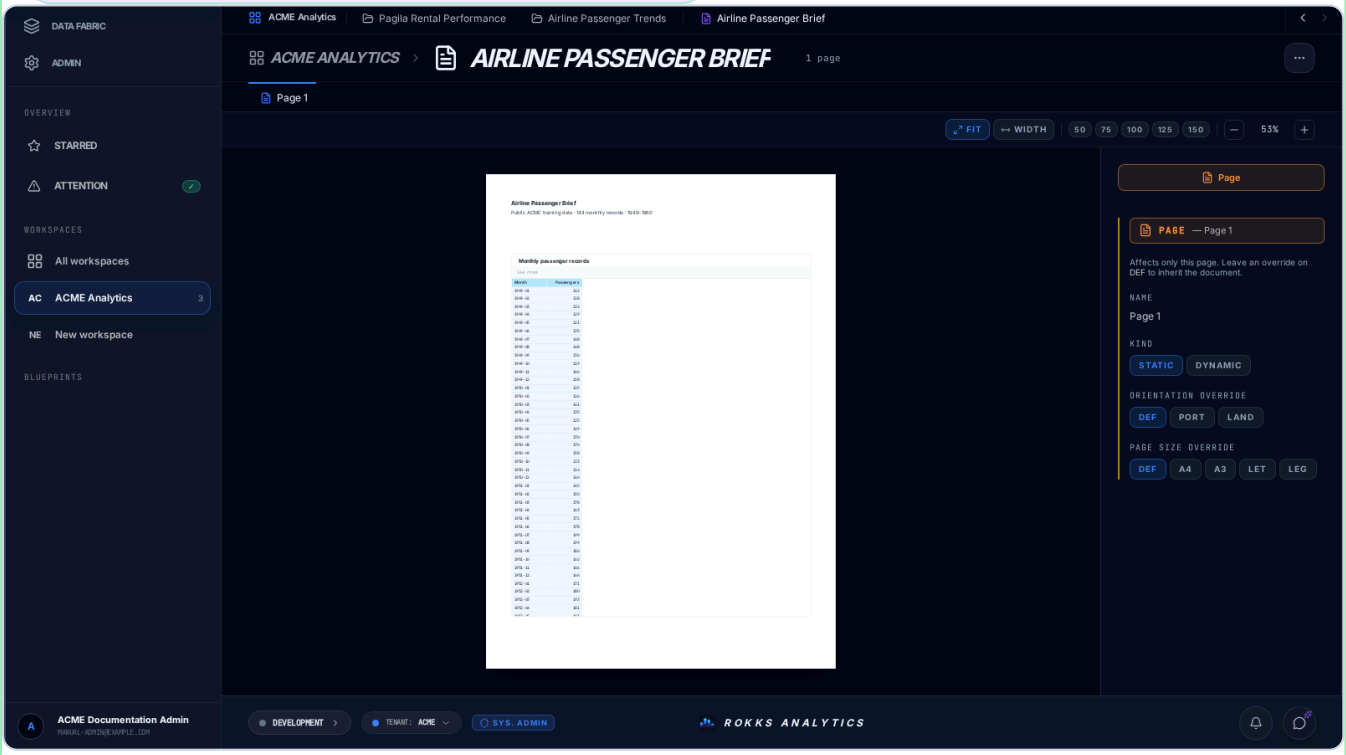
How it works

Each report contains an ordered set of pages. A page can hold widget, text, image, and shape blocks positioned on a fixed sheet. Document settings define page size, orientation, margins, and optional repeating header and footer bands; an individual page can override its size and orientation.

The header's ••• action menu contains **Print**, **Add page**, **Compose**, and **Edit**. Edit mode adds the block palette, page reordering, snapping, selection handles, and the Element/Page inspector. Changes auto-save when you finish editing.

The Document Composer has Overview, Brief, Data, and Settings areas. Linking a backing dashboard makes its widgets available through **Link table**. If that dashboard defines filter fields, the report header also offers a dataset selector whose selection applies across linked widgets on every page.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Real beta portrait report: a heading and the dashboard-linked Monthly passenger records table render from the public Airline Passengers sample.

Captured 12 Sept 2026 UTC [Open documented view ↗]

The real capture shows a one-page portrait report containing a text heading and a live-linked table. The table still belongs to the backing **Airline Passenger Trends** dashboard: report placement does not create a second data query or a detached copy.

Step-by-step

1. Open a real workspace and find the **Reports** tier.
2. Click **New report**. ROKKS creates the report in that workspace and opens it.
3. Rename the report by selecting its title in the header.
4. Open ⋯, choose **Compose**, and write a brief that describes the document's purpose and structure.
5. In Composer **Settings**, choose the page size, orientation, and margins. Add a header or footer only when the document needs a repeating band.
6. In Composer **Data**, select a dashboard when the report should contain live-linked widgets.
7. Close the Composer, open ⋯, and click **Add page**.
8. Choose **Edit**. Arm **Text**, **Image**, **Shape**, or a widget block and click or drag on the sheet to place it. Use **Link table** to place a widget from the backing dashboard.
9. Use the page tabs to switch pages. In edit mode, drag a page tab to reorder it.
10. Finish editing, open ⋯, and choose **Print**. ROKKS prepares the complete report and opens the browser print dialog, where you can save it as a PDF.

Common mistakes

Do not look for a separate Save button. Report changes auto-save, and the active Edit control finishes layout editing.

Do not expect **Link table** before you link a backing dashboard that has widgets. Configure the source in Composer **Data** first.

Do not expect a dataset selector merely because a dashboard is linked. The linked dashboard must also define filterable fields.

Be careful with table presentation settings in the Element inspector. The rendered-row limit and column width or wrapping settings are saved on the backing dashboard widget, so they can affect its other placements.

Use the browser print dialog's cancel control if you do not want to complete the PDF hand-off. **Print** is disabled until the report has at least one page.

Related

- Workspaces
- Dashboard Filters
- Tables
- Widget Builder

Widget Builder

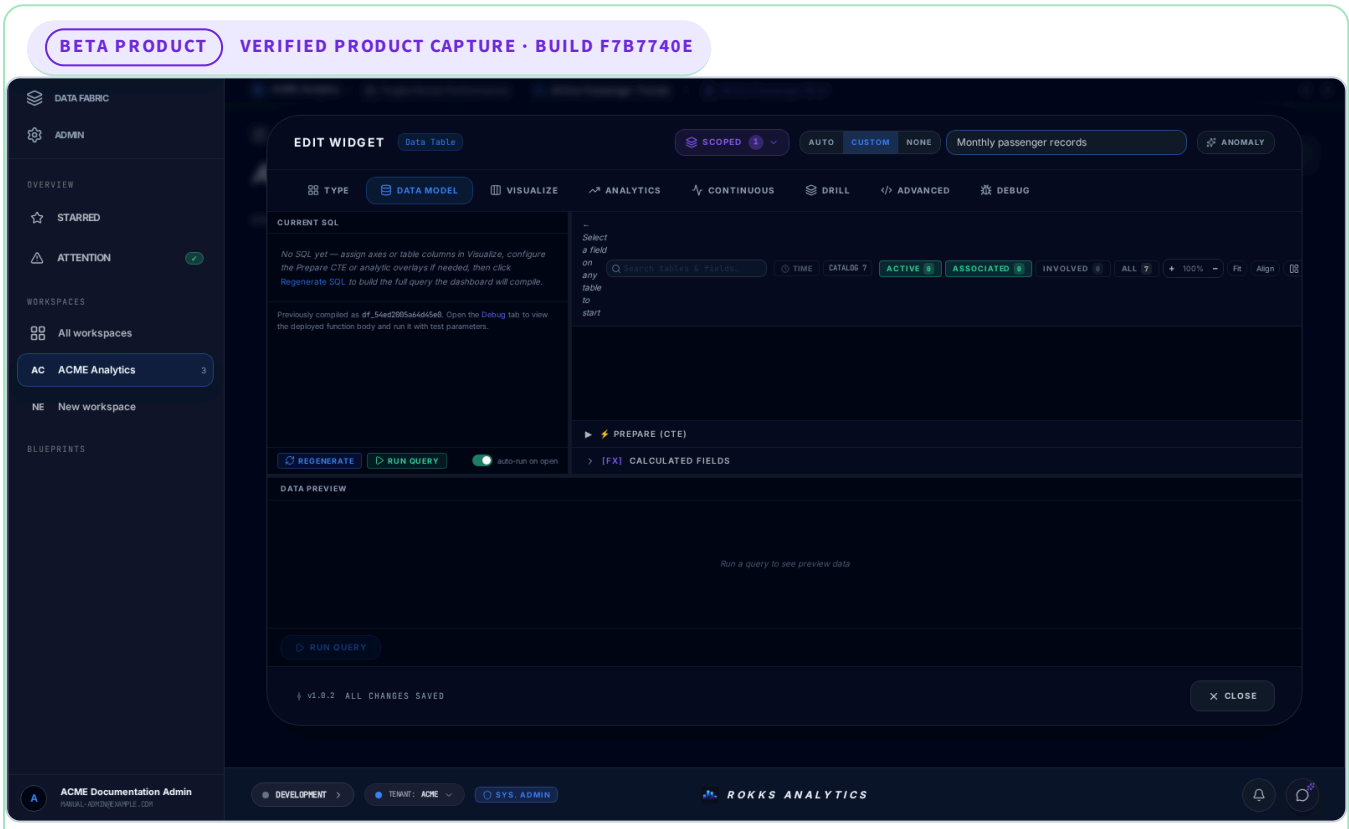
Overview

The Widget Builder turns governed catalog data into a dashboard widget. It lets you choose a source, select fields, configure the visualization, preview the result, and close the editor after the instant save completes.

How it works

The builder reads the catalog instead of asking you to type technical identifiers. You select business-friendly field names, metrics, dimensions, filters, timelines, and display options. Rokks compiles the selection into a query and returns preview rows. In the dashboard flow, edits apply immediately; the footer shows **Close**, not a separate Save action.

The reviewed beta capture below shows the real editor frame, its catalog-backed schema canvas, live-preview area, and the **All changes saved** footer. The particular fields depend on the data source and widget you open.



Reviewed ACME beta capture of the Widget Builder data-model screen. Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open a dashboard.
2. Click **Add Widget** or edit an existing widget.
3. Choose a data source.
4. Select the visualization type.
5. Add metrics, dimensions, time fields, filters, sorting, or table columns.

6. Review the live preview.
7. Wait for the instant save to finish, then click **Close**.

For a chat-driven alternative, ask Sidekick to inspect the governed source in **Ask** mode, then switch to **Agent** for one named widget. The dashboard authoring cookbook keeps query compilation and review as an explicit checkpoint.

Common mistakes

Do not build a widget before the source metadata is clean. Poor names and wrong types make the builder harder to use.

Do not use a chart type just because it is visually interesting. Choose the type that answers the question.

Related

- Data Source Connector
- Charts
- Tables
- Sidekick task cookbook

Data Source Connector

Overview

The Data Source Connector is the part of the Widget Builder that connects a widget to governed data. It is where you choose the source table or connected sources that the widget should query.

How it works

The connector reads the current catalog and offers available tables and fields. When a widget needs more than one table, it uses configured relationships or selected join fields. The connector sends resolved schema columns to the rest of the builder so chart, table, and filter panels stay consistent.

Step-by-step

1. Open the Widget Builder.
2. Choose a primary source.
3. Review the available fields.
4. Add related sources only when the widget needs them.
5. Confirm the join or relationship path.
6. Continue to chart, table, metric, or filter settings.

Common mistakes

Do not join sources just to make the field list larger. Add related sources only when the visual question needs them.

Do not select fields by old names from memory. Use the displayed catalog labels and update metadata if the labels are wrong.

Related

- Widget Builder
- Schema Editor
- Widget Capabilities

Charts

Overview

Charts communicate patterns in governed data. Rokks supports time series, category charts, scorecards, maps, networks, gauges, and other visual forms.

How it works

Most chart widgets combine a data source, one or more measures, one or more dimensions, and optional filters or time settings. The Widget Builder validates the selected fields, previews the result, and instant-saves dashboard edits.

Step-by-step

1. Start from a clear question.
2. Choose a source that contains the needed fields.
3. Select the chart type that matches the question.
4. Add metrics and dimensions.
5. Configure timeline, sorting, labels, colors, and overlays.
6. Preview the chart and check for misleading scales.
7. Wait for the instant-save state, then choose **Close** when the result answers the question.

Common mistakes

Do not use pie or donut charts for many categories. Use bars or tables when categories exceed a small set.

Do not assume that positive variance is always good. Favorability depends on the measure’s business direction.

Related

- Widget Builder
- Filters
- Widget Capabilities

Tables

Overview

Table widgets show row-level or grouped data when users need exact values, operational lists, or ranked records. Use tables when the question cannot be answered by a single visual mark.

How it works

A table widget selects columns from a governed source and can apply sorting, grouping, formatting, conditional styling, and filters. Tables are built for scrolling through records rather than choosing a fixed page size.

Step-by-step

1. Choose a source in the Widget Builder.
2. Select the columns users need.
3. Put the most important identifier or time column early.
4. Configure sorting.
5. Add grouping only when it helps scan the data.
6. Apply conditional formatting for operational states.
7. Preview the table at the intended dashboard size.

Common mistakes

Do not overload a table with every available field. A good table supports a decision; it is not a raw export.

Do not sort time-based operational tables oldest first unless users explicitly need historical order. Recent records are usually more actionable.

Related

- Widget Builder
- Filters
- Data Types

Dashboard Filters

Overview

Filters narrow dashboard data so users can focus on a time window, category, tenant segment, status, or selected value. They are essential for dashboards that serve more than one question.

How it works

A filter applies to widgets that can resolve the selected field. Time filters affect time-aware widgets. Dimension filters affect widgets with matching catalog fields. The chart-drill menu offers a cross-filter shortcut only when a supported chart-series click resolves to a dimension that matches a configured dashboard filter field.

Step-by-step

1. Decide whether the dashboard needs global or widget-specific filters.
2. Add a time filter for time-based dashboards.
3. Add dimension filters for key business categories.
4. Confirm that each target widget responds correctly.
5. Test empty states and default values.
6. Wait for the automatic persistence or compile indicator to finish after each change. The dashboard has no separate Save action.

Common mistakes

Do not add many filters before users ask for them. Too many controls make dashboards harder to read.

Do not filter on fields with unclear labels. Fix metadata first so users understand what they are selecting.

Related

- Charts
- Tables
- Drilldowns

Drilldowns

Overview

Drilldowns let users move from a supported chart-series point, bar, or segment into more detail. A **STAT** card has a separate whole-card action that opens all rows when its source can be resolved. Table rows and alert cards do not start this chart-drill workflow in the documented build.

How it works

A chart click resolves the clicked value and source context, then opens a menu. The entries depend on the catalog and widget configuration: **Drill into rows**, configured **Drill down** or **Drill through** targets, and **Drill by…** when a reachable dimension exists. A matching configured dashboard filter can also appear as a filter shortcut. When the click resolves but no destination is available, the menu says **No drill destinations available.**

A source-resolvable **STAT** card opens all rows directly instead of showing this chart-click menu. Drill detail uses the same environment, tenant, and governed catalog context as the dashboard.

Step-by-step

1. Open a dashboard containing a chart with rendered data.
2. Click a supported series point, bar, or segment.
3. Choose an available menu action such as **Drill into rows**, **Drill by…**, or a configured drill target.
4. If the menu says **No drill destinations available**, choose another source-resolvable chart point or have an author configure a target.
5. Inspect the result. Use the drill overlay's **Back**, dashboard breadcrumb, close, or Esc control to return.

For a source-resolvable **STAT** card, click the card itself to open all rows.

Common mistakes

Do not use drilldowns as a replacement for good dashboard design. The main dashboard should still show the most important signal.

Do not assume a drilldown has different data permissions. It follows the same environment, tenant, and data access context.

Do not expect a table-row click or an alert card to open this workflow in this build.

Related

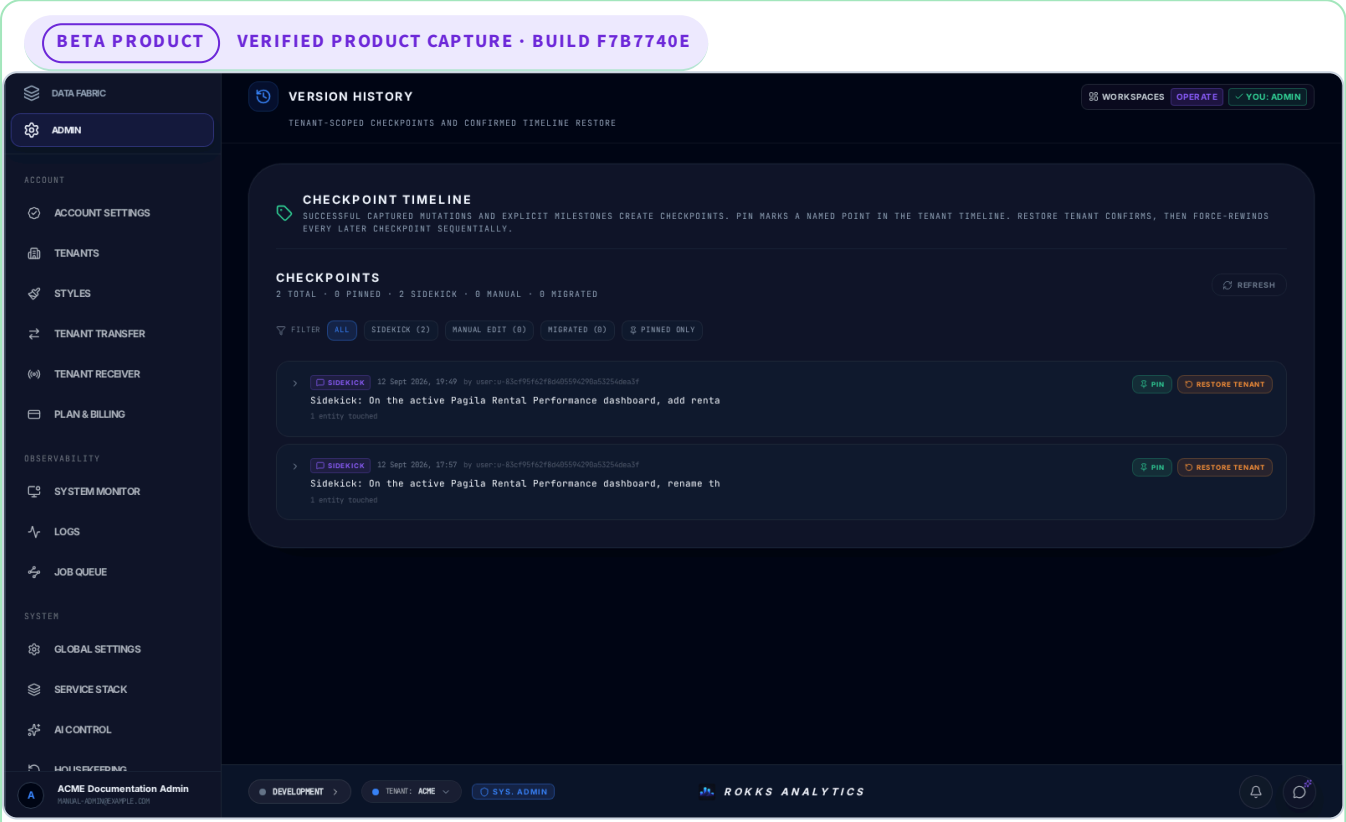
- Filters
- Charts
- Widget Builder

Version History

Overview

Version History lists checkpoints created by mutating Sidekick Agent turns, explicit milestones, and legacy migration entries. Ordinary manual edits and read-only Ask/Jury planning do not currently create checkpoints.

Version History is a tenant timeline, not a per-dashboard snapshot or an operational backup. Treat **Restore tenant** as a tenant-wide administrative operation.



Reviewed ACME Version History capture showing two Sidekick checkpoints created while editing the Pagila dashboard.

Captured 12 Sept 2026 UTC [Open documented view ↗]

How it works

Expanding a checkpoint shows its captured entity identifiers and a limited pre-change summary. It does not provide a complete visual or field-by-field comparison.

Restoring to a marker sequentially reverts every later checkpoint in the active environment and tenant, newest first. This is also true when the marker was selected from a dashboard-filtered History modal: later checkpoints may include other dashboards and resources. The selected marker itself is not reverted.

The confirmation dialog names the exact environment and tenant. A restore cannot be cancelled after confirmation, is not a cross-document transaction, and can stop after partially applying changes if a request fails or a conflict is detected. Changes made outside the checkpoint timeline are not detected and may be overwritten. Restored documents use merge writes, so a field added after capture may remain if that field did not exist in the captured snapshot.

Step-by-step

1. Open **Version History**.
2. Confirm the active environment and tenant.
3. Review the timeline of checkpoints.
4. Expand a checkpoint to inspect its touched-entity summary.
5. Coordinate with other users and verify that every later tenant change may be rewound.
6. Select **Restore tenant**, read the scope warning, and confirm only when the marker is the intended tenant state.
7. Wait for completion. If the operation stops, review the reported partial result and refresh affected resources before taking another action.

Common mistakes

Do not restore just because one dashboard looks unfamiliar. The operation is not limited to that dashboard.

Do not use Version History as a substitute for deployment backups, disaster recovery, or a complete audit log. Important environments also need operator-run backups, clear ownership, and change coordination.

Related

- Dashboard Collections
- Backup and Restore
- Sidekick

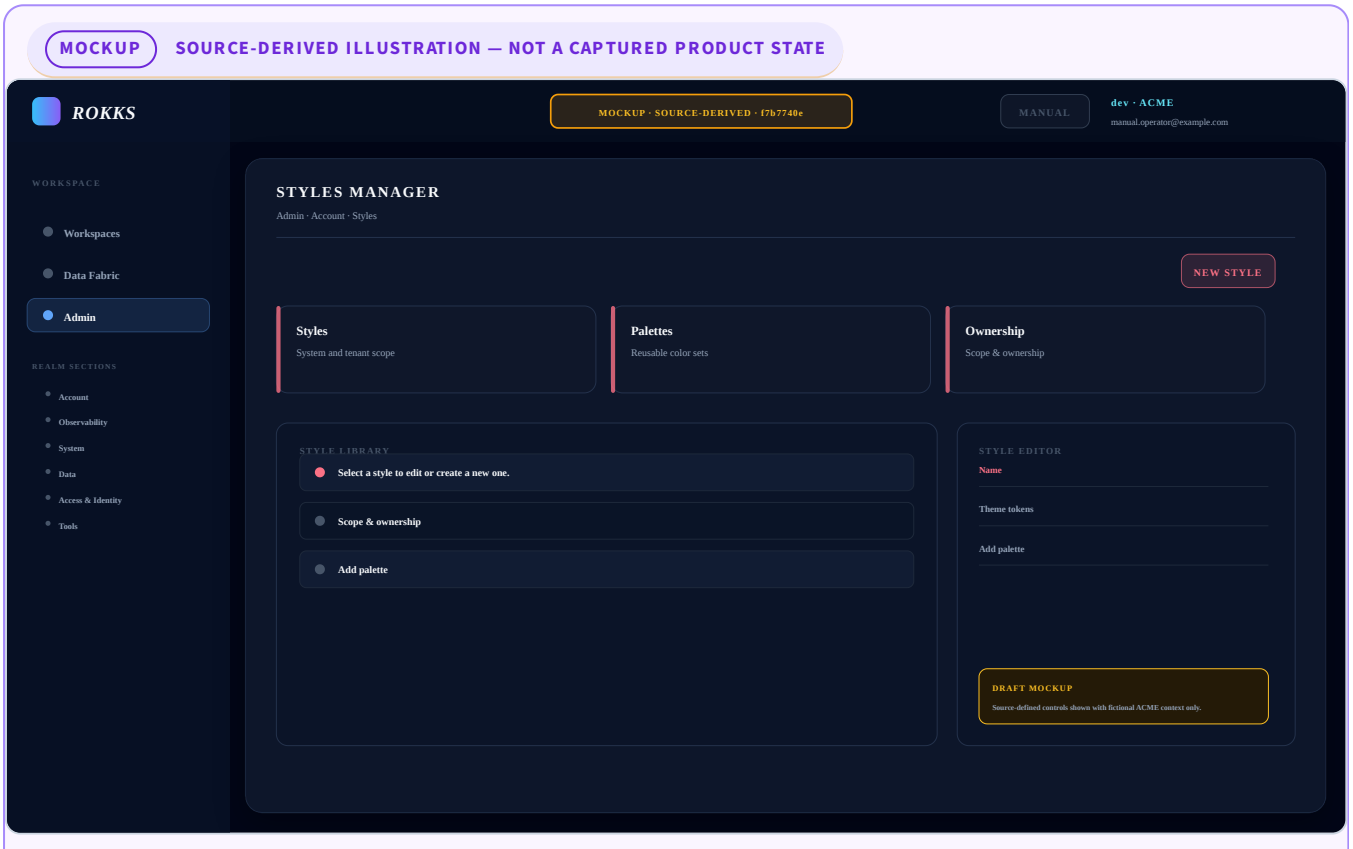
Styles

Overview

Styles control how dashboards look and how data states are communicated. Use them to keep dashboards consistent across teams and tenants.

How it works

Rokks separates action colors, identity colors, and data-state colors. Users normally choose themes, palettes, and widget options through the UI rather than editing technical settings. Data-state colors should represent meaning such as success, warning, failure, or informational state.



Styles Manager illustration showing scope, ownership, and palette controls. Source baseline: beta f7b7740e

Step-by-step

1. Open **Styles**.
2. Review the active style configuration.
3. Adjust approved visual options.
4. Expand the built-in **Preview** section and check its synthetic sample widgets.
5. Choose **Create Style** or **Save Changes** only after confirming contrast and readability in that preview.
6. Open representative real dashboards after saving to verify the applied style with actual content.

Common mistakes

Do not use color alone to encode a business scenario. Labels, form, and layout matter for accessibility.

Do not make every widget colorful. A restrained visual language helps users notice real exceptions.

Related

- Charts
- Tables
- Feature Matrix

SECTION 4

AI Assistance

2 topics

Sidekick

Overview

Sidekick is the chat-driven alternative to many point-and-click workflows in ROKKS. The Sidekick chat orb is available from the application shell, and an active dashboard gives it additional dashboard context. It can inspect product state in **Ask** mode and use shipped product tools in **Agent** mode to work with File Areas, governed tables, workspaces, dashboards, widgets, jobs, and selected connections.

Use this page as the canonical task cookbook. Feature pages link back to the relevant recipe instead of duplicating the chat workflow.

How it works

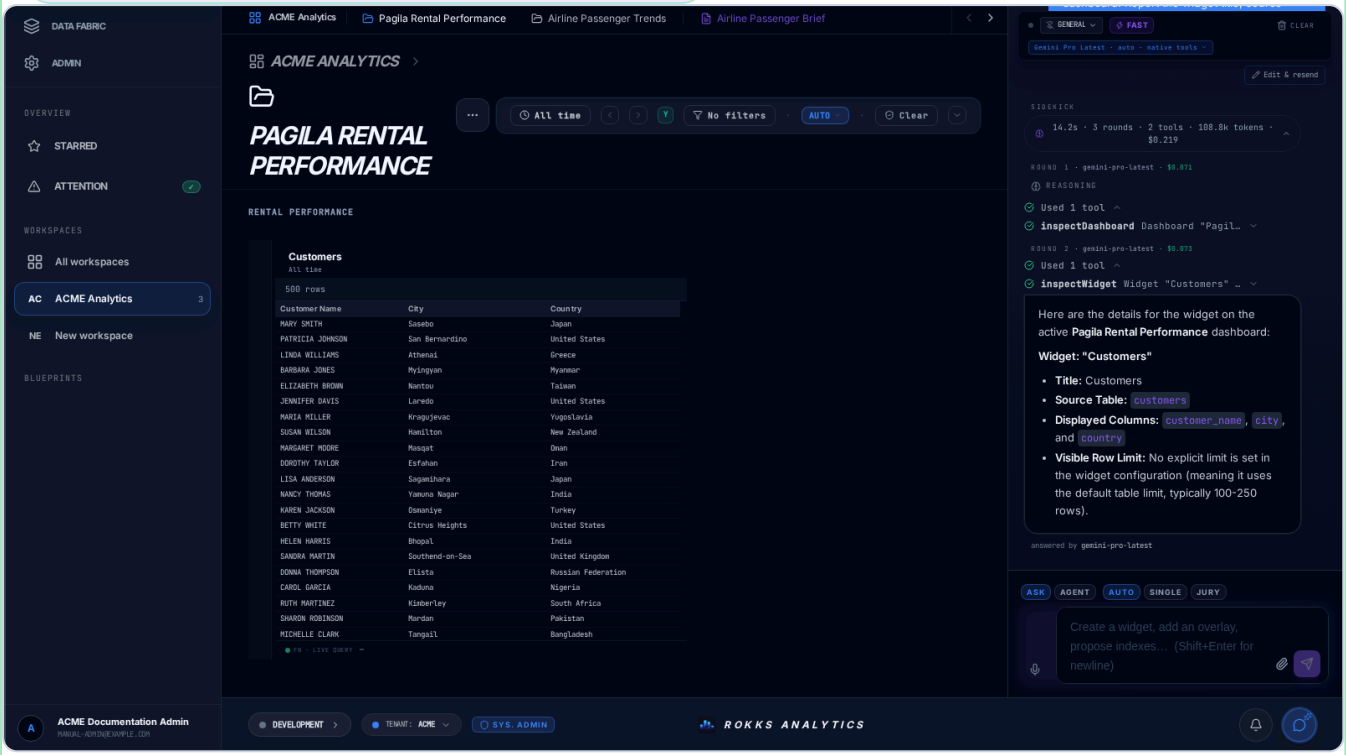
Sidekick uses the same product operations exposed to the application. It does not gain a separate route around tenant scope, permissions, validation, or Worker jobs.

Mode	What it can do	Use it when
Ask	Use read-only tools to inspect dashboards, tables, File Areas, connections, and jobs, then explain the result	You want analysis or a preflight check with no product mutation
Agent	Use allowed read and mutation tools and apply changes while the turn runs	You want Sidekick to create, configure, load, or update something
Agent + Jury	Produce the narrower proposal flow with an explicit **Apply** decision before mutation	You want candidate review before an Agent action

Write prompts as an outcome, a scope, constraints, and a stopping condition. For example: “In ACME’s active `dev` scope, inspect the Pagila File Area, report any failed job, and do not retry or change anything.” The final clause makes the boundary testable.

Real mutating Agent turns can capture document changes in a checkpoint and expose a **Revert turn** action. Read-only Ask/Jury planning does not open one. This is not a universal undo guarantee: jobs, SQL effects, external systems, ordinary manual edits, and changes not represented by the checkpoint may not be reversible. A multi-entity revert is sequential and can stop after partial progress; the chat is preserved when any entity remains unresolved.

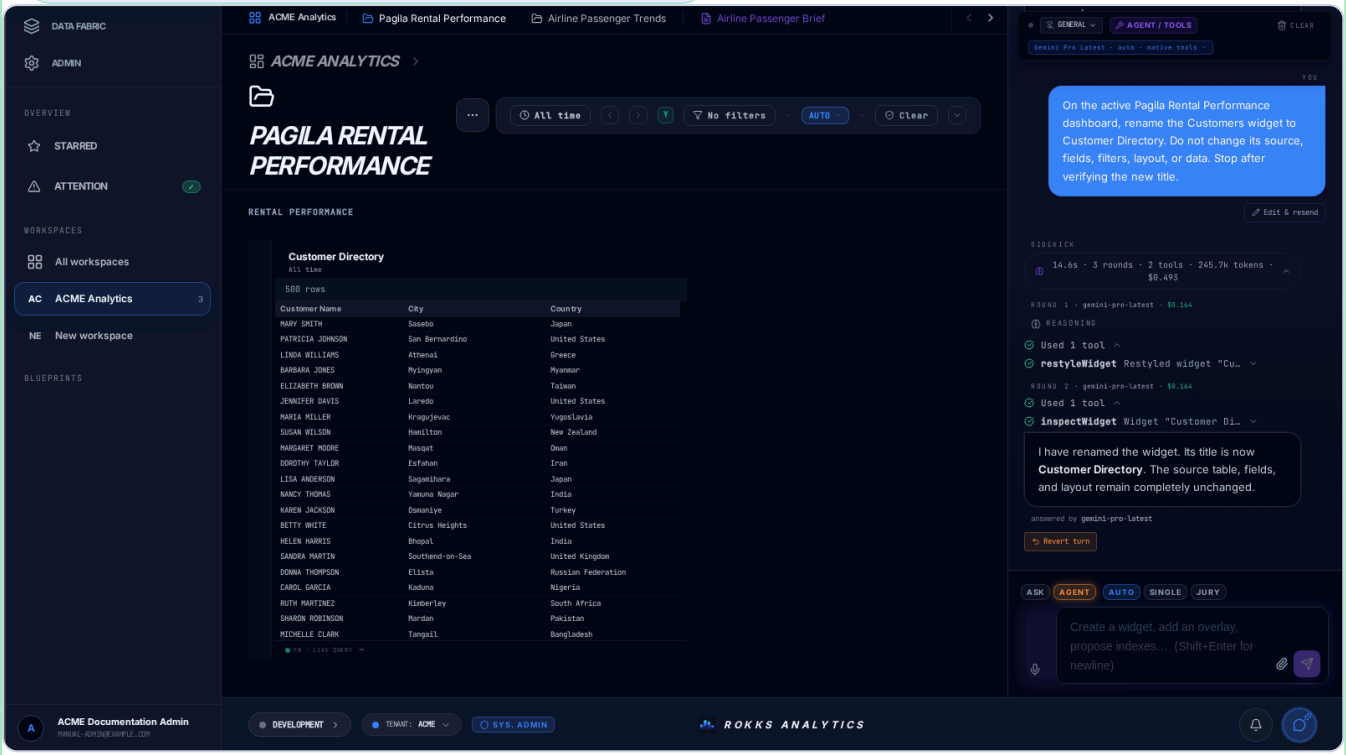
BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Real beta Sidekick result using inspectDashboard and inspectWidget beside the public Pagila Customers table.

Captured 12 Sept 2026 UTC [Open documented view ↗]

The capture records a real **Ask + Single** turn on the public Pagila dashboard. Sidekick used `inspectDashboard` and `inspectWidget`, then correctly identified the Customers widget, its source table, and its three displayed columns. Its final row-limit sentence is wrong for this build: the widget beside it visibly reports **500 rows**, not “typically 100–250.” This grounding defect and the unusually large token trace are tracked internally as product issue **#1370**. Until that issue is resolved and the result is recaptured, trust the visible product value and verify Sidekick answers against the inspected surface.



Sidekick Agent confirms a title-only update on the ACME Pagila dashboard and offers Revert turn.

Captured 12 Sept 2026 UTC [Open documented view ↗]

The next capture records a real **Agent + Single** follow-up on that same public Pagila dashboard. The prompt restricts the change to the widget title. Sidekick uses `restyleWidget`, re-inspects the result, and the dashboard visibly changes from **Customers** to **Customer Directory** without regenerating its query. The completed turn exposes **Revert turn**, so you can review the exact product state before deciding whether to keep this checkpoint-backed document change. The 245.7k-token trace shown for this small turn is also covered by product issue **#1370**; treat that number as unverified until the instrumentation is fixed or validated.

Step-by-step

1. Select the intended environment and tenant before opening the chat orb.
2. Open **Sidekick**. Use a global session for cross-product work or a dashboard-pinned session when the conversation should keep that dashboard context.
3. Choose **Ask** for a read-only preflight or **Agent** for tool-assisted work.
4. Name the target objects and fields. Include the source table, dashboard, File Area, or job when ambiguity is possible.
5. State what Sidekick must not do and where it must stop. A plan-review stop is useful before loading data; “inspect only” is useful for operational diagnosis.
6. Watch the visible tool activity. Treat a tool failure as a failed step even if the surrounding prose sounds confident.
7. Inspect the resulting product state. For a load, confirm the terminal job and output rows; for a widget, run the preview and inspect its fields, filters, and aggregation.
8. When **Agent** and **Jury** are both selected, review the proposal card and choose **Apply** only if it matches your intent.
9. If a checkpoint-backed document change is not right, inspect **Revert turn** or Version History and read the full scope warning before confirming.

File Area and constellation cookbook

Attach the source file to the Sidekick composer before sending an ingestion request. Use this two-turn pattern for the public Pagila JSON:

```
In Agent mode, use the attached pagila-rental-constellation.json. In the active
ACME scope, create a regular File Area named Pagila Samples if it does not
exist, then create a child Constellation named Pagila Rental Constellation.
Detect the file shape, ingest the attachment into that Constellation, wait for
the plan, and stop after previewing the generated six-table plan. Do not confirm
or load the plan.
```

Review the six nodes, five relationships, field types, and keys in the visible plan. The first turn deliberately stops before the forward-only load. If the plan is correct, continue with:

```
Confirm the Pagila Rental Constellation plan, start its load, and wait until all
six output tables are available. Report the terminal AREA_ETL_SYNC parent and
its child FILE_ETL outcome, then report the row count for each table. Do not
create a dashboard yet.
```

Sidekick can create the parent File Area and Constellation, ingest an attached JSON file, preview the area-level plan, submit the Constellation load, and inspect its progress. Use **Define Transformation** in File Areas when a Constellation node or edge needs manual correction; the chat field-edit action is for flat per-file plans, not multi-table node editing. A checkpoint does not undo loaded SQL rows or a submitted Worker job.

For other flat CSV files, ask Sidekick to create or reuse a regular File Area, ingest the attachment, preview the per-file plan, and stop before confirmation. Name any type, key, or unit changes in a follow-up turn. The Pagila maiden flight itself uses the six-table JSON.

Dashboard authoring cookbook

Start with an inspection turn when you do not yet know the exact table or column names:

```
In Ask mode, inspect the governed Pagila sources. Resolve the catalog relationship
from payments.rental_id to rentals.rental_id, then tell me whether
"payments"."amount" and "rentals"."rental_date" support monthly revenue. Do not
create or update anything.
```

Then switch to **Agent** and request one independently reviewable result:

```
Create or use the workspace ACME Analytics and create a dashboard named Pagila —
Rental Performance. On that dashboard, create one line widget named Monthly
Revenue using SUM of "payments"."amount" by month of "rentals"."rental_date".
Use the catalog rental_id relationship from payments.rental_id to
rentals.rental_id. Run the query and stop after the first widget is compiled.
Do not add filters or other widgets.
```

Inspect the first widget before asking for the next change. Follow-up requests can add a dashboard filter, update or restyle a widget, add an overlay, reorganize the layout, or publish a reviewed draft. Name the dashboard and widget on every follow-up when the session is not pinned to that dashboard.

Operations cookbook

Use **Ask** to diagnose without changing the queue:

```
List the recent jobs for the active ACME scope. Inspect the newest failed file
load and explain the failed step and its error. Do not cancel, retry, or submit
any job.
```

After you fix the underlying cause, switch to **Agent** and name the exact job when requesting **Retry** or **Cancel**. A retry creates more operational work; it is not a read-only follow-up.

Catalog and connection cookbook

Sidekick can find governed sources, inspect a table, rename catalog display metadata, set column units, and add or remove a table relationship. Use **Ask** first to resolve the exact source and field names, then use **Agent** for the selected metadata change.

For App Connect, Sidekick can list existing Airtable Constellations and start a supported sync. Create or edit the Airtable connection and its credential in the App Connect UI. Sidekick does not replace that point-and-click credential workflow. The same principle applies to Local Tables provisioning and identity administration: use their dedicated UI pages when no matching Sidekick mutation tool is available.

Common mistakes

Do not ask vague questions such as “make this better” when you need a specific business answer.

Do not assume a normal Agent turn waits for a separate Apply click. Use Ask when you want read-only help, and check the source, fields, filters, and chart type after an Agent action.

Do not treat a prose response as proof that work completed. Check the tool cards and then verify the resulting File Area, output table, dashboard, widget, or job in its product page.

Do not combine source ingestion, plan approval, six widgets, filters, and publication in one maiden prompt. Stop at meaningful review boundaries so a bad mapping cannot propagate through the rest of the task.

Do not use **Revert turn** as a promise to undo SQL loads, jobs, or external-system effects. It only applies to the checkpointed document changes named by the confirmation dialog, and a multi-entity revert can be partial.

Related

- Pagila Maiden Flight
- File Areas
- Widget Builder
- Job Queue
- AI Prompts
- Version History

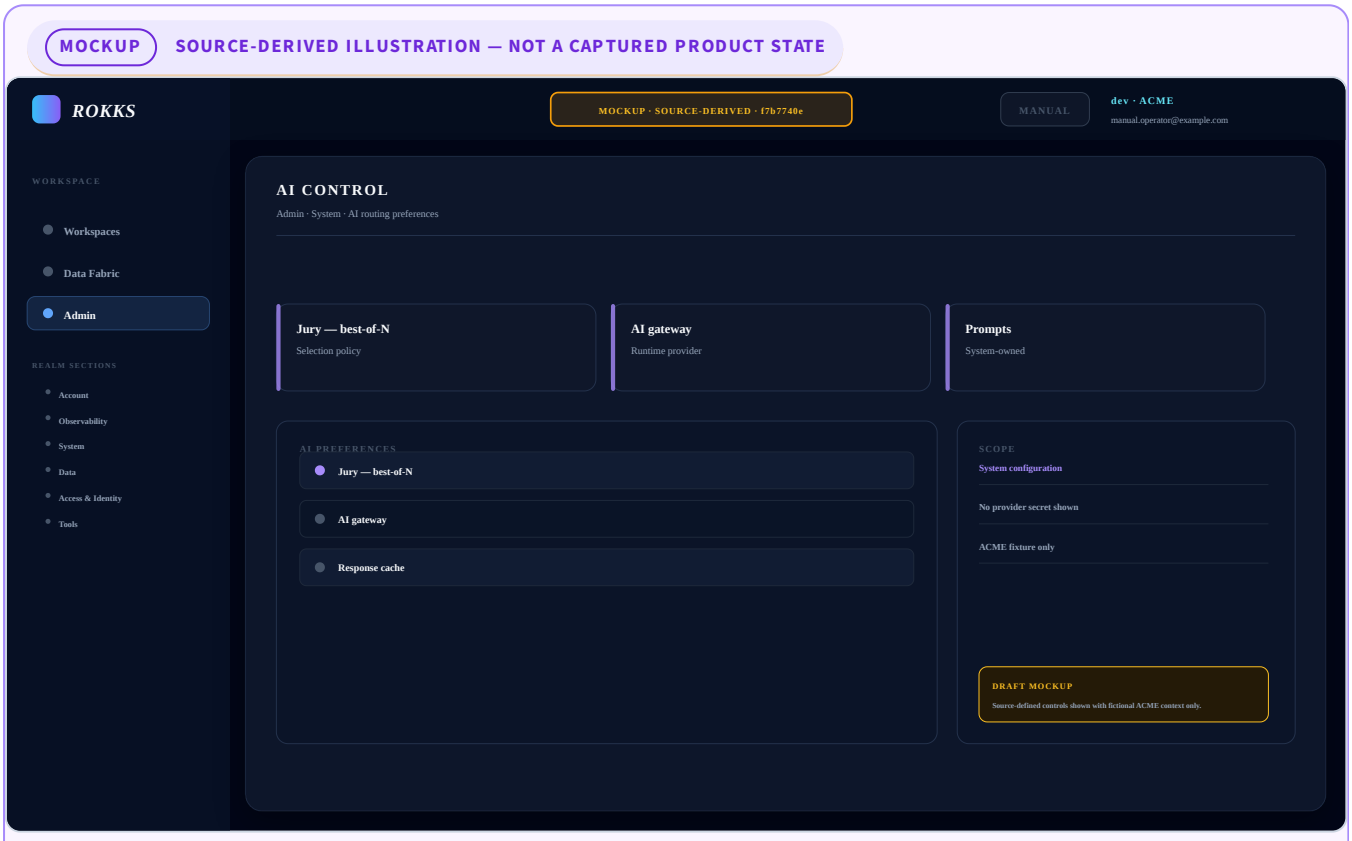
AI Control

Overview

AI Control is the dedicated page for AI *preferences* — the cross-environment knobs that decide how the platform uses AI:

- **Model routing** — the routing strategy (free-only / free-first / economy / quality), a catch-all default model, and per-need First/Fallback pins (vision, reasoning, agent, fast, long-context).
- **The Jury** — the best-of-N candidates and judges that compete on a turn.
- **Response cache** — serve byte-identical re-requests for free.
- **Prompts** — domain knowledge injected into each AI context (dashboard generation, widget refinement, the Sidekick chat, the dashboard consultant).

Everything here is one shared configuration, persisted to the registry and applied across all environments.



AI Control illustration based on the documented beta source. Source baseline: beta f7b7740e

> **Credentials live elsewhere.** Provider API keys, each provider's enabled-model pool, and billing/admin secrets stay on **Service Stack** → **AI** — they are per-environment secrets, not preferences. AI Control links there, and back.

How it works

When the platform makes an AI request it resolves a **need** (e.g. `agent`, `vision`, `reasoning`) to a concrete model through the routing chain:

1. **Strategy filter** — narrow the eligible pool to models that match the selected tier (free-only, free-first, economy, quality).
2. **Per-need pin** — the First model for that need is tried first; if unavailable (quarantined by a provider-supplied `Retry-After`) the gateway falls over to the Fallback in the same request, then to the catch-all default. This failover is **in-turn**, but it is not a guarantee of an answer: a request with no resolved model is rejected, and an exhausted routing chain returns an error.
3. **Jury** (optional) — when enabled, N candidates independently answer the same prompt and a judge model scores them; the highest-scoring answer is returned.
4. **Response cache** — cache-equivalent requests are served without a provider round-trip. The key covers more than model and messages, including provider, environment, tools and tool configuration, generation configuration, and other request context that can change the answer.

Provider quarantines use only the window the provider signals (`RetryInfo` / `Retry-After`); no invented cooldown durations are applied. A quarantine that carries no explicit window is not applied — the request goes to the provider.

How prompts work

Each context has a prompt slot. A blank slot uses the built-in default; typing a value overrides it. Clearing the field restores the default.

Slot	Injected when
Global prompt	Prepended to every AI request
Dashboard generation	The AI builds a full dashboard from a schema
Widget refinement	The AI edits or rewrites a single widget
Sidekick chat	Every Sidekick conversation, as persistent context
Dashboard consultant	The structured business-case consultant flow

Step-by-step

1. Open **Admin** → **System** → **AI Control** (next to Service Stack).
2. Set the routing strategy, default model, and per-need pins under **Model routing**.
3. Curate candidates and judges under **Jury** (optional).
4. Toggle the **Response cache** and tune its TTL / max entries (optional).
5. Edit any **Prompt** slot to steer a context; leave a slot blank for its default.
6. Changes are shared across all environments — persisted to the registry.

Common mistakes

Do not treat AI output as final. Users should review generated transforms, widgets, and explanations.

Do not paste secrets into a prompt — prompts are stored in the registry config blob, not the secret store. API keys belong on **Service Stack** → **AI**.

Related

- Sidekick
- Widget Builder
- Worker Jobs

SECTION 5

Operations

6 topics

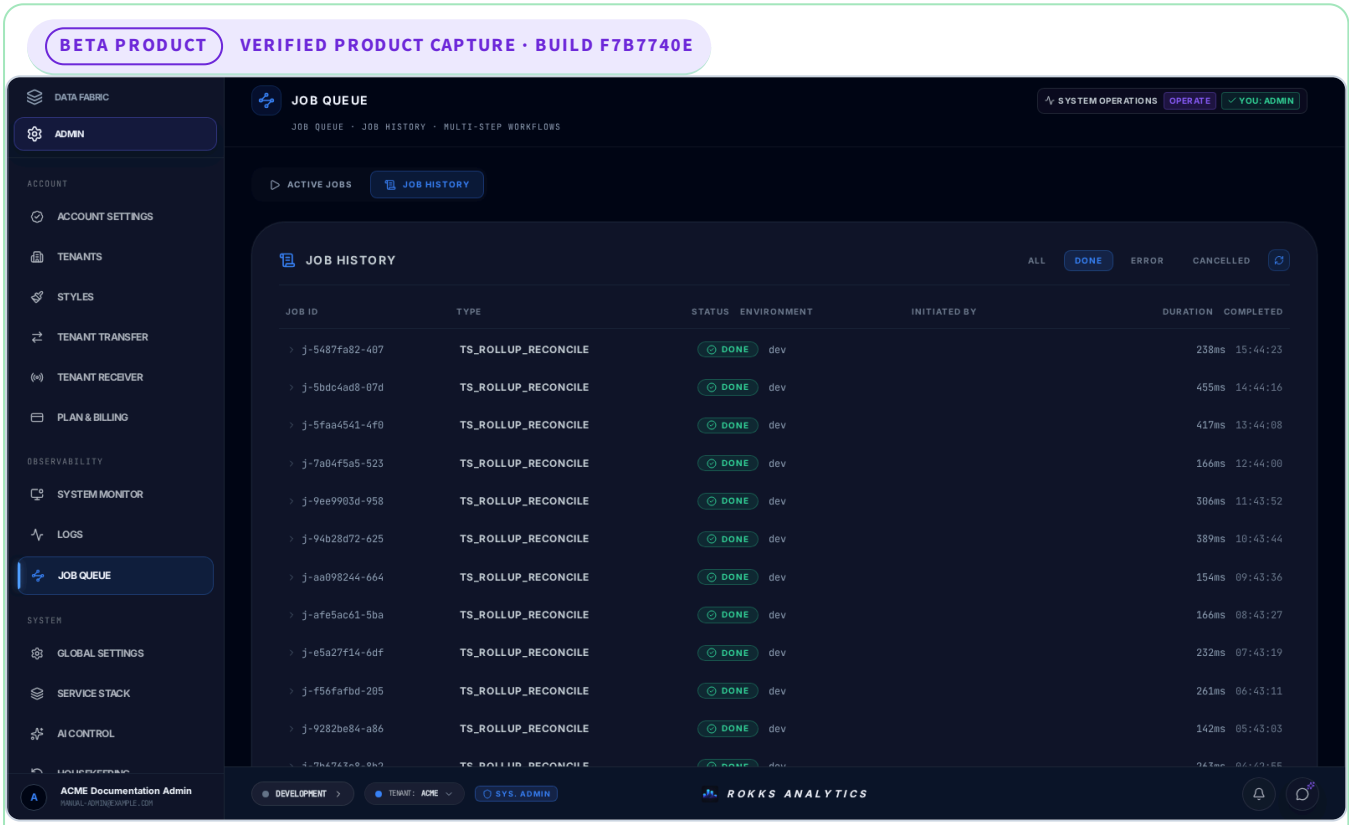
Job Queue

Overview

The Job Queue shows long-running work such as file loads, web syncs, AI generation, tenant operations, and maintenance tasks.

How it works

Each job has a type, status, progress, steps, result, and error if something failed. Jobs continue server-side even if the browser closes. The queue is the best place to check whether work is active, complete, cancelled, or failed. A terminal status describes that job handler; for a parent/child workflow, DONE can mean that the parent safely queued a child rather than that every downstream effect has finished.



Job History with DONE selected and completed roll-up maintenance jobs visible. This is not evidence that the Pagila import has completed.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

1. Open **Job Queue**.
2. Use the status pills when you need to narrow by state, then scan the **Type** column for the relevant job. This page has no job-type filter.
3. Open the job details.
4. Review steps and progress.
5. If failed, read the error and affected resource.

6. Fix the source issue and rerun the user action if needed.

Sidekick can list and inspect jobs in **Ask** mode without retrying them. Use the operations cookbook to keep diagnosis separate from an explicit **Agent** request to retry or cancel one exact job.

Common mistakes

Do not start the same operation repeatedly because a spinner is still visible. Check the job queue first.

Do not ignore partial failures. A job may complete some steps and fail on final persistence or cleanup. When end-to-end completion matters, inspect the result, any child job, and the affected resource instead of relying on the parent status alone.

Related

- Worker Jobs
- Web API Streams
- Backup and Restore
- Sidekick task cookbook

Alerts

Overview

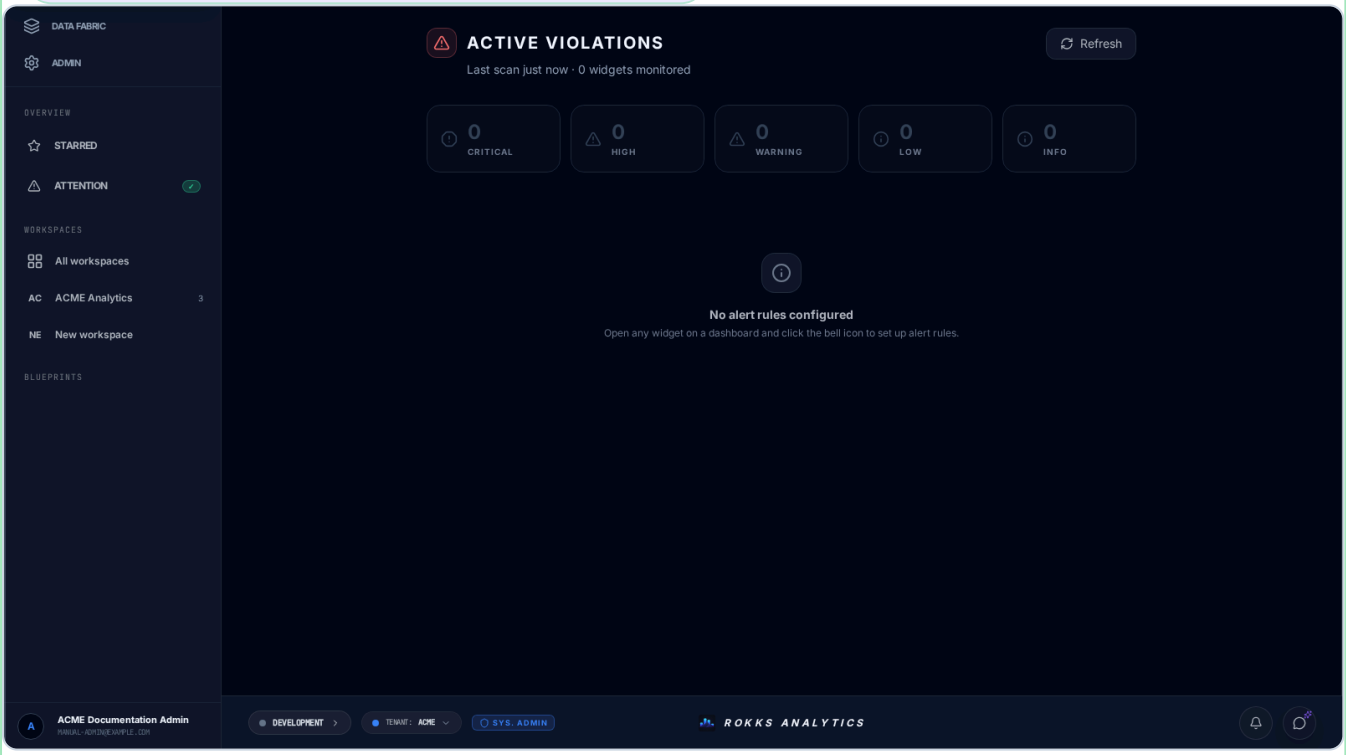
Alerts monitor governed data and notify users when a rule changes state. Use alerts for thresholds, stale syncs, anomalies, service indicators, or operational signals that require attention.

How it works

An alert rule evaluates data in the active scope and records state per rule or group. Sentinel emits browser events immediately and delivers configured external channels directly when the alert changes state. Notification delivery does not pass through the Worker job queue.

The reviewed beta capture shows the real **Active Violations** overview before any alert rules have been configured.

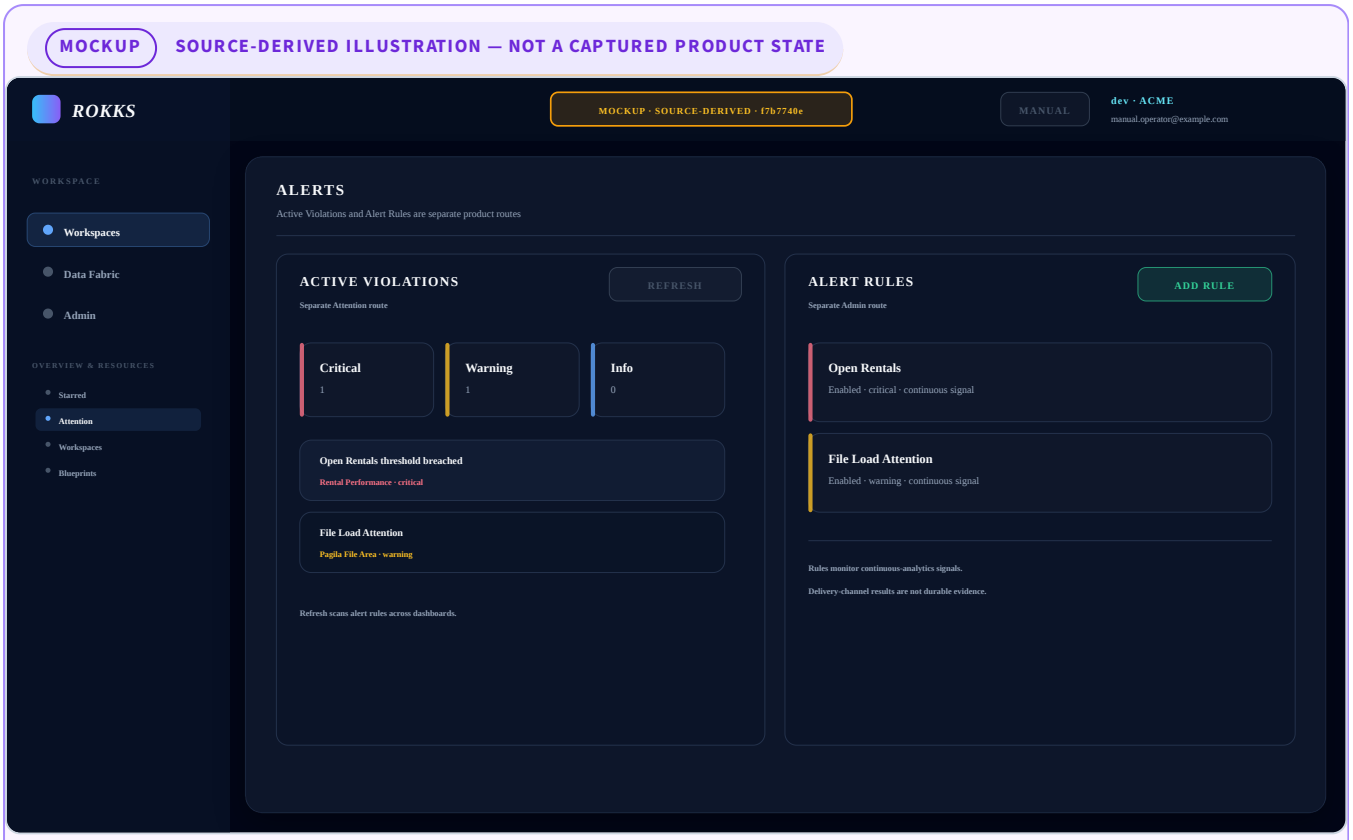
BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Reviewed ACME Active Violations capture showing zero monitored widgets and no configured alert rules.

Captured 12 Sept 2026 UTC [Open documented view ↗]

The rule-management illustration remains a labelled source-derived mockup until a configured Sentinel rule can be captured safely.



Alerts illustration based on active-violation and Sentinel surfaces. Source baseline: beta f7b7740e

Step-by-step

1. Create and run the required continuous-analytics signal before creating its alert rule.
2. Open **Workspaces** → **Attention**, choose **Manage rules**, then choose **Add rule** or select an existing rule to edit it.
3. Select the existing **Signal to watch**.
4. Configure the condition and threshold, breach duration, repeat-notification cooldown, and severity.
5. Select **Browser SSE** and, when configured, **Pushover** under **Notify via**.
6. Set the rule to **Enabled** and choose **Save**. This editor has no test action.
7. Monitor the saved rule and its current state on the rules page. Return to **Attention** for breached widgets. For external-channel delivery troubleshooting, use provider-side evidence or ask an operator to inspect Sentinel logs.

Common mistakes

Do not alert on noisy fields without grouping or thresholds. Too many alerts reduce trust.

Do not assume a missing external notification means the rule did not trigger. Rule state is visible in Rokks; external delivery outcomes are not currently persisted or shown in this page.

Related

- Job Queue
- Starred and Attention
- Worker Jobs

Service Stack

Overview

The Service Stack page has two operator surfaces: **Config** and **Diagnostics**. Config is the home for environment and persistence-instance assignments. Diagnostics runs the supported health and advanced test suites for platform services.

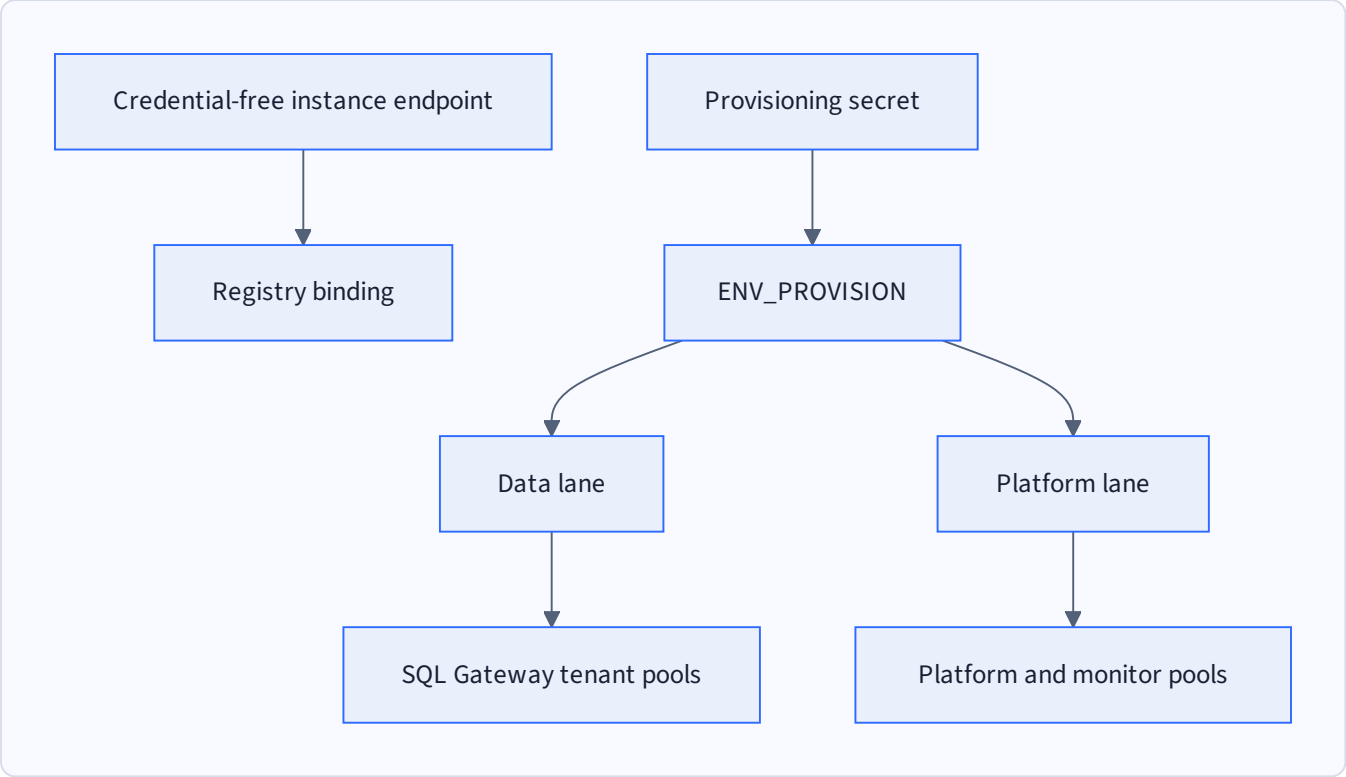
How it works

Config shows which PostgreSQL and document-storage instance each environment uses, including bundled and externally managed targets. Diagnostics shows live status and age, then lets an operator run each service's health or advanced checks. Those are the complete Service Stack modes; read-only architecture inspection lives in Live Service Topology.

Instance records deliberately contain only network endpoints (`protocol://host[:port]`). Credentials, database names, and TLS policy live in Registry secrets and are never displayed as instance URLs. PostgreSQL uses four separate authority lanes:

Dev/appliance deployments declare bundled ALPHA, BRAVO, and CHARLIE persistence slots; beta/production currently declare only CHARLIE. Only active, deployment-enabled slots are eligible for Setup. Bundled identity cannot be added, renamed, re-pointed, or deleted through the console. **Add instance** creates an external target only. If declared bundled inventory is missing or invalid, Setup stops and reports the Registry problem instead of trying to recreate infrastructure at runtime.

Lane	Purpose
Data	Tenant reads, writes, and bounded DDL under a least-privilege environment role
Platform	`_platform` control-plane data under a dedicated non-superuser role
Provisioning	Database/role creation only; never admitted to a normal pool
Read replica	Optional exact standby; absence sends reads to the primary

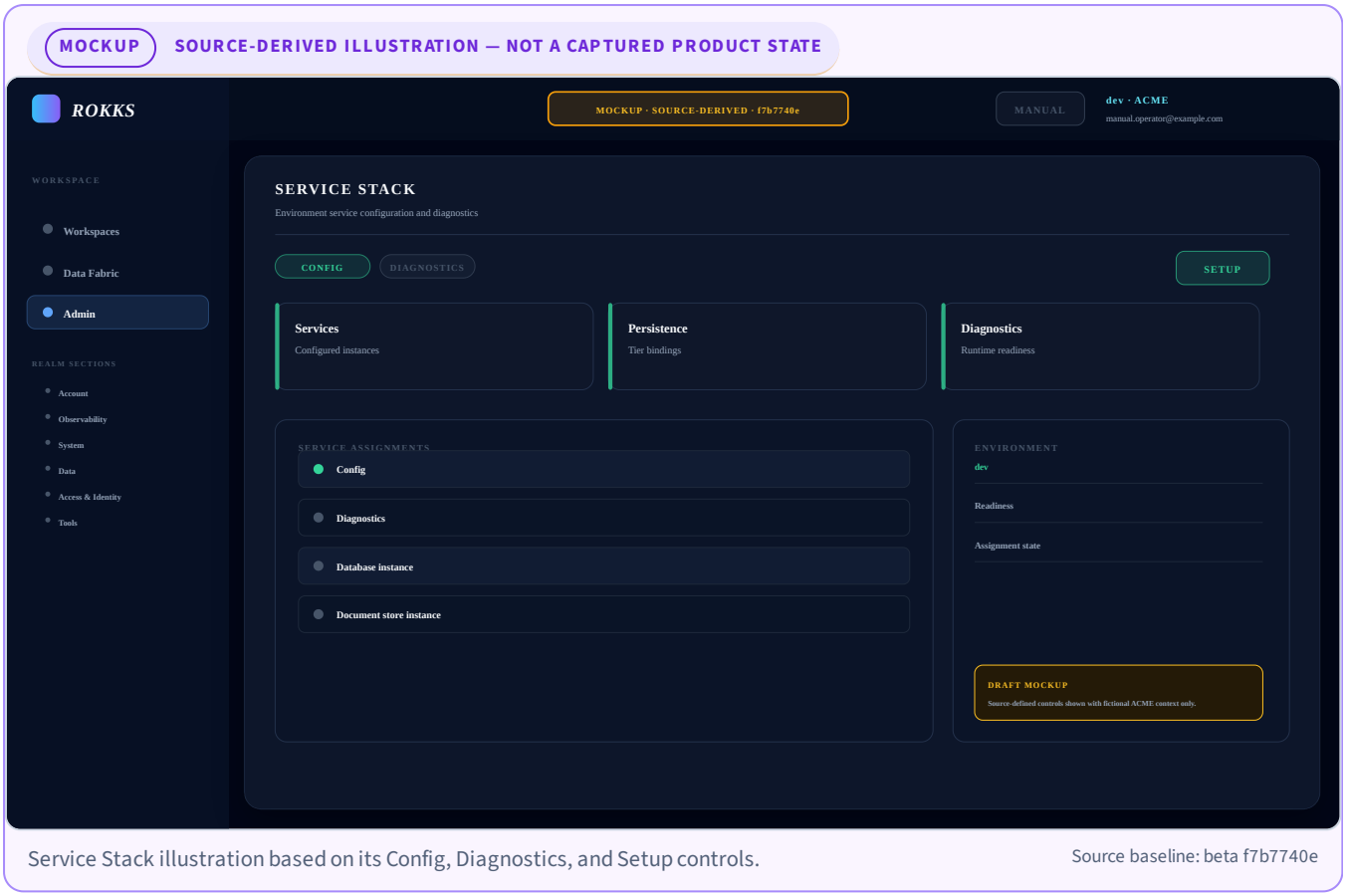


Environment toggles and persistence assignments apply immediately after Registry accepts them. There is no separate page-level Apply or Discard step. Registry rejects missing instance ids and SQL/document-store type mismatches before changing the environment; the page keeps the previous value and surfaces the error.

The Setup Wizard preserves an exact existing binding. With one eligible bundled candidate it may preselect that candidate; with several it requires an explicit tier choice and shows the exact label, Compose service, and instance id in review. It never assigns by response order. MonIO is 1:1, so a tier already claimed by another environment is unavailable.

Applying Setup records enabled environments as **PROVISIONING** and queues `ENV_PROVISION`; submitting a job is not the same as reaching **READY**. Terminal Worker-to-Registry status reconciliation is still a prelaunch hardening item, so verify the job outcome and do not treat the wizard closing as completion.

Registry control calls and persistence readiness probes have a five-second deadline. **Test All Connections** finishes only after every participating card has settled. A failed maintenance-state read preserves the last confirmed state; if none is known, the card shows **Unknown** and disables the maintenance mutation.



Service Stack illustration based on its Config, Diagnostics, and Setup controls. Source baseline: beta f7b7740e

Step-by-step

1. Open **Service Stack**.
2. Use **Config** to confirm the selected environments and their persistence assignments.
3. Resolve missing credentials or intentionally unbound environments before inviting users.
4. Open **Diagnostics** and review live service status.
5. Run Health checks first, then Advanced Diagnostics only when deeper evidence is needed.

Common mistakes

Do not treat an intentionally unbound or disabled environment as an outage. Configured deployment variants do not need every optional persistence target.

Do not run advanced diagnostics repeatedly without reason. They may call real downstream systems.

Related

- Monitoring
- Architecture
- Troubleshooting

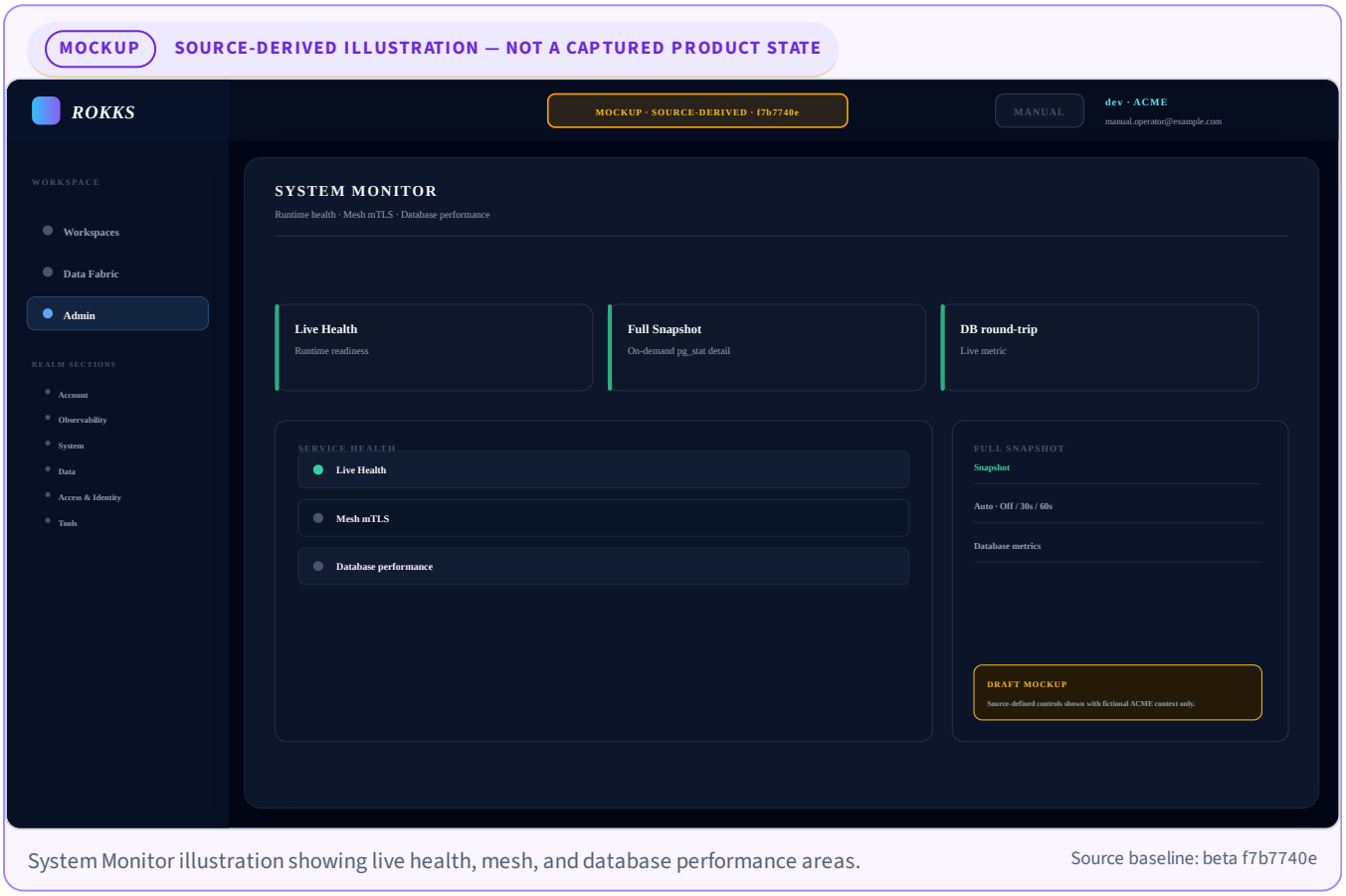
Monitoring and Logs

Overview

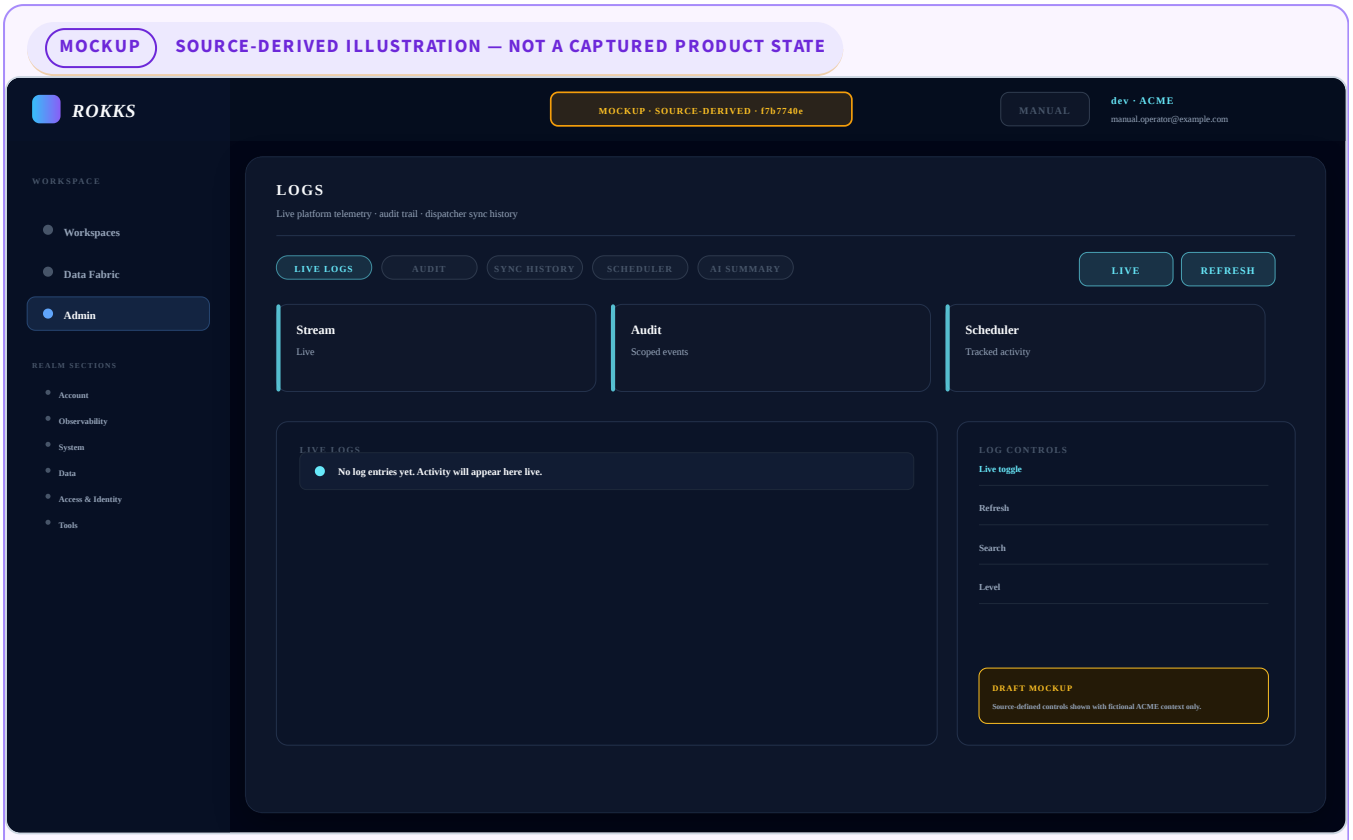
Monitoring views show whether the platform is healthy and what recently happened. Use them when users report failures, stale data, slow jobs, or missing dashboards.

How it works

Health checks show service readiness. Logs show operational events and errors. Job records show long-running work. Together they answer three questions: is the system reachable, did the requested work start, and where did it fail?



System Monitor illustration showing live health, mesh, and database performance areas. Source baseline: beta f7b7740e



Logs illustration showing live, audit, sync, scheduler, and AI summary tabs. Source baseline: beta f7b7740e

Step-by-step

1. Start with the user-visible symptom.
2. Check the Job Queue for related background work.
3. Check monitoring for affected services.
4. Review recent errors around the time of the issue.
5. Fix the source problem or escalate with the job and error details.

Common mistakes

Do not troubleshoot from browser symptoms alone. Many user-visible errors are downstream service, job, or scope issues.

Do not treat a healthy service as proof the workflow succeeded. A service can be healthy while a specific job failed.

Related

- Job Queue
- Service Stack
- Troubleshooting

Backup & Restore

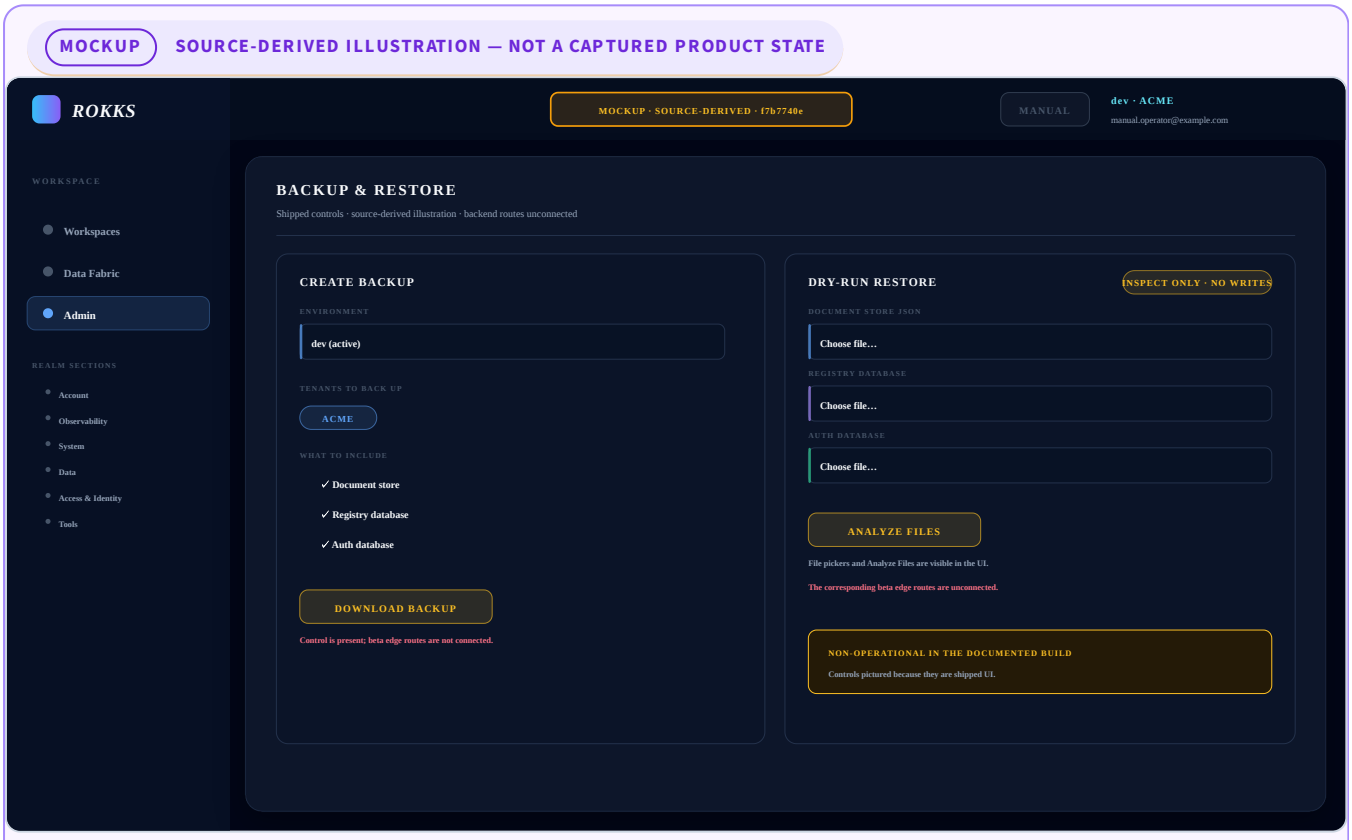
Overview

The appliance-only Backup & Restore page is an inspection aid, not an operational backup or restore mechanism.

How it works

The current Edge routing does not expose the page's backup-export endpoints or its Registry/Auth SQLite inspection endpoints. Do not rely on the visible **Create Backup** controls: they are not connected to an operational backend.

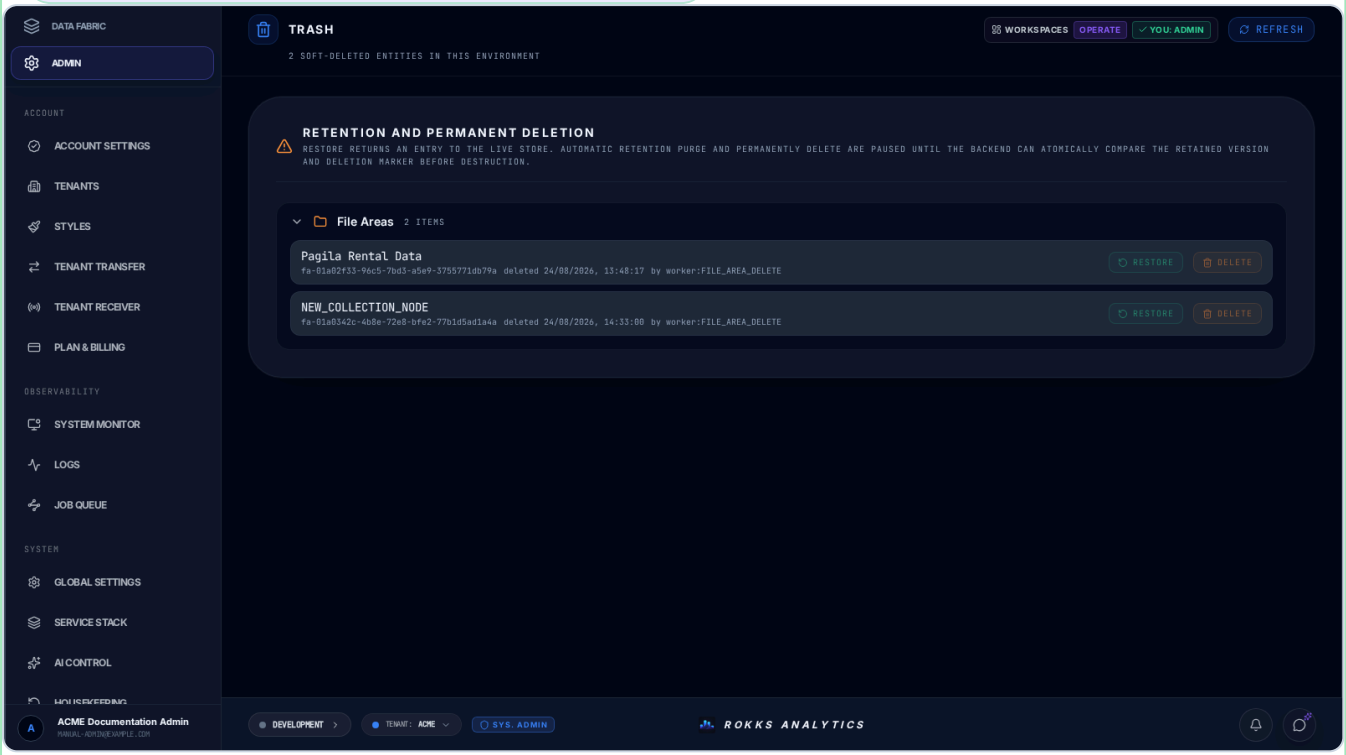
The working inspection path reads a document-store JSON backup locally in the browser and summarizes its metadata. It performs no writes and does not execute a restore. Deployment backups and recovery remain operator-run procedures outside this page.



Backup and Restore illustration; shipped controls are shown while unconnected beta routes remain visibly disclosed.

Source baseline: beta f7b7740e

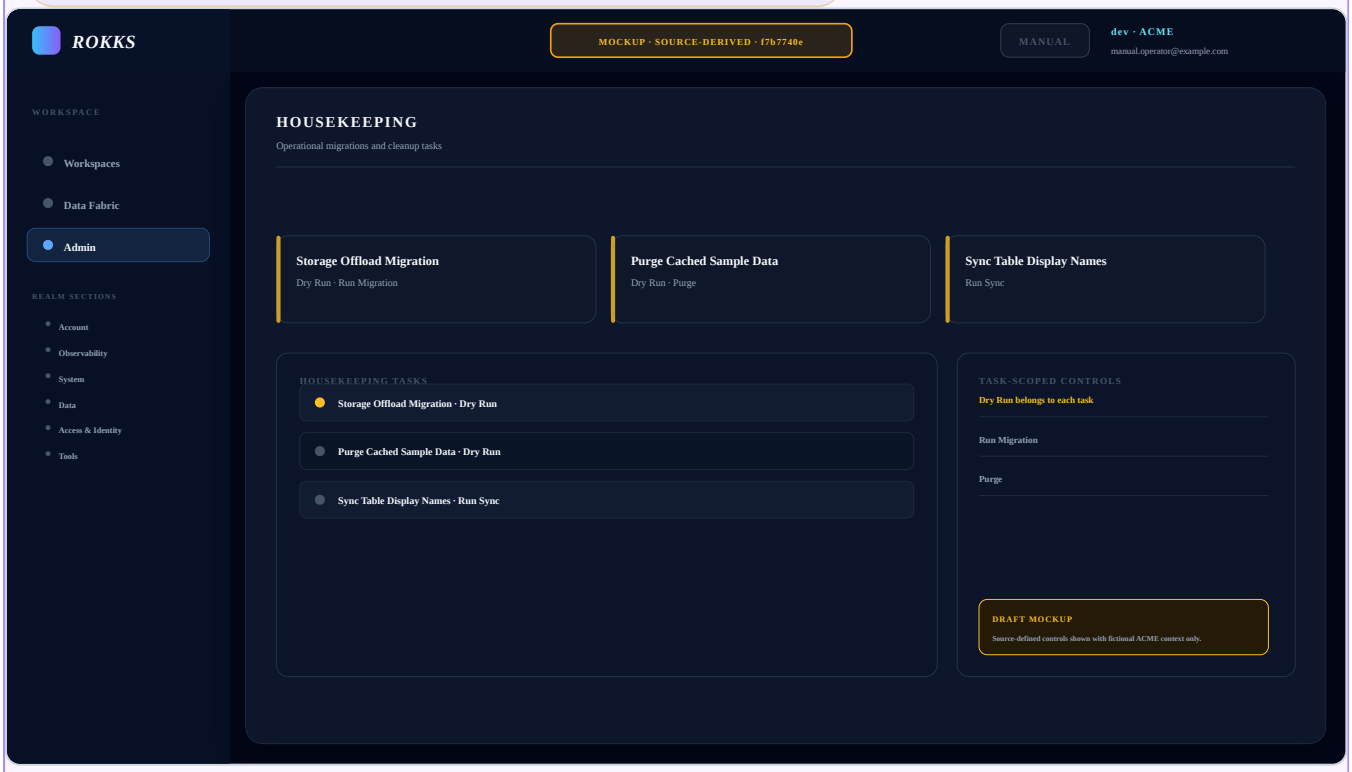
BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Reviewed ACME Trash capture showing retention guidance and restore/delete controls for two sample File Areas.

Captured 12 Sept 2026 UTC [Open documented view ↗]

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Housekeeping illustration showing operational migration and cleanup tasks.

Source baseline: beta f7b7740e

Step-by-step

1. Before a risky change, ask an authorized operator to create a deployment backup using the deployment recovery runbook.
2. On an appliance deployment, open **Admin** → **Account** → **Backup & Restore** only if you already have a document-store JSON backup to inspect.
3. In **Dry-run Restore**, select that JSON file.
4. Review its reported environment, tenants, collections, and record counts.
5. If recovery is required, hand the file to an authorized operator and follow the deployment-specific recovery runbook. The UI does not apply it.

⚠Caution

Registry and authentication database inspection is also not connected through Edge. A file picker appearing in the UI does not make that path operational.

Common mistakes

Do not assume Version History is a single-dashboard recovery tool. Its restore operation rewinds every later checkpoint in the selected environment and tenant and can stop after partial progress. Inspect the tenant-wide impact before using it.

Do not mistake a successful dry-run inspection for a completed restore. Inspection validates and summarizes files; an operator-run recovery is a separate procedure.

Do not treat a successful browser download prompt as proof of a backup. The export routes are unavailable until they are deliberately wired, authorized, and end-to-end tested.

Related

- Version History
- Tenants
- Troubleshooting

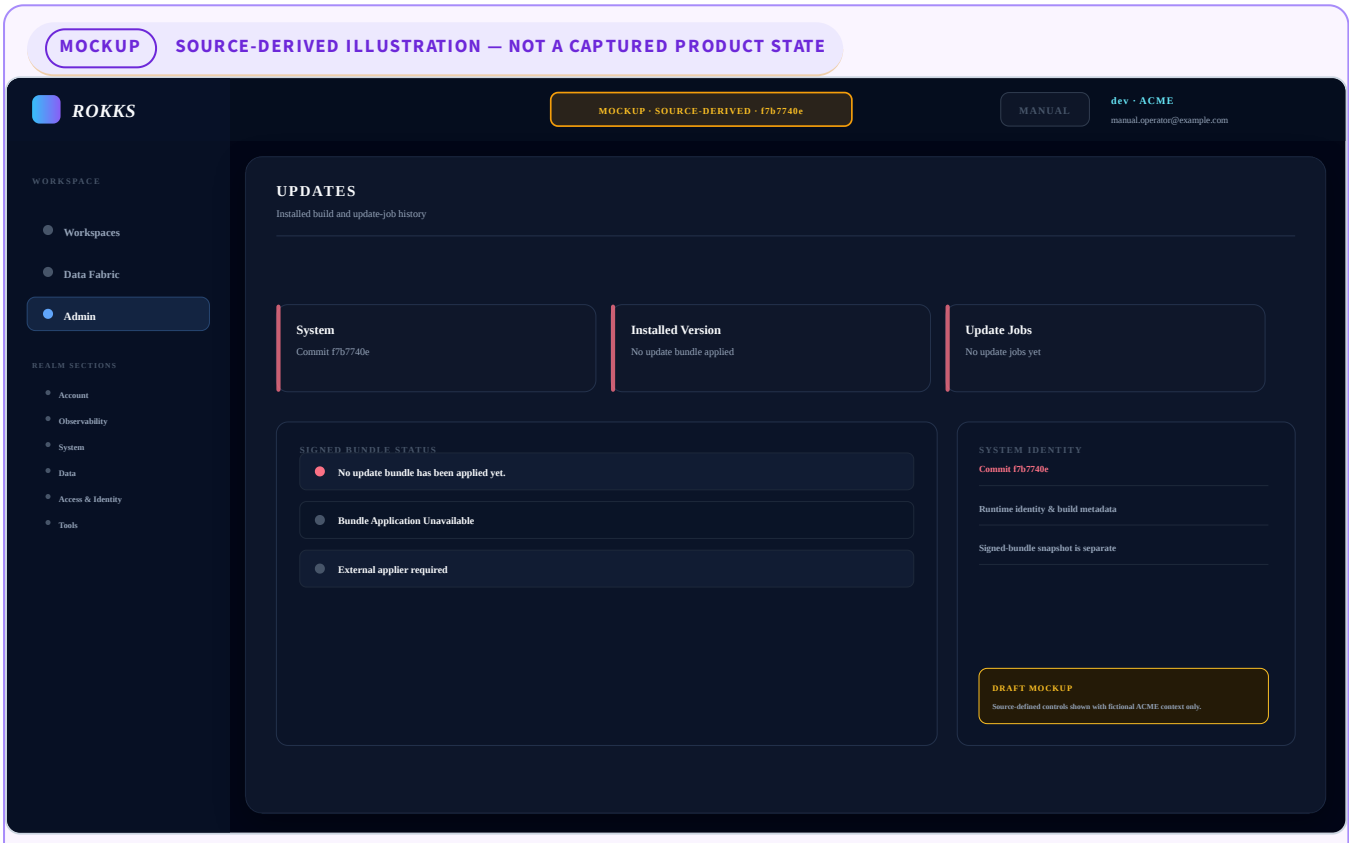
Updates

Overview

The appliance-only Updates page helps system owners inspect the running build identity, any recorded signed-bundle installation snapshot, and earlier update jobs. It is an operational status page, not a normal user workflow.

How it works

The current release retains the signed-bundle format and verifier, but deliberately disables in-process application. The Worker cannot safely replace its own container and then finish health verification or rollback. On an appliance, the page therefore shows **Bundle Application Unavailable** until a host-owned external applier is delivered. Cloud deployments continue to update through their deployment pipeline.



Updates illustration showing the current fail-closed unavailable application state. Source baseline: beta f7b7740e

Step-by-step

1. On an appliance deployment, open **Admin** → **System** → **Updates**.
2. Review **System runtime identity** and the installed snapshot, if one exists.
3. If the page shows **Bundle Application Unavailable**, do not attempt to bypass it or post a bundle directly to the API.
4. Follow the deployment channel operated for your installation (for example, your managed CI/CD pipeline).

5. Before an externally managed deployment, check Service Stack and Job Queue health and schedule any required user communication.
6. After deployment, confirm Service Stack health, open a dashboard, and run a small query.

Common mistakes

Do not interpret the presence of bundle-building or verification code as an operational in-app updater. `applyAvailable: false` is a hard safety boundary, not a configuration prompt.

Do not skip a smoke test after an externally managed update. Open a dashboard, run a small query, and check the Job Queue.

Related

- Service Stack
- Monitoring
- Deployment Topologies

SECTION 6

Administration

6 topics

Access Inventory

Overview

Open **Access Inventory** in Administration to reach the **API Policy** page. It provides read-only inspection of the policy loaded by the deployment: API functions, deployed grant clauses, required scope, and declared object and row access constraints.

Policies change through repository review and deployment. This page cannot edit, publish or activate them. The instructions and illustration below describe the newer source baseline displayed beneath the illustration; they do not claim a new capture of the rest of the manual.

How it works

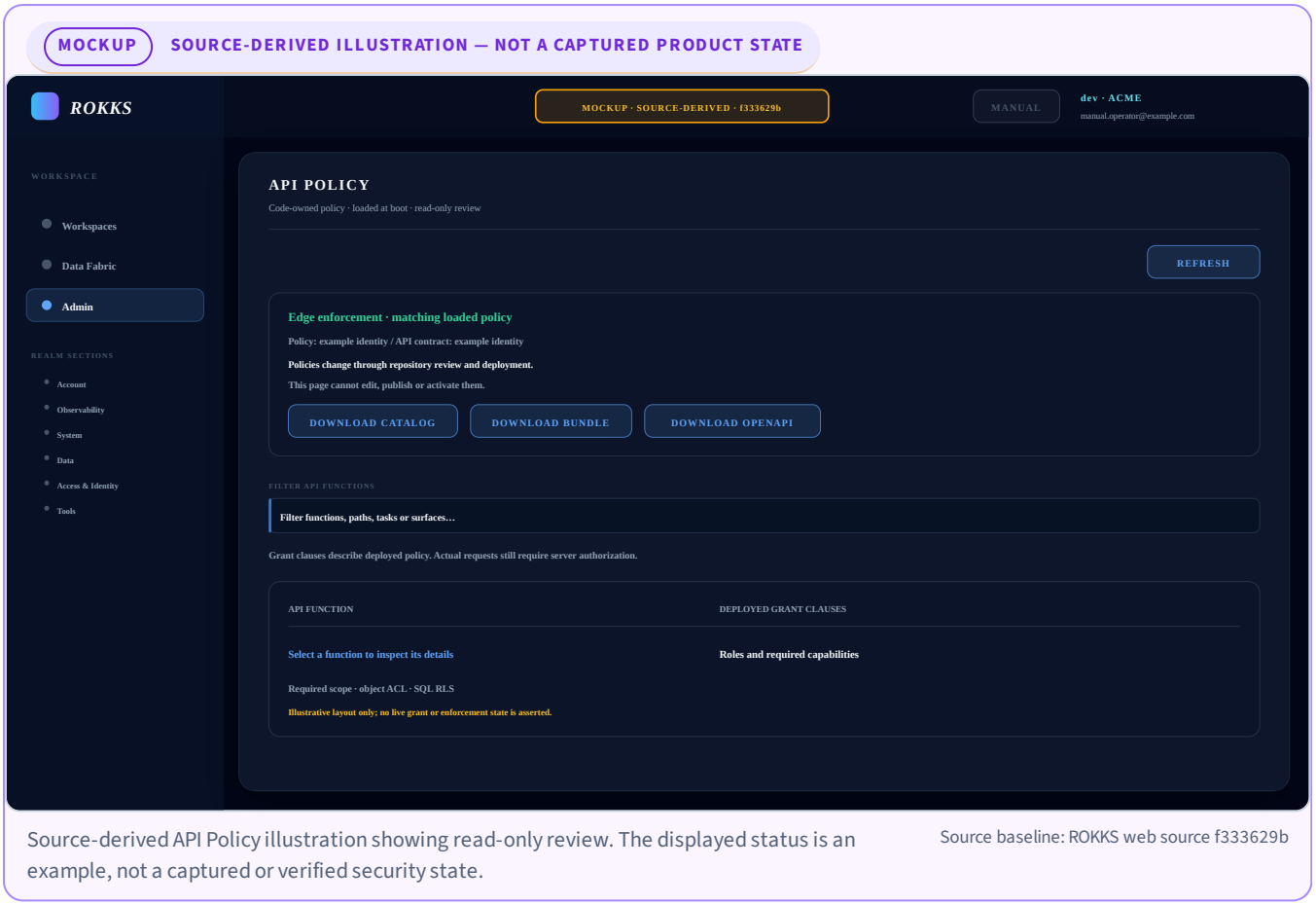
The status card shows the loaded **Policy** identity and **API contract** identity. **Edge enforcement · matching loaded policy** means the page received an available policy whose identity matches the policy reported by Edge in enforcement mode. **Policy / Edge identity unavailable or mismatched** means that comparison has not succeeded; it is not confirmation that requests are permitted.

The function table has two columns:

- **API function** identifies an action and its HTTP method, service, and path.
- **Deployed grant clauses** shows the configured roles and required capabilities for that action. An unavailable grant remains labelled **Unavailable**.

Filter API functions narrows the displayed rows by action, operation, path, task, or surface. Selecting a function opens an inline explanation with its required environment and organization scope, declared object ACL and SQL RLS obligations, and the deployed grant, function, and operation details. **Close** dismisses that explanation.

The status card offers **Download catalog**, **Download bundle**, and **Download openapi**. These retrieve the operation catalog, deployed policy bundle, and public OpenAPI JSON respectively. Each request remains subject to its server-side access checks. A download exports information; it does not change or activate policy.



Source-derived API Policy illustration showing read-only review. The displayed status is an example, not a captured or verified security state.

Source baseline: ROKKS web source f333629b

Step-by-step

1. Open **Admin**, then **Access Inventory**. Confirm the page heading is **API Policy**.
2. Read the status card and both policy identities. If loading fails or a mismatch appears, retain the reported error and identities for investigation.
3. Click **Refresh** to reload the deployment's policy information.
4. Enter an action, path, task, or surface in **Filter API functions** to narrow the table. Clear the filter to restore all returned functions.
5. Select a function. Read its deployed grant clauses, required scope, and declared object ACL and SQL RLS obligations. Use **Close** to dismiss the detail.
6. If needed, select the appropriate download. Use **Download openapi** for the public API contract; treat the catalog and policy bundle according to their intended audience.
7. Request a reviewed repository change and deployment when policy needs to change. There is no policy-authoring workflow on this page.

Common mistakes

Do not treat a displayed grant clause as a promise that your next request will succeed. Arbiter evaluates the actual request and scope; object ACL and SQL RLS still constrain data access.

Do not treat the filter or selected function as an access simulation. They change which information you inspect, not your roles, capabilities, or effective permissions.

Do not try to repair an unavailable or mismatched policy through an in-app approval or activation. Review the deployment and its policy identity. Application settings, identity assignments, object ACL, and SQL RLS configuration have their own controls; this page does not replace them.

Do not share a catalog or policy bundle outside its intended audience. Use the public OpenAPI download when a public contract is sufficient.

Related

- Account Settings
- Service Monitoring
- Alerts

Account Settings

Overview

Account Settings currently combines one active configuration area with several preview controls. Use **Dashboard Grid** for shared layout defaults. Treat the **Organisation** and **Team** policy panels as disabled, non-enforcing previews in this documented build; personal display choices belong under your user preferences.

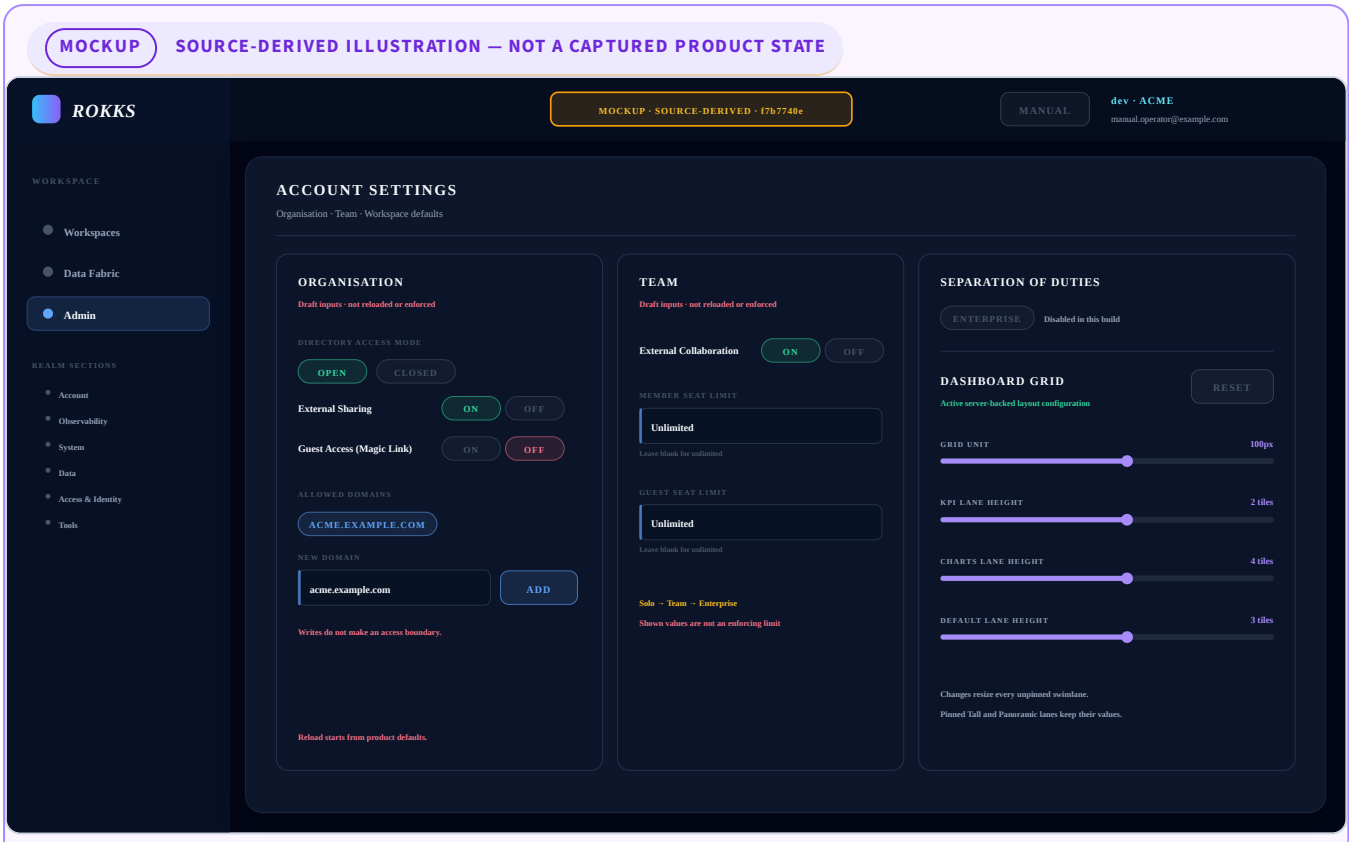
> **Important — Organisation and Team are not an access boundary in this build.** Their controls are disabled and do not update, store, reload, or enforce any policy. Authentication, sharing, guest admission, and seat limits do not consume these tenant/team previews; in particular, per-tenant **Allowed Domains** enforcement is deferred. Do not rely on these panels to grant or deny access.

How it works

The page has four sections with different maturity:

- **Organisation** shows disabled previews for directory mode, external sharing, guest access, and allowed email domains. The page labels the card **Preview · Not Enforced**. Nothing on the card is applied, stored, or read back on reload.
- **Team** shows disabled previews for external collaboration and member and guest seat limits. No service counts seats or checks the displayed collaboration setting.
- **Separation of Duties** explains the Enterprise dedicated-security-officer capability. Its control is deliberately disabled and writes nothing in this build.
- **Dashboard Grid** is active configuration. It controls the grid unit, default lane heights, and XS through XXXL widget-width presets across the account. Its values use the server-backed settings path; swimlanes manually pinned to an explicit height keep that value.

There is no Save button. Organisation and Team controls cannot be changed and issue no policy write. Dashboard Grid changes apply immediately and are the operative settings on this page.



Source-derived ACME Account Settings mockup. In this documented build, Organisation and Team are disabled, non-enforcing previews; Dashboard Grid is the active configuration area.

Source baseline: beta f7b7740e

Step-by-step

1. Open **Admin**, then **Account Settings**.
2. Inspect **Organisation** or **Team** only to understand the proposed policy shape. Their controls remain disabled, so do not use the displayed values to make an access, sharing, guest-admission, or licensing decision.
3. Manage current admission and membership through the Auth IAM directory. **Allowed Domains**, **External Collaboration**, and the seat-limit fields on this page are explanatory previews only.
4. In **Dashboard Grid**, adjust the grid unit and lane-height sliders. Review the effective pixel sizes shown at the bottom of the section.
5. Adjust the XS–XXXL width presets only when the whole account needs different widget widths.
6. Use **Reset** only when you intend to restore all Dashboard Grid sliders to the product defaults.

Common mistakes

Do not interpret the absence of a Save button as proof that every panel applies immediately. Organisation and Team are disabled previews in this build; Dashboard Grid is active configuration.

Do not treat **Open**, **Closed**, **External Sharing**, **Guest Access**, **Allowed Domains**, **External Collaboration**, or seat limits as enforced policy. Their displayed values are fixed preview content, not a policy authority.

Do not interpret the **Allowed Domains** preview as authentication behaviour. Per-tenant domain enforcement is deferred.

Do not click **Reset** as a preview. It immediately replaces the current Dashboard Grid values and does not ask for confirmation.

Do not treat the disabled **Dedicated Security Officer** pills as a configurable switch in this build. The card is an explanatory Enterprise capability only.

Related

- Access Inventory
- Preferences
- Styles
- Workspaces

Companies and Teams

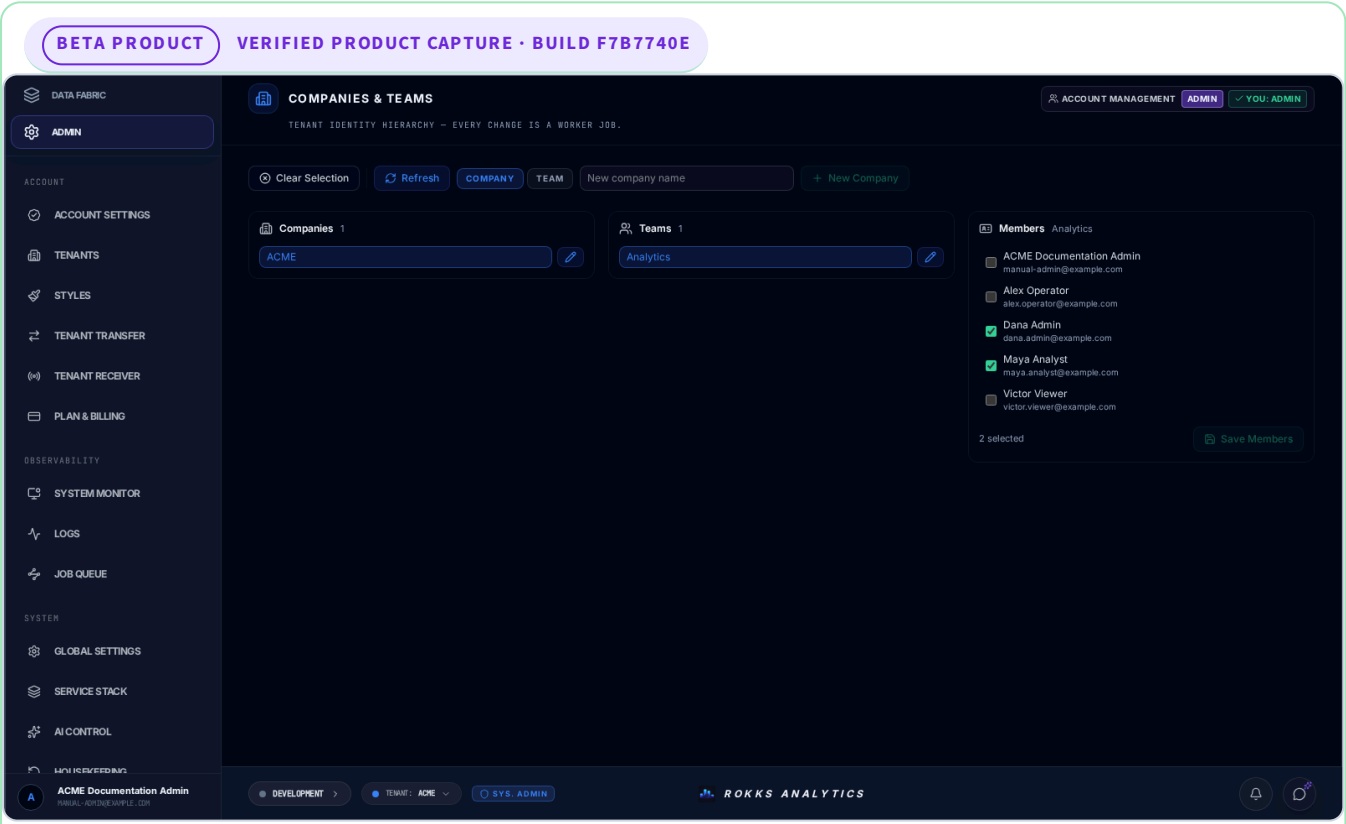
Overview

Open **Admin** → **Companies & Teams** to maintain the identity hierarchy inside the selected tenant. Use this page to group people into companies and teams; use an object's **Permissions** editor to grant that group access to a workspace or other resource.

How it works

The page has three columns: **Companies**, **Teams**, and **Members**. A company belongs to the selected tenant. A team belongs to exactly one company. Selecting a company shows its teams and its own members; selecting a team changes the Members column to that team.

Creating, renaming, and saving memberships run as background jobs. Wait for the operation to finish and the refreshed result to appear before making the next change. Switching tenants clears the previous selection. Without a selected tenant, the page asks you to choose one and does not show a hierarchy.



Real beta Companies and Teams page: ACME, its Analytics team, and saved memberships for Dana Admin and Maya Analyst.

Captured 12 Sept 2026 UTC [Open documented view ↗]

The real capture shows the **ACME** company, its **Analytics** team, and two saved memberships: **Dana Admin** and **Maya Analyst**. The remaining demo users are not members of this team.

Step-by-step

1. Select the intended environment and tenant, such as **ACME**.
2. Open **Admin** → **Companies & Teams**. If access is denied, ask the tenant administrator to check your assigned role and administration capabilities.
3. Choose **Company**, enter **ACME**, and click **NEW COMPANY**.
4. Select the new company. Choose **Team**, enter **Analytics**, and click **NEW TEAM**. A team cannot be created until its parent company is selected.
5. Select either the company or the team. In **Members**, check the users who should belong to that exact organization, then click **Save Members**.
6. Confirm the saved membership after the operation completes. Repeat for the other organization if it needs a different member set.
7. To rename a company or team, click its pencil control, edit the name, and save it. Cancel or press Escape to discard the edit.

Clear Selection only clears the current selection. **Refresh** reloads the hierarchy and directory; it does not create or delete anything. The documented page has no company or team deletion control.

Common mistakes

In the documented beta build, an identity job can complete while this page remains busy. Check the operation in **Job History**. After confirming **DONE**, reload the page and verify the saved company, team, or members. Do not submit the same creation again just because its screen has not refreshed.

Do not assume that adding somebody to a company automatically selects them in every team. Edit and verify the intended member set.

Do not treat an unavailable directory as an empty organization. A failed read is shown as an error, and an unresolved member set cannot be saved. If an enrolled member is no longer listed in the directory, the page identifies that situation and preserves the enrollment when you save; it does not silently remove the person.

Membership is not itself an object-access grant. A company or team can be selected as a subject in **Permissions**, but creating it does not make every workspace available to its members. Recheck access as the affected user after changing membership or assignments.

Related

- Tenants
- Workspaces
- Access Inventory
- Job Queue

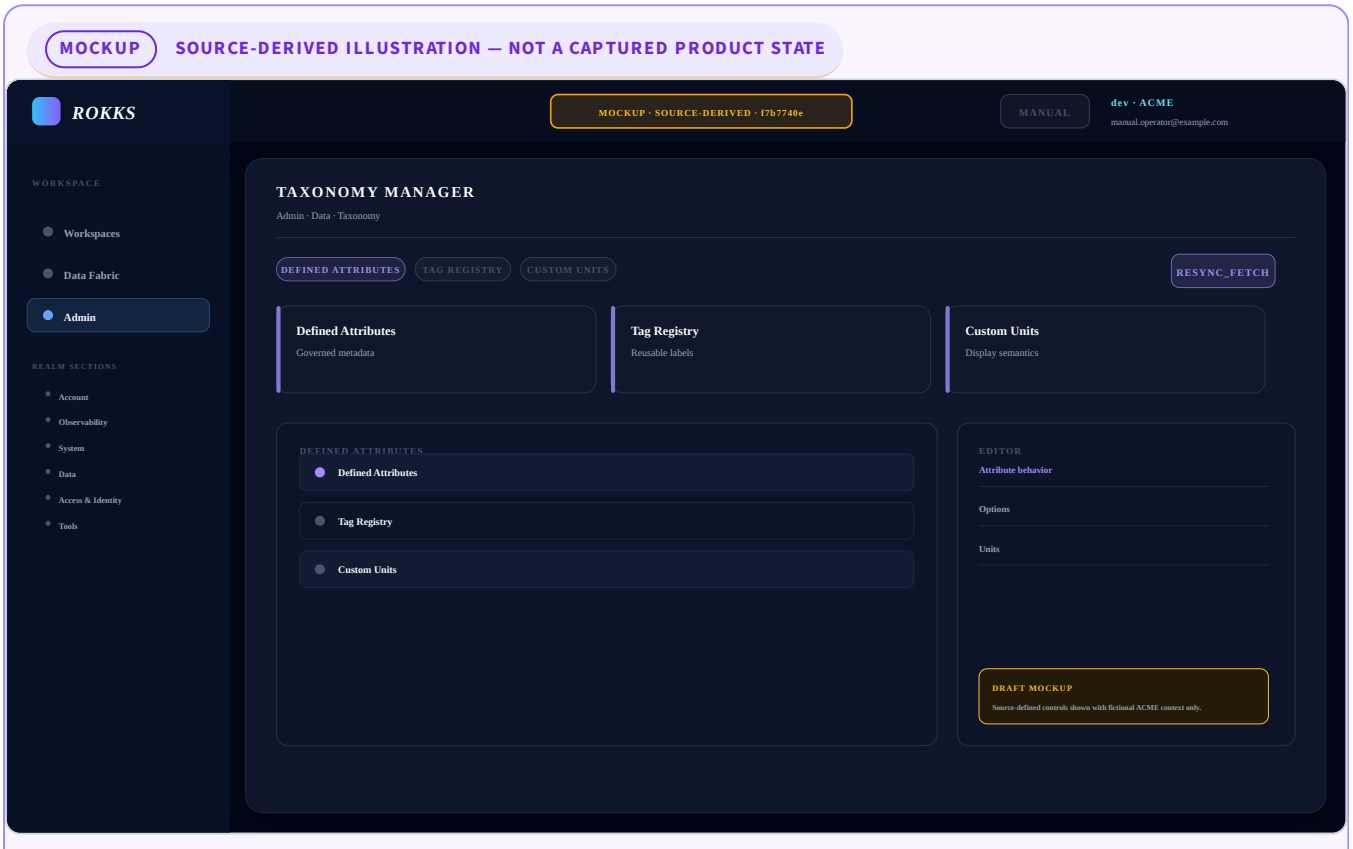
Taxonomy

Overview

Taxonomy gives shared meaning to fields, dashboards, and business concepts. Use it to standardize tags, attributes, units, and categorization across teams.

How it works

Taxonomy Manager maintains the shared definitions that other surfaces can use. Its three tabs are **Defined Attributes**, **Tag Registry**, and **Custom Units**. Assignments are made from consuming surfaces such as File Areas, Web Data Interfaces, or Local Tables rather than from Taxonomy Manager itself.



Taxonomy Manager illustration showing its source-owned registries and controls. Source baseline: beta f7b7740e

Step-by-step

1. Open **Taxonomy**.
2. Select **Defined Attributes**, **Tag Registry**, or **Custom Units**.
3. Use **New Attribute**, **Create Tag**, or **Add Custom Unit** for a reusable definition.
4. Enter the required name, options, symbol, or category. Attribute and tag changes persist immediately; for a custom unit, choose **Save Unit**.
5. Open the relevant File Area, Web Data Interface, or Local Table to map tags and attributes to that resource.
6. Choose **Resync_Fetch** when you need to reload the registry state.

Common mistakes

Do not create duplicate tags with slightly different spelling. Consolidate terms before adding new ones.

Do not use taxonomy for one-off notes. Use descriptions or dashboard text when the concept is not reusable.

Related

- Schema Editor
- Styles
- Glossary

Preferences

Overview

The profile menu contains two different self-service pages:

- **User Settings** shows the identity and access facts in your signed session and lets you manage your passkeys, personal API tokens, and Sentinel notification channels.
- **Preferences** controls where ROKKS opens after sign-in, which environment it selects, how AI and the Widget Editor engage, and how numbers and dates appear.

These pages affect your own account and experience. They do not grant roles, add group membership, or change a shared dashboard for other people.

How it works

User Settings

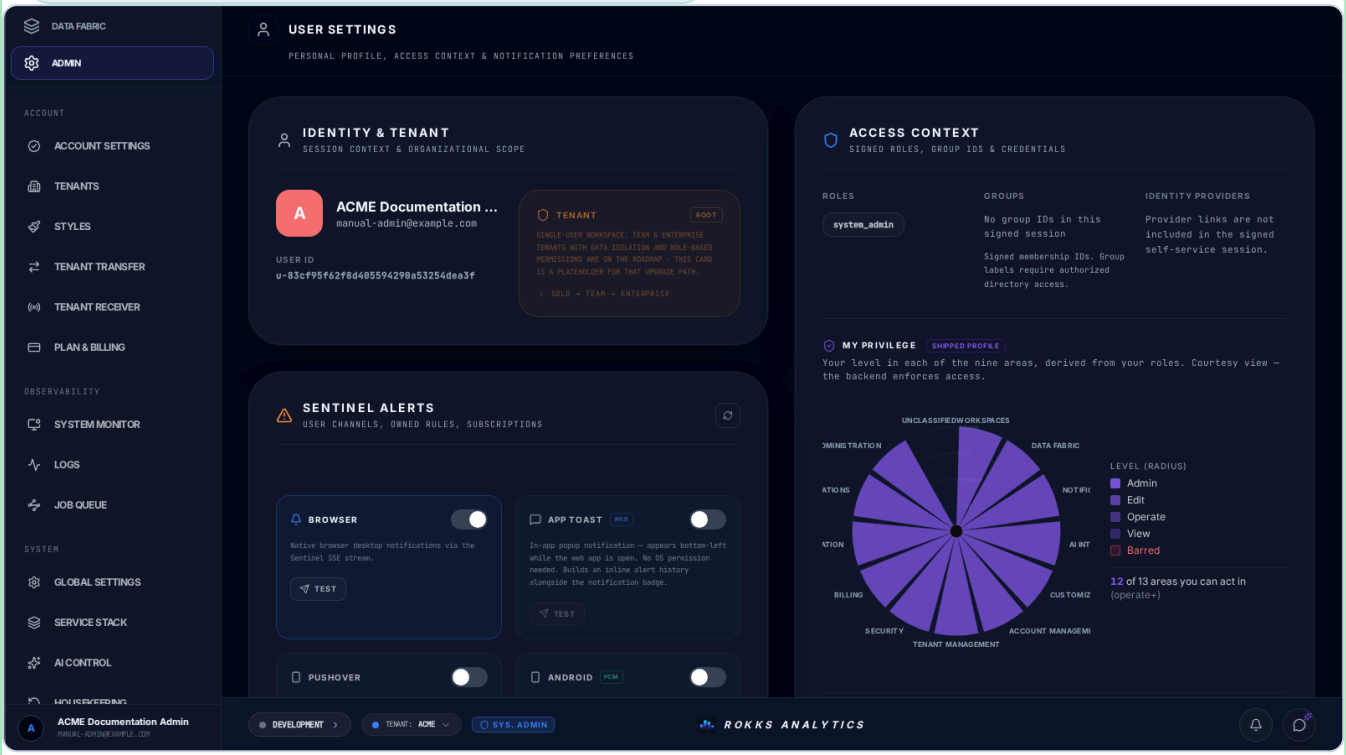
Open **User Settings** to review the signed identity attached to the current session. **Access Context** lists every signed role and exact group ID, followed by **My Privilege**, passkeys, and personal API tokens. A group ID remains visible even when you are not allowed to open the administrator directory that resolves its display label. Linked identity-provider details are not included in this self-service session; the page says that explicitly instead of claiming that no providers exist.

Use **Passkeys** to add a phishing-resistant WebAuthn sign-in credential through the browser or operating-system prompt. ROKKS shows the Passkeys block only when the browser reports an available platform authenticator. The **Device name** field is a label for the credential; the authenticator decides whether the passkey stays local or synchronizes, so ROKKS does not promise that it is device-bound. Use **API Tokens** for command-line tools, scripts, and CI. A newly generated token is shown in plaintext once; copy it before leaving the amber reveal panel. The active-token list later shows only its prefix, last four characters, creation date, and last-used date. Revocation requires a second click on **Confirm?**.

The **Sentinel Alerts** card manages four personal delivery channels:

- **Browser** sends an operating-system notification after browser permission is granted.
- **App Toast** displays an in-app message.
- **Pushover** needs both a user key and app token before it can be tested.
- **Android** uses Firebase Cloud Messaging after the Android app has registered the device.

Each enabled channel has a **Test** action and reports its diagnostic steps inline. The same card lists alert rules you own under **My Alerts** and rules you follow under **Subscribed Alerts**.



Reviewed ACME User Settings capture showing the demo identity, signed access context, and notification-channel controls.

Captured 12 Sept 2026 UTC [Open documented view ↗]

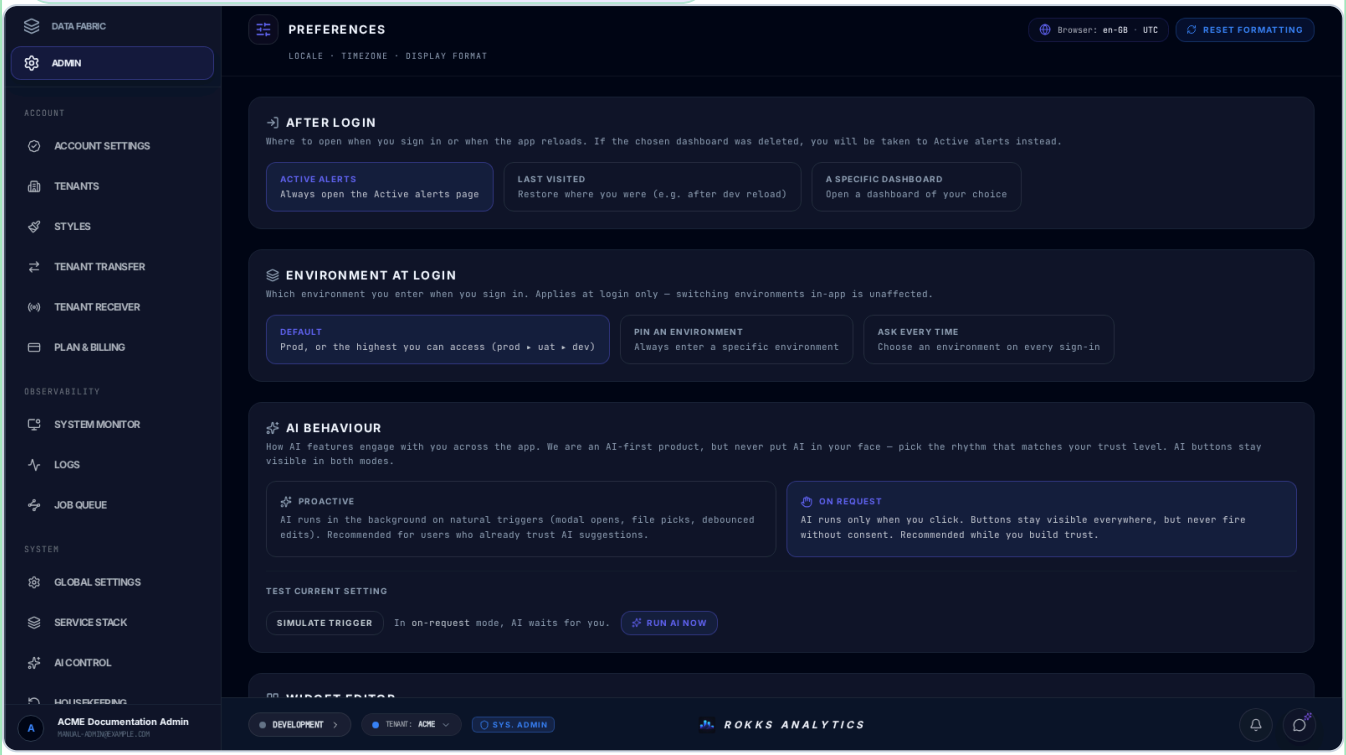
Preferences

Open **Preferences** to configure:

- **After login:** Active alerts, Last visited, or a specific dashboard. If the selected dashboard no longer exists, ROKKS opens Active alerts.
- **Environment at login: Default** follows the sign-in authorization priority—prod, then uat, then dev—among environments that are enabled and permitted for your account. If none of those three is permitted, it uses the first other permitted environment. This resolver does not elect or store a default environment. You can instead pin an environment or choose **Ask every time**. The choice applies at sign-in; it does not prevent you from switching environments afterwards.
- **AI behaviour:** Proactive permits automatic AI work at natural triggers; On request waits for an explicit click. AI actions remain visible in both modes. **Simulate trigger** demonstrates the current choice without leaving the page.
- **Widget editor:** Auto-run on open immediately runs the saved query to populate its preview and field assignments; Manual run waits for you to select **Run**.
- **Number & Date Format, Date Format, Timezone, and Clock format:** choose explicit values or the browser-derived automatic values. **Live Preview** shows the effective result before you leave the page.

Changes apply immediately; there is no **Save** button. **Reset formatting** restores only the locale, timezone, date format, and clock format to their product defaults. It preserves the post-login destination, AI behaviour, Widget Editor and Workbench behaviour, Sidekick choices, and other non-formatting preferences. It also leaves the separately stored **Environment at login** choice unchanged. The control is unavailable without a selected tenant and shows **Resetting formatting…** while the save is in progress.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Real preferences for the ACME documentation account: Active Alerts after login, the default environment choice, and AI behaviour set to On Request.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Step-by-step

Review access and credentials

1. Open the profile menu, then **User Settings**.
2. Confirm that the name, email, user ID, roles, and group IDs match the account you intended to use.
3. Read the **My Privilege** summary before attempting an administrative task.
4. To add a passkey, enter a device name and select **Add Passkey**. Complete the browser or operating-system prompt.
5. To create a personal API token, select **Generate Token**, enter a descriptive name, then select **Create**. Copy the plaintext token immediately and store it in the approved secret store; ROKKS will not show it again.
6. To retire a token, select **Revoke**, then **Confirm?**.
7. Enable only the notification channels you intend to use. Supply the required Pushover values or open the Android app once when those channels apply, then select **Test** and read every diagnostic step.

Choose personal behaviour and formatting

1. Open the profile menu, then **Preferences**.
2. Choose the **After login** destination. Select an existing dashboard if you choose the dashboard option.
3. Choose the **Environment at login** behavior. When pinning, select one of your available environments.
4. Select **Proactive** or **On request** under **AI behaviour**, then use **Simulate trigger** to verify the difference.
5. Choose whether the **Widget editor** should auto-run saved queries. Prefer Manual run for slow or expensive queries.

6. Pick a locale, date format, timezone, and clock format. Check **Live Preview** for large and small numbers, percentages, dates, and timestamps.
7. Return to the affected workflow and confirm the result. There is no separate save step.

Common mistakes

Do not use User Settings as an administrator directory. It describes the authority in your signed session; it does not resolve group labels or linked identity-provider records.

Do not navigate away before copying a newly generated API token. The plaintext is deliberately shown only once.

Do not enable Pushover without both required values, and do not expect Android tests to work before the app registers the device.

Do not confuse assistant engagement with feature access. On-request mode can still show AI buttons; it only changes when AI runs automatically.

Do not pin an environment or dashboard that you may lose access to without expecting fallback behavior. An unavailable pinned environment falls back through **Default** mode's permitted-environment priority; a deleted post-login dashboard falls back to Active alerts.

Do not expect personal preferences to fix shared dashboard design. If every user needs a change, edit the dashboard or the appropriate platform setting.

Do not use **Reset formatting** as a preview or as a reset of all personal settings. It saves a reset of those four formatting fields; it does not change AI behaviour or your landing-page choice. Wait for the save to finish before making another change.

Related

- Sidekick
- AI Control
- Navigation
- Styles

Advanced Platform Tools

Overview

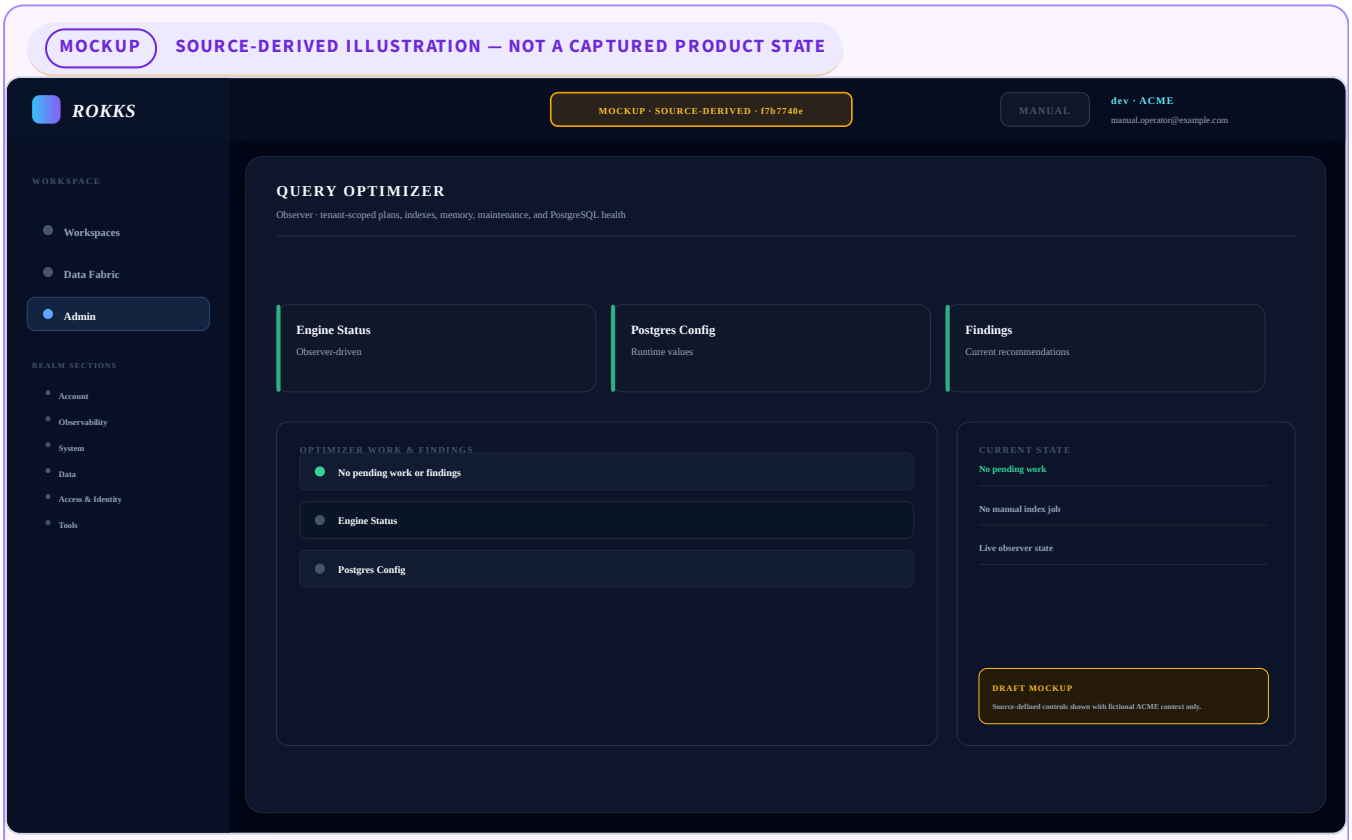
These restricted administration pages expose platform-wide inspection and configuration controls. They may be absent from your navigation because of your role or deployment edition. Use them only within your organization's operating procedure; ordinary dashboard and data work does not require them.

How it works

Each subsection below corresponds to a separate page in the shipped navigation. The illustrations show source-defined controls or neutral empty states; they do not imply that a configured optimizer, identity provider, container host, or registry record was available during capture.

Query Optimizer

Open **Admin** → **Data** → **Query Optimizer** to inspect **Engine Status**, **Postgres Config**, and current optimizer work or findings. Use the **Refresh all** icon to reload the displayed state. The empty state reads **No pending work or findings**.

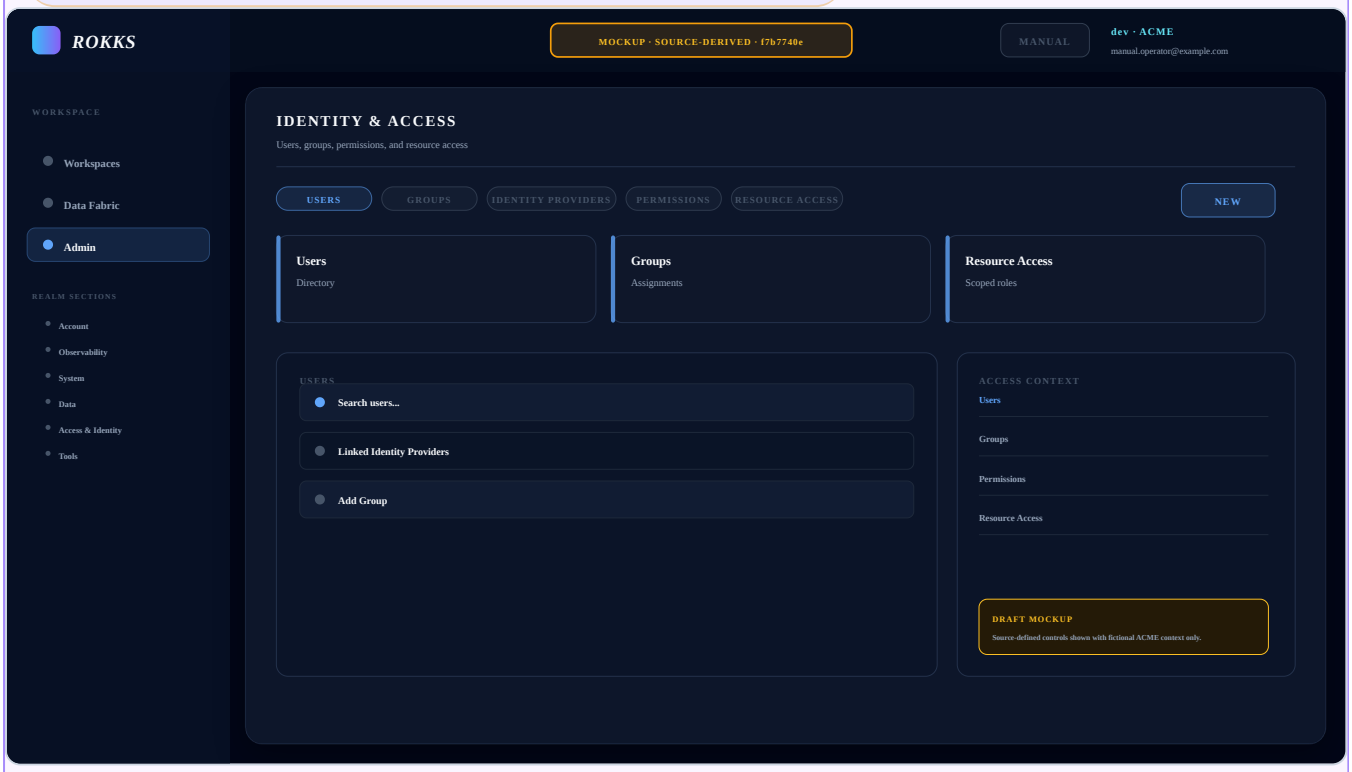


Query Optimizer illustration showing engine state and the source-defined empty findings state. Source baseline: beta f7b7740e

Identity and Access

Open **Admin → Access & Identity → Identity & Access** to work with the separate **Users**, **Groups**, **Identity Providers**, **Permissions**, and **Resource Access** areas. Open **Admin → Access & Identity → Identity Providers** for provider configuration; that page uses **Add Identity Provider** when an environment has no configured provider.

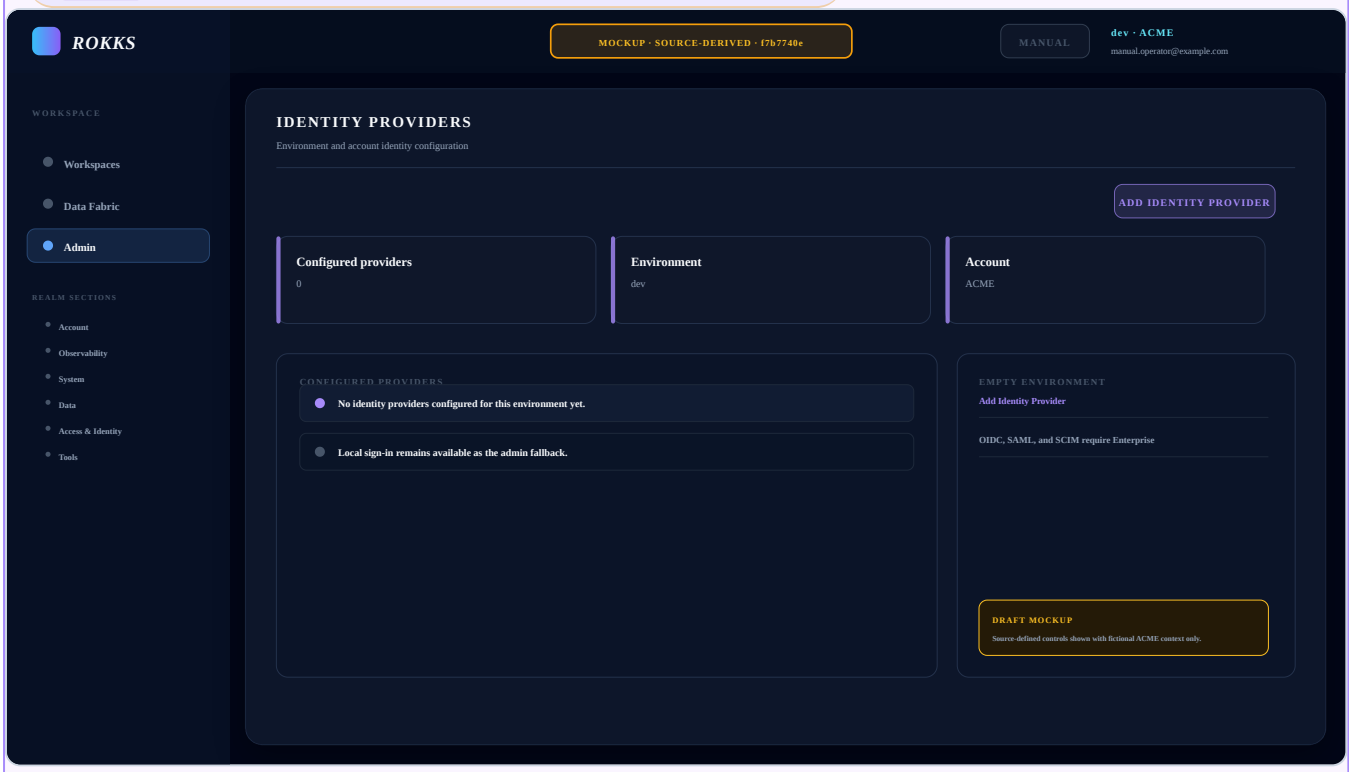
MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Identity and Access illustration using fictional ACME and example.com identities only.

Source baseline: beta f7b7740e

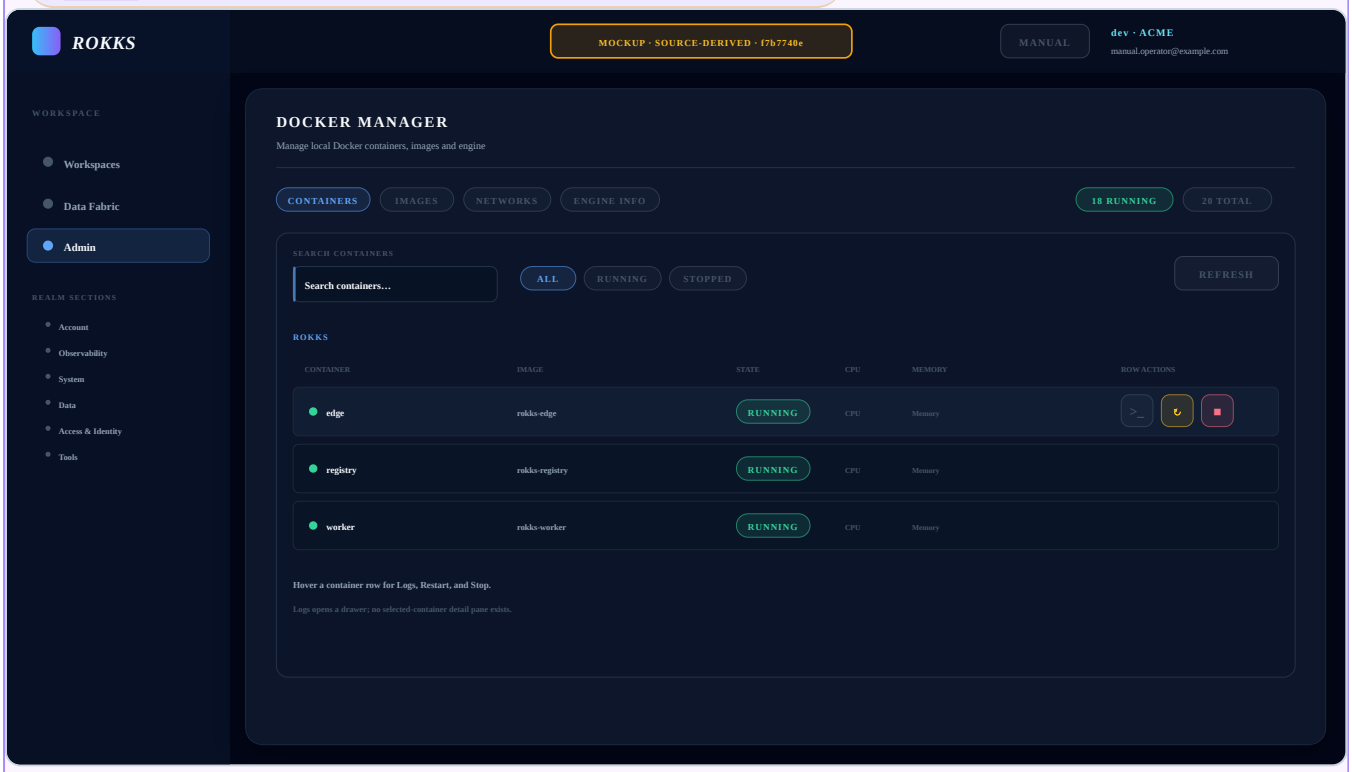
MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Identity Providers illustration showing one coherent source-defined empty environment state. Source baseline: beta f7b7740e

Docker Manager

On appliance deployments, open **Admin** → **System** → **Docker Manager** to inspect the **Containers**, **Images**, and **Networks** views. Container actions such as **Start**, **Restart**, and **Stop** affect runtime services; follow the operator procedure for the target environment before using them.

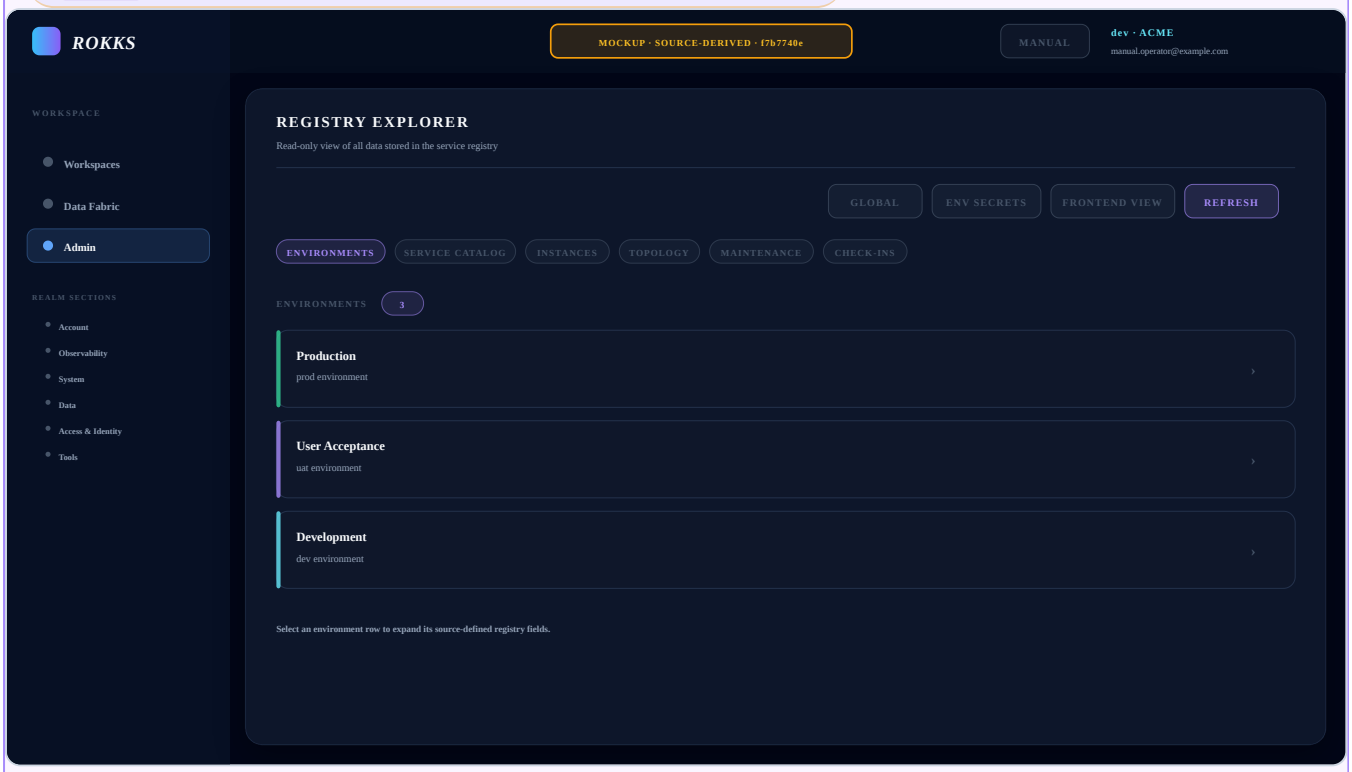


Docker Manager illustration showing Containers, Images, Networks, and Engine Info tabs. Source baseline: beta f7b7740e

Registry Explorer

Open **Admin** → **Tools** → **Registry Explorer** to inspect the registry's **Environments**, **Service Catalog**, **Instances**, **Topology**, **Maintenance**, and **Check-ins** projections. Access to protected environment details depends on your role; do not copy secrets or real organization data into support material.

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Registry Explorer illustration showing registry projections and topology tabs. Source baseline: beta f7b7740e

Common mistakes

- Do not treat an empty optimizer findings list as proof that every query is optimal; it reports the current observed state.
- Do not assume Identity Providers and Resource Access are the same control. Authentication configuration and resource authorization are separate.
- Do not use Docker Manager as a substitute for the deployment runbook.
- Do not publish Registry Explorer output. Manual examples must remain ACME and `example.com` only.

Related

- Access Inventory
- Authentication
- Troubleshooting
- Deployment Topologies

SECTION 7

Mobile

4 topics

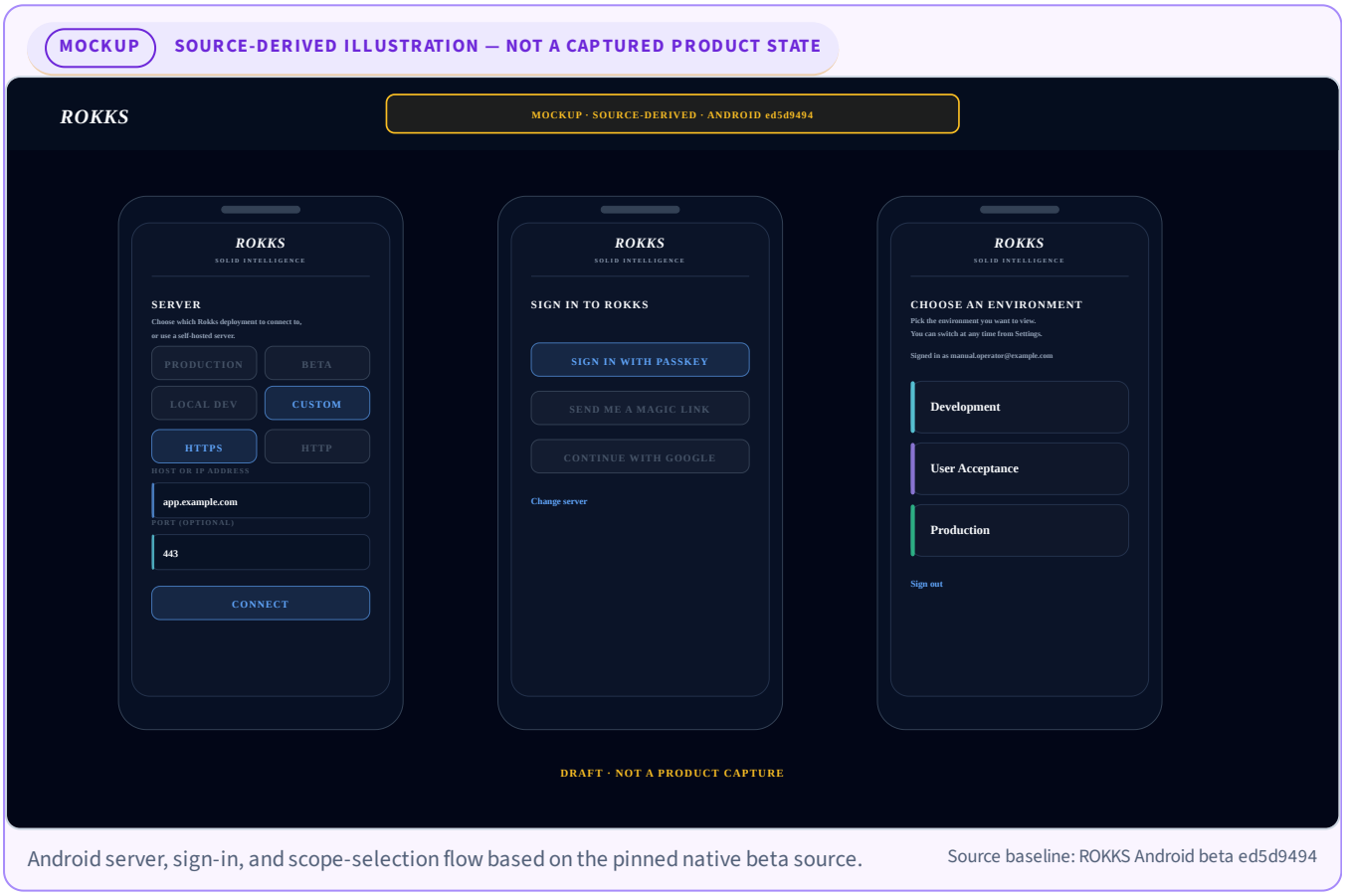
Android App

Overview

The Rokks Android app is a native mobile companion for users who need to check dashboards, alerts, and workspace state away from the desktop. It is designed as a focused viewer, not a full administration or dashboard-authoring client.

How it works

On first launch, the app asks for the Rokks server URL supplied by your organization and verifies it before saving. After the server is confirmed, you sign in, choose an allowed environment and tenant, and land in the mobile shell. The shell has three tabs: **Workspaces**, **Alerts**, and **Settings**. Workspaces lead to their dashboards and reports.



Android server, sign-in, and scope-selection flow based on the pinned native beta source.

Source baseline: ROKKS Android beta ed5d9494

The mobile app uses native Android screens. It is not a web view. It reaches the public Rokks application over HTTPS as a signed-in user and is intentionally read-only for product data. The illustration and this section are pinned to Android beta source `ed5d9494be9e5d3919d0c34d9d45440457e96ca3`, separate from the web build marker.

Step-by-step

1. Install the app from your organization’s approved distribution channel.

2. Enter the Rokks application URL provided by your administrator.
3. Sign in with one of the methods enabled for your server.
4. Select the environment and tenant you intend to view.
5. Open **Workspaces**, select a workspace, then select a dashboard or report.
6. Open **Alerts** to review notification events.
7. Open **Settings** to change server, tenant, or sign-in state.

Common mistakes

Do not expect the Android app to replace the desktop experience. Create data sources, edit dashboards, manage schemas, and perform administration from the web app.

Do not troubleshoot missing data without checking environment and tenant scope. The mobile app follows the same isolation model as the web app.

Related

- Mobile Dashboards
- Mobile Alerts
- Android Client Integration

Mobile Dashboards

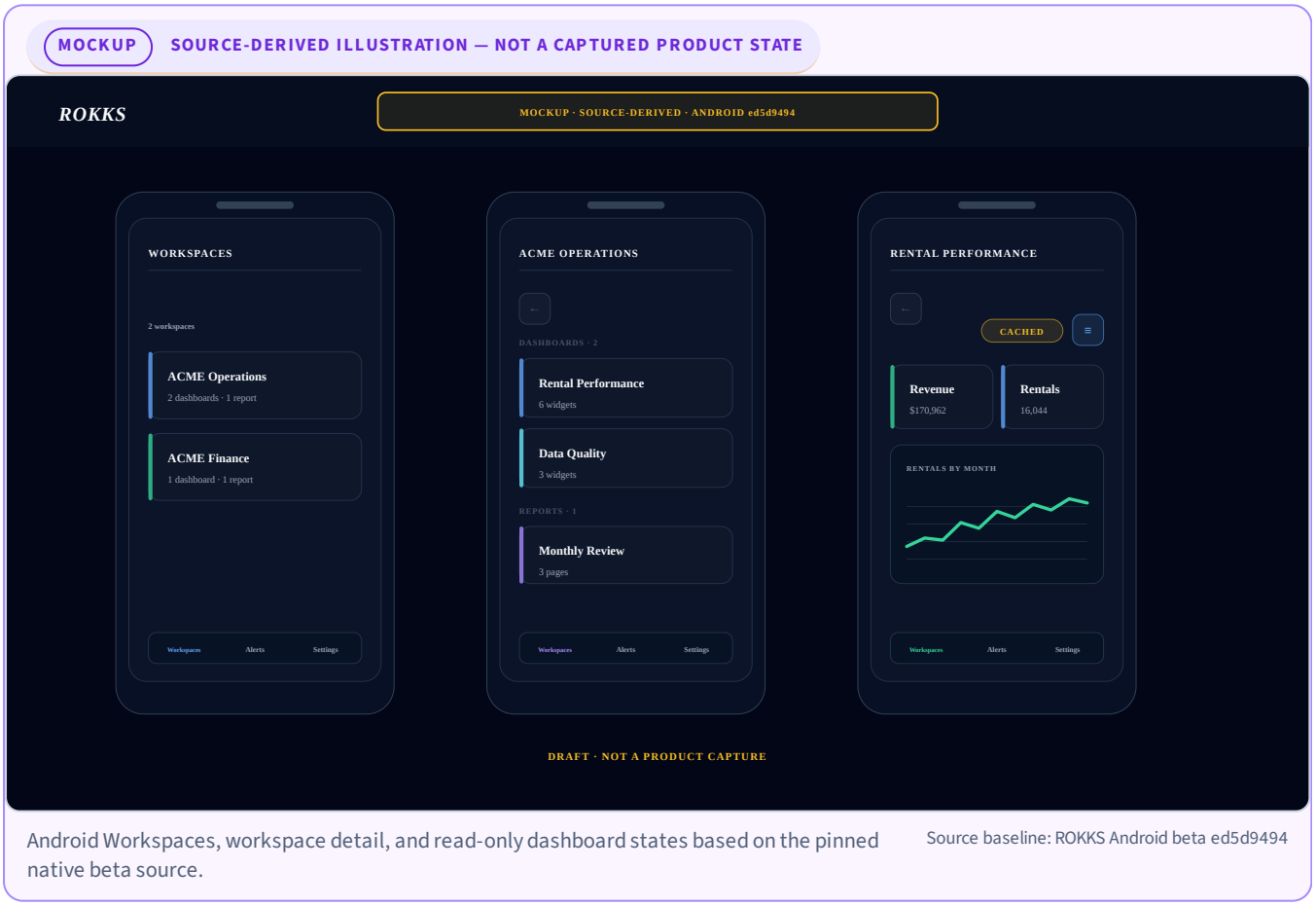
Overview

Mobile dashboards let you inspect operational information without opening the full desktop app. Use them for monitoring, review, and lightweight investigation.

How it works

The **Workspaces** tab lists workspaces available in the selected environment and tenant. A workspace opens its **Dashboards** and **Reports** sections. Opening a dashboard fetches a rendered payload from the platform and displays supported widgets as native mobile cards. Tables, KPI cards, charts, gauges, maps, and ECharts-backed visuals can be represented when the dashboard data supports them.

The app can show cached dashboard content when a recent rendered dashboard exists and loading the live result fails. Cached content is marked **Cached** so users understand that it may be stale. Filter and time controls appear when the dashboard exposes them.



Android Workspaces, workspace detail, and read-only dashboard states based on the pinned native beta source.

Source baseline: ROKKS Android beta ed5d9494

Step-by-step

1. Open the Android app.
2. Confirm the active environment and tenant.

3. Open the **Workspaces** tab.
4. Select a workspace.
5. Under **Dashboards**, tap the dashboard you want to inspect.
6. Use available filters or time controls when the dashboard exposes them.
7. Return to the workspace with the back action; return again for the workspace list.

Common mistakes

Do not use cached mobile data as proof that a live source is healthy. If a value matters operationally, reopen it with connectivity and make sure the **Cached** indicator is absent.

Do not expect every desktop-only layout affordance to be available on a phone. The mobile view optimizes the same information for smaller screens.

Related

- Android App
- Dashboard Collections
- Widget Capabilities

Mobile Alerts

Overview

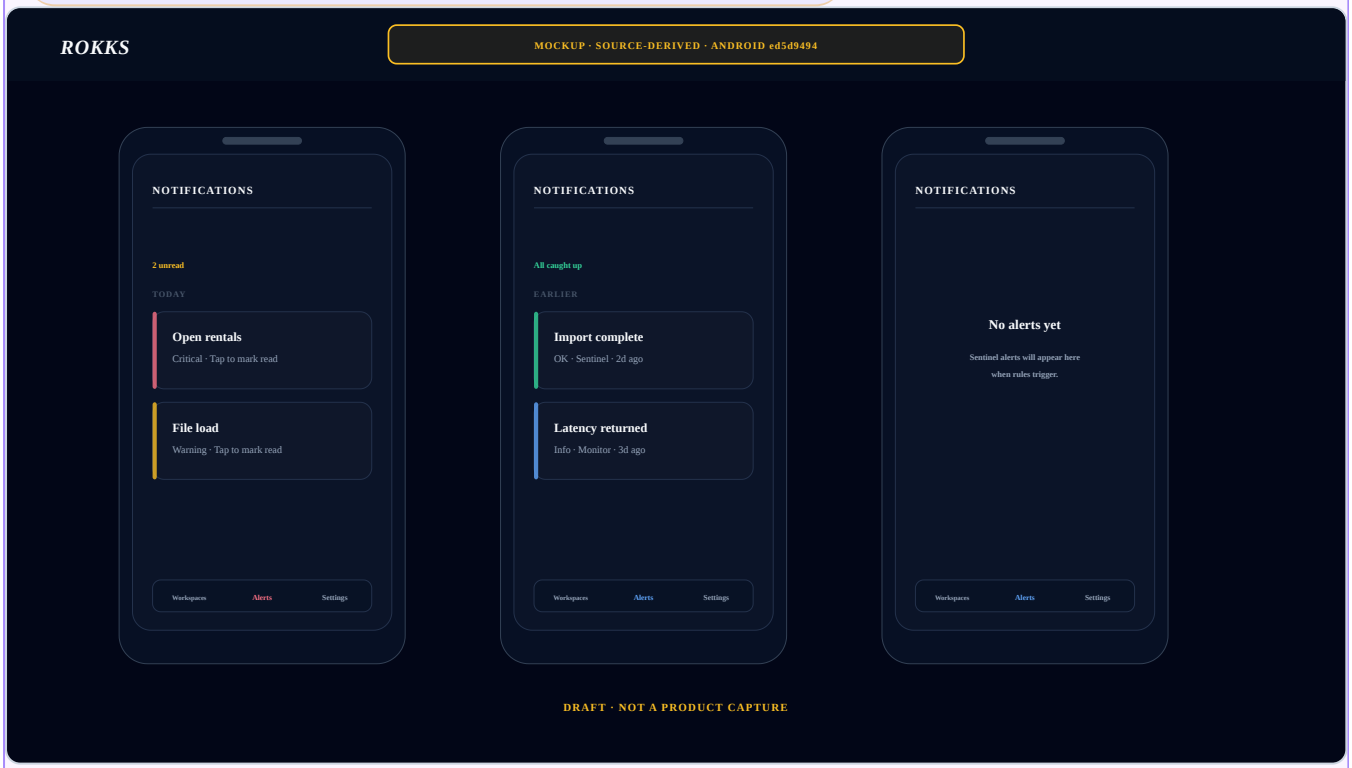
Mobile alerts help users notice and review important operational events while away from a desktop browser. The Alerts tab shows events the signed-in user can access in the active scope.

How it works

The **Alerts** tab opens the **Notifications** inbox. Events are grouped into **Today** and **Earlier** and show unread state, severity, source, and event time. Tapping an unread card marks it read; the current screen does not open a related dashboard from that card.

The beta contains device-token registration and native notification receiving code, but native delivery still depends on the deployment and Android notification permission. This build does not provide an in-app notification-permission prompt, so treat the Notifications inbox as the dependable review surface rather than assuming push delivery is active.

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Android Notifications inbox states based on the pinned native beta source.

Source baseline: ROKKS Android beta ed5d9494

Step-by-step

1. Sign in to the Android app.
2. Open the **Alerts** tab.
3. Review the unread count and severity labels.

4. Read the event title, body, source, and relative event time.
5. Tap an unread card to mark it read.
6. Open the relevant dashboard separately from **Workspaces** when investigation is required.

Common mistakes

Do not assume a missing push notification means no alert fired. Check the **Alerts** tab and the relevant dashboard if you suspect a notification channel issue.

Do not disable native notifications if the phone is part of an operational response process.

Related

- Android App
- Alerts
- Monitoring and Logs

Mobile Settings

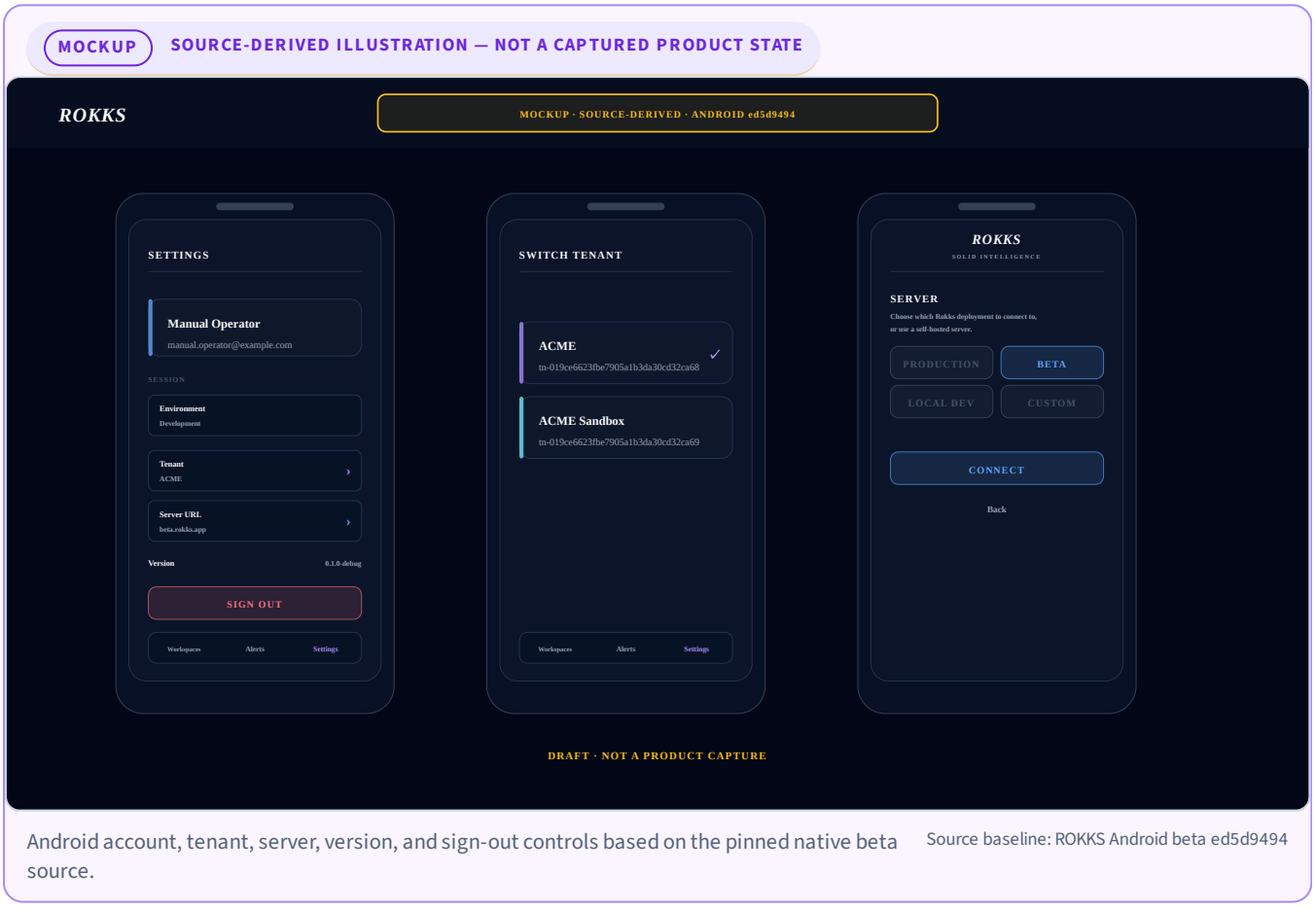
Overview

Mobile Settings controls the connection and account context for the Android app. Use it when you need to change server URL, switch tenant, review app information, or sign out.

How it works

The Android app stores the selected server URL separately from your token. Saving a different server URL clears the token and active scope, then requires a new sign-in. The current environment is shown for review; **Change** on the tenant row opens the tenant picker. The About section shows the app version.

Signing out clears the mobile token, active scope, dashboard-list cache, and rendered-dashboard cache on the device.



Android account, tenant, server, version, and sign-out controls based on the pinned native beta source.

Source baseline: ROKKS Android beta ed5d9494

Step-by-step

1. Open **Settings** from the bottom navigation.
2. Review the signed-in account, environment, and tenant.
3. Choose **Change** beside **Tenant** when you need another accessible tenant.

4. Choose **Edit** beside **Server URL** only when your administrator tells you to move to another Rokks deployment; changing it signs you out.
5. Review **Version** under About when support asks which app build you use.
6. Choose **Sign out** to clear the current mobile session and dashboard caches.

Common mistakes

Do not change the server URL to troubleshoot a missing dashboard. Check environment and tenant first.

Do not share a signed-in device. Mobile access follows the identity and permissions of the person who signed in.

Related

- Android App
- Environments
- Tenants

SECTION 8

Tutorials

5 topics

Pagila Maiden Flight

Goal

Complete one end-to-end ROKKS workflow with public fictional data: download the **Pagila Rental Constellation** JSON, load its six related tables through **File Areas**, verify the governed outputs, and create a relation-aware dashboard named **Pagila Rental Performance** in the **ACME Analytics** workspace.

This tutorial uses no customer data. The organization in examples is **ACME**, and any identity shown in an illustration uses the `example.com` domain.

Preflight check

Confirm this preflight checklist before uploading the file:

- You are signed in to the intended environment and the ACME tenant selected for this exercise.
- Your role can create File Areas and dashboards.
- File operations are available. If ROKKS shows a service-offline banner, stop and ask an administrator to restore the service.
- The configured maximum upload size is at least **14 MB**. The JSON is 14,316,401 bytes, so **15 MB** or more leaves useful margin.
- You have enough time to let the upload, sample, transform plan, and load jobs finish. Do not close a modal while it reports an in-progress save.
- You understand whether **Auto-Load Data** and **Proactive AI** are on. **Proactive AI** can advance Sample → Plan; **Auto-Load Data** can advance Plan → Load. You can still open the **Plan** bead to review the mapping.

The source upload is stored as a blob, not as document metadata. The platform's separately configured upload maximum is the relevant limit. If the upload is rejected as too large, ask an administrator to raise that configured maximum before continuing; do not split or truncate the canonical fixture.

Download the public Constellation fixture

Download the primary source, its derivation record, and the common licence:

- Pagila Rental Constellation JSON
- Constellation source and derivation record
- MIT License

The Constellation JSON is 14,316,401 bytes and has this SHA-256 checksum:

```
83281cc7936bf37a8f9a232ab1f468b4b4fe5664edb6a01d5b176dcb00c7ae4a
```

The **Pagila Rental Constellation** is a reduced six-table relational projection, not the complete upstream Pagila database schema. It contains `categories`, `stores`, `customers`, `films`, `rentals`, and `payments`. Fact coverage is complete for this projection: all 51,805 rental facts and all 51,056 non-empty rental-level aggregate

payment facts are present. A payment row represents the aggregate for one rental; it is not a copy of the upstream payment-table grain. The payment rows preserve the certified total revenue of 170,962.39 USD.

Deterministic derivation material

The JSON was generated deterministically from the companion Pagila rental facts CSV. The CSV source and derivation record pins the public Pagila v4.0.0 inputs, the transformation contract, and this SHA-256 checksum:

```
0e7a033d85e4e5657e2fefd5e81f68c7c067de92a43e3dae4b96ad75aef4a2ee
```

The CSV is 11,138,814 bytes and contains 51,805 rental rows with 21 flattened columns. It remains the reproducible input to the JSON generator and the public sample for the separate CSV to Dashboard tutorial. It is not a second maiden-flight import path. Both public fixture files were derived only from the pinned Pagila release and contain no ROKKS customer or vault data.

Steps

Step 1: Create the Constellation and upload the JSON

1. Open **Data Fabric**, then **File Areas**.
2. Click **New** in the File Areas rail.
3. Replace the generated name with **Pagila Samples**.
4. Open that area's action menu and choose **Add Constellation**. Name the child **Pagila Rental Constellation**.
5. Select the Constellation, choose **Upload Source**, and upload `pagila-rental-constellation.json`. A Constellation accepts one JSON source.
6. Wait for sampling to finish, then choose **Generate Plan**.

Step 2: Review and save the six-table plan

In **Define Transformation**, confirm that the overview contains exactly these six nodes and five relationships:

Node	Primary key	Expected rows after loading
`categories`	`category_name`	16
`stores`	`store_id`	2
`customers`	`customer_id`	997
`films`	`film_id`	958
`rentals`	`rental_id`	51,805
`payments`	`payment_id`	51,056

From	To
`films.category_name`	`categories.category_name`
`rentals.store_id`	`stores.store_id`
`rentals.customer_id`	`customers.customer_id`
`rentals.film_id`	`films.film_id`
`payments.rental_id`	`rentals.rental_id`

Check the field types, nullable dates, primary keys, and relationship endpoints. Do not run a plan with a missing node, an empty field list, a different key, or an incorrect edge. Choose **Save Constellation** when the plan matches the two tables above.

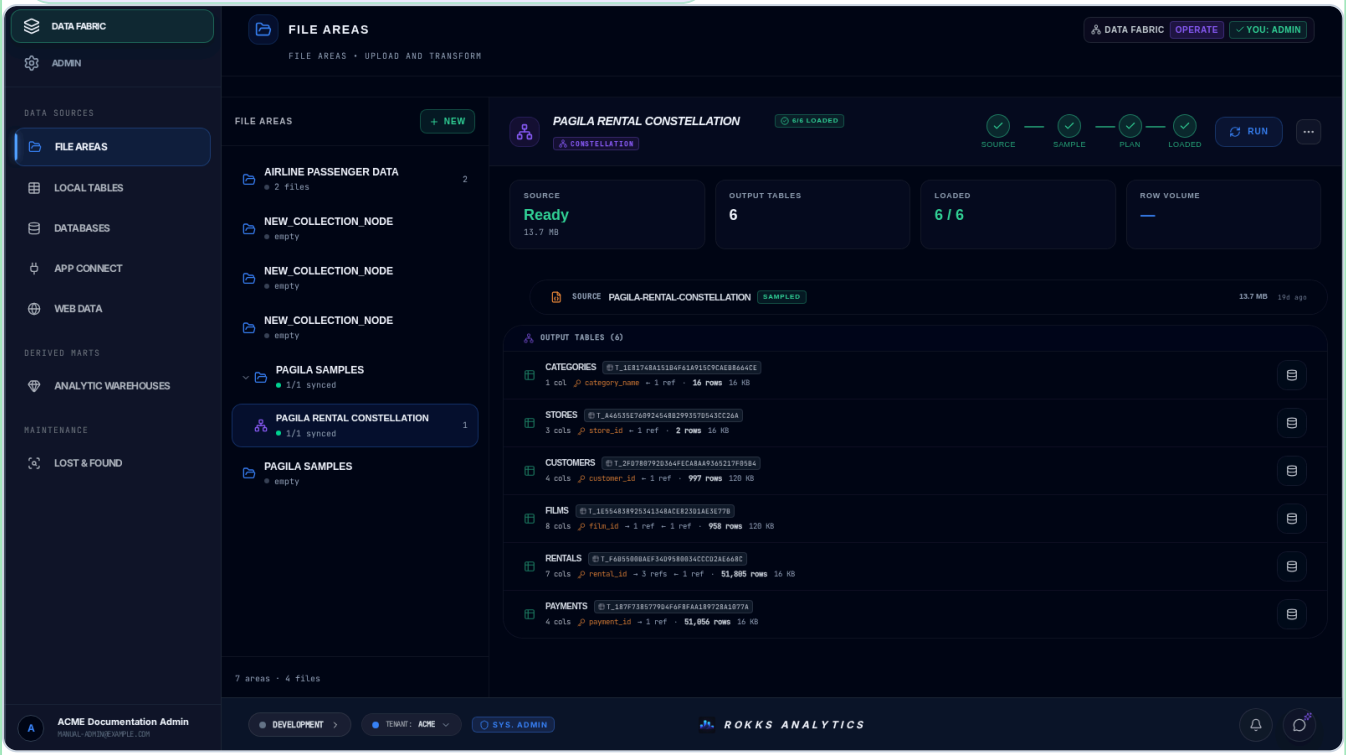
For the equivalent chat-driven workflow, follow the File Area and constellation cookbook. Keep the plan-review stop in the prompt so you can inspect the six nodes before starting the forward-only load.

Step 3: Run and verify every output

1. Select **Pagila Rental Constellation** and choose **Run**.
2. If your role exposes operational pages, open **Admin** → **Job Queue**, select **Job History**, and inspect the jobs for the Constellation. The `AREA_ETL_SYNC` parent dispatches the child `FILE_ETL` load. An `AREA_ETL_SYNC` parent reaching **DONE** proves dispatch, not the child load outcome. Require the child `FILE_ETL` job to reach **DONE**; **ERROR** or **CANCELLED** is not a successful maiden flight.
3. Return to File Areas and require the Constellation to report **6/6 loaded**.
4. In the **Output tables** list, confirm Categories 16, Stores 2, Customers 997, Films 958, Rentals 51,805, and Payments 51,056. Use each row's **Inspect** action to review the governed table.
5. Open **Analytic Warehouse** → **Data Source Tables** to find the same governed outputs, then open **Relationship Map** to review their connections. They do not appear under **Local Tables**.

The captured ACME constellation below passed this load checkpoint. All six table counts match the fixture; the child load job completed successfully. This is the reduced rental projection described above, not the complete upstream schema.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Real beta File Areas output: six loaded tables, including 51,805 rentals and 51,056 rental-level payment aggregates.

Captured 12 Sept 2026 UTC [Open documented view ↗]

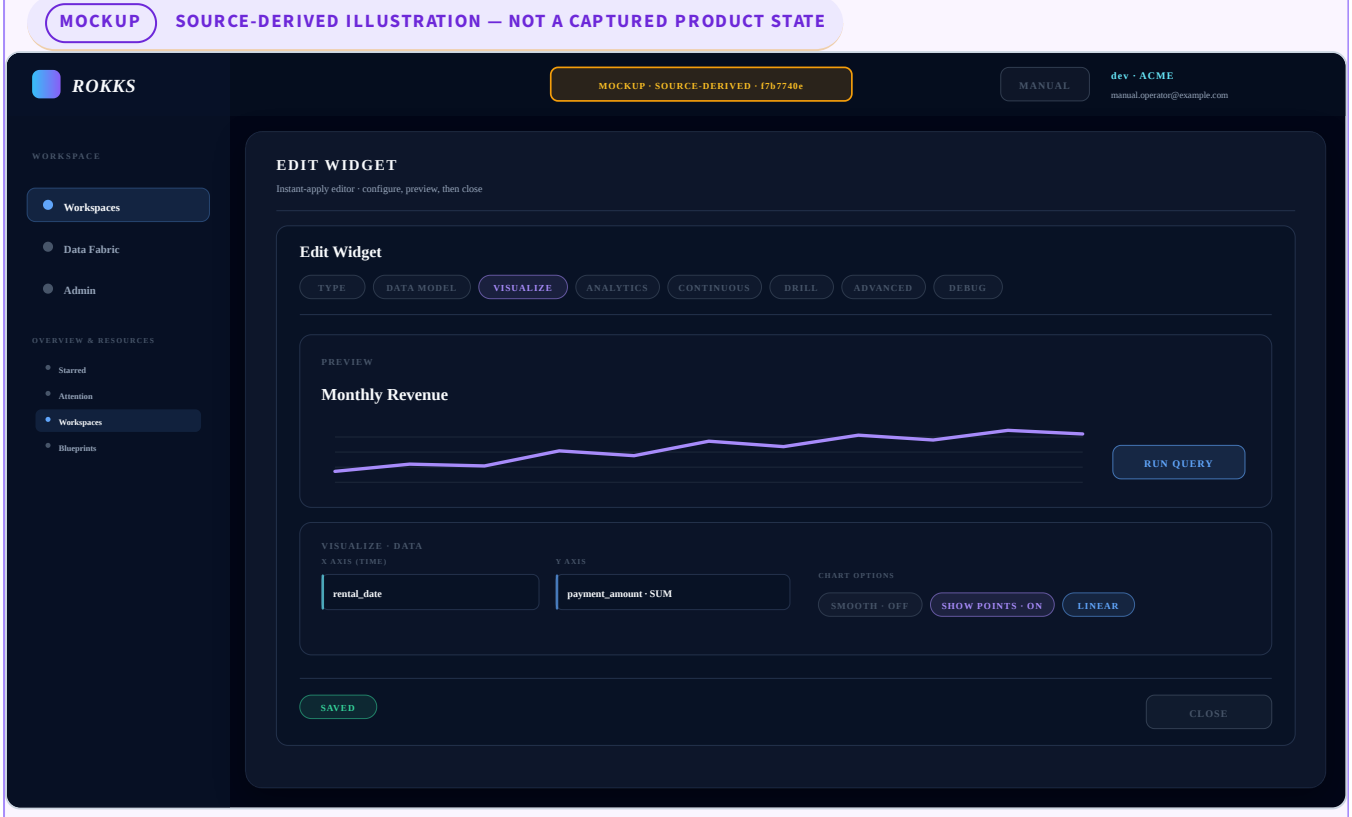
Step 4: Create the relation-aware dashboard

1. Open **Workspaces**, then enter or create **ACME Analytics**.
2. Click **New**. If the workspace is empty, the equivalent action is **Add dashboard**.
3. Hover the new dashboard title, click its pencil, and name it **Pagila Rental Performance**.
4. On the empty-dashboard screen, choose **Build it by hand**, then click **Start building**.
5. Click **Add column**. In the automatically created empty row, click **Add or drag widget here** to open **Edit Widget**.

Create the first widget as follows:

1. Open **Type** and choose **Line**.
2. Open **Data Model** and use the schema canvas. The canvas begins with **Select a field on any table to start**; there is no separate source dropdown. If it reports no data sources, add **Pagila Samples** under dashboard **Settings → Data Context**.
3. Select the qualified fields `"payments"."amount"` and `"payments"."payment_date"`; do not substitute similarly named unqualified fields.
4. Open **Visualize** and map `"payments"."payment_date"` to **X Axis (time)**, `"payments"."amount"` to **Y Axis 1**, and aggregation to **SUM**.
5. Select **Custom** in the title controls and enter **Monthly Revenue**.
6. Click **Run Query** and inspect **Preview**. Resolve any error, wait for the save state to settle, then click **Close**. The instant-apply editor adds the widget on its first successful save and recompiles it in the background.
7. In the dashboard toolbar's **Resolution** menu, select **1mo — 1 month**. In the documented build, known product issue **1374** can make this selection render an unnecessarily fine grain. If the chart becomes dense, return to **Auto** and continue; the source values and the other widgets remain usable.

MOCKUP — verified Pagila capture pending. This source-derived asset illustrates a Pagila-specific Widget Builder layout. Follow the qualified `"payments"."amount"` and `"payments"."payment_date"` mapping above. The real product editor is pictured in the Widget Builder guide.

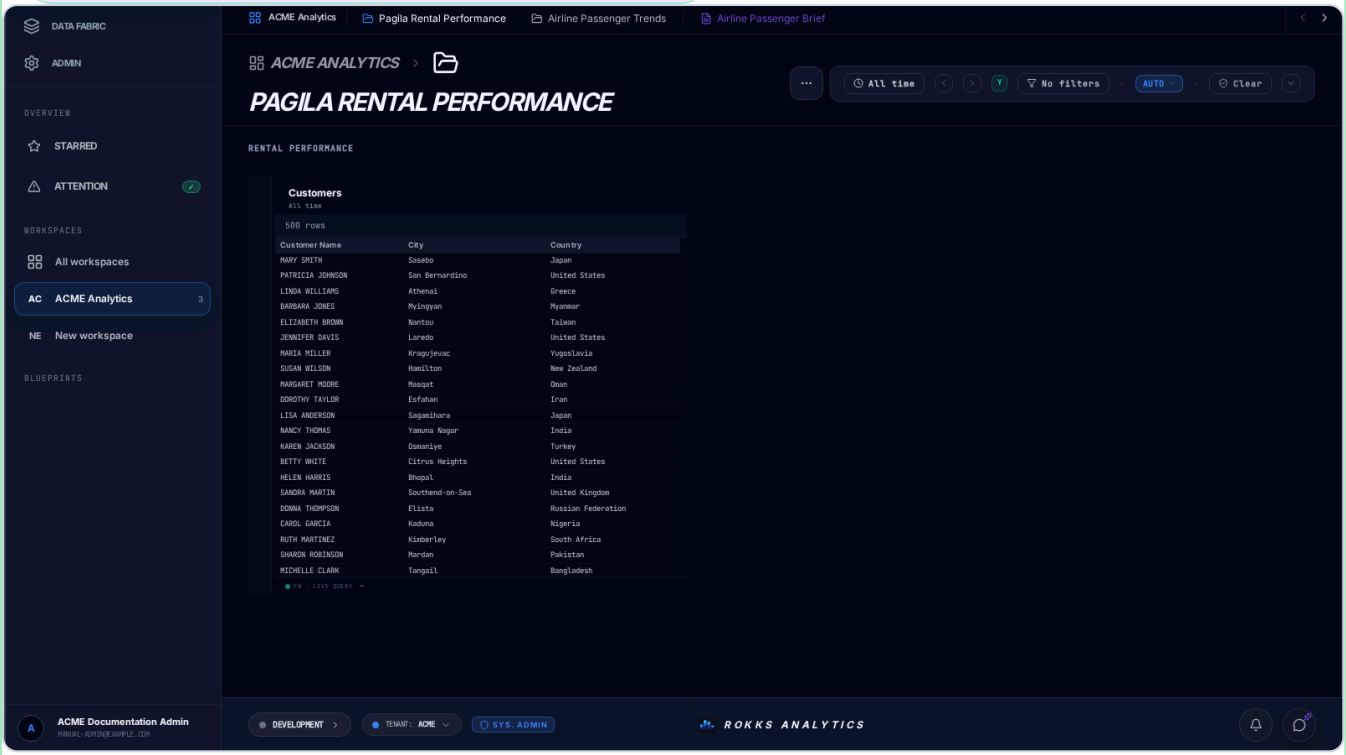


Widget Builder configuration based on the current catalog-backed editor. Source baseline: beta f7b7740e

Verified intermediate dashboard

Before building the aggregate widgets, you can prove that the dashboard can query one governed output table. Add a **Data Table**, anchor it on **Customers**, and add `customer_name`, `city`, and `country` to the presentation fields. The real beta capture below shows that checkpoint with a compiled live query. Its **500 rows** label is the widget preview limit; the loaded Customers output still contains the 997 rows verified in Step 3.

BETA PRODUCT VERIFIED PRODUCT CAPTURE · BUILD F7B7740E



Real beta Pagila dashboard with a compiled Customers table showing 500 public sample rows by name, city, and country.

Captured 12 Sept 2026 UTC [Open documented view ↗]

Return to **Edit Layout** when the table renders, then continue with the aggregate starter dashboard below.

Build this eight-widget starter dashboard:

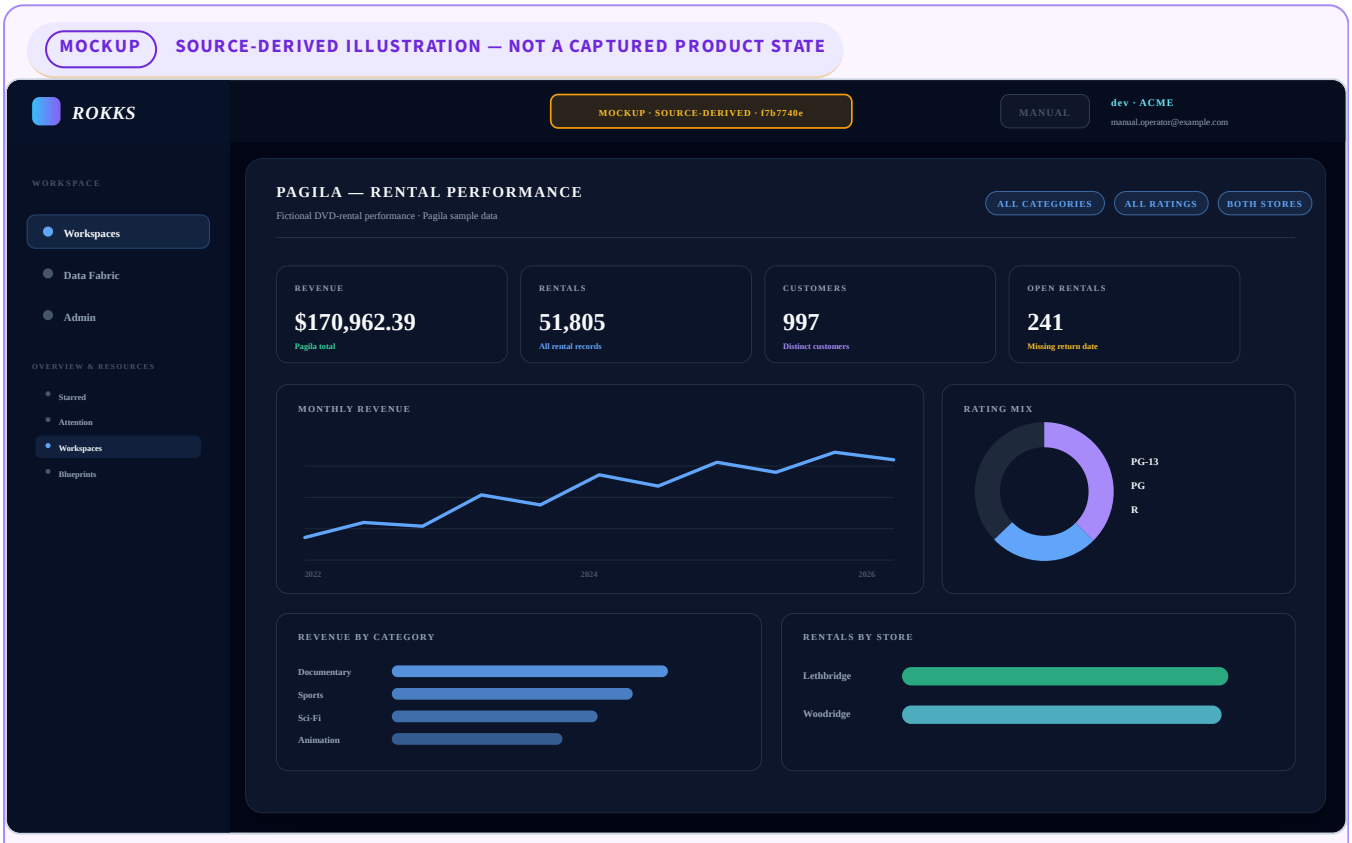
Use the catalog relationship `payments.rental_id → rentals.rental_id` whenever a widget combines payment measures with rental, film, category, or store fields. Continue through the declared rental relationships rather than matching similarly named fields by sight.

Widget	Qualified fields and relationship path	Definition
Revenue	`"payments"."amount"`	`SUM("payments"."amount")`
Rentals	`"rentals"."rental_id"`	Count rentals
Average Rental Hours	`"rentals"."rental_hours_h"`	Average returned-rental duration in hours
Customers	`"rentals"."customer_id"`	Distinct count
Customer Directory	`"customers"."customer_name"`, `"customers"."city"`, `"customers"."country"`	Data table of customer locations
Monthly Revenue	`"payments"."amount"`, `"payments"."payment_date"`	Sum revenue by payment month

Widget	Qualified fields and relationship path	Definition
Revenue by Category	`payments.rental_id → rentals.rental_id; rentals.film_id → films.film_id; "films"."category_name"`	Sum `"payments"."amount"` by film category
Revenue by Store	`payments.rental_id → rentals.rental_id; rentals.store_id → stores.store_id; "stores"."city"`	Sum `"payments"."amount"` by store city

Run each query in **Preview**, then close one configured widget at a time. This makes an incorrect field or aggregation easy to isolate.

> **MOCKUP — verified capture pending.** This source-derived asset is a proposed dashboard layout, not a captured product state. Use only the fixture-certified values below for acceptance.



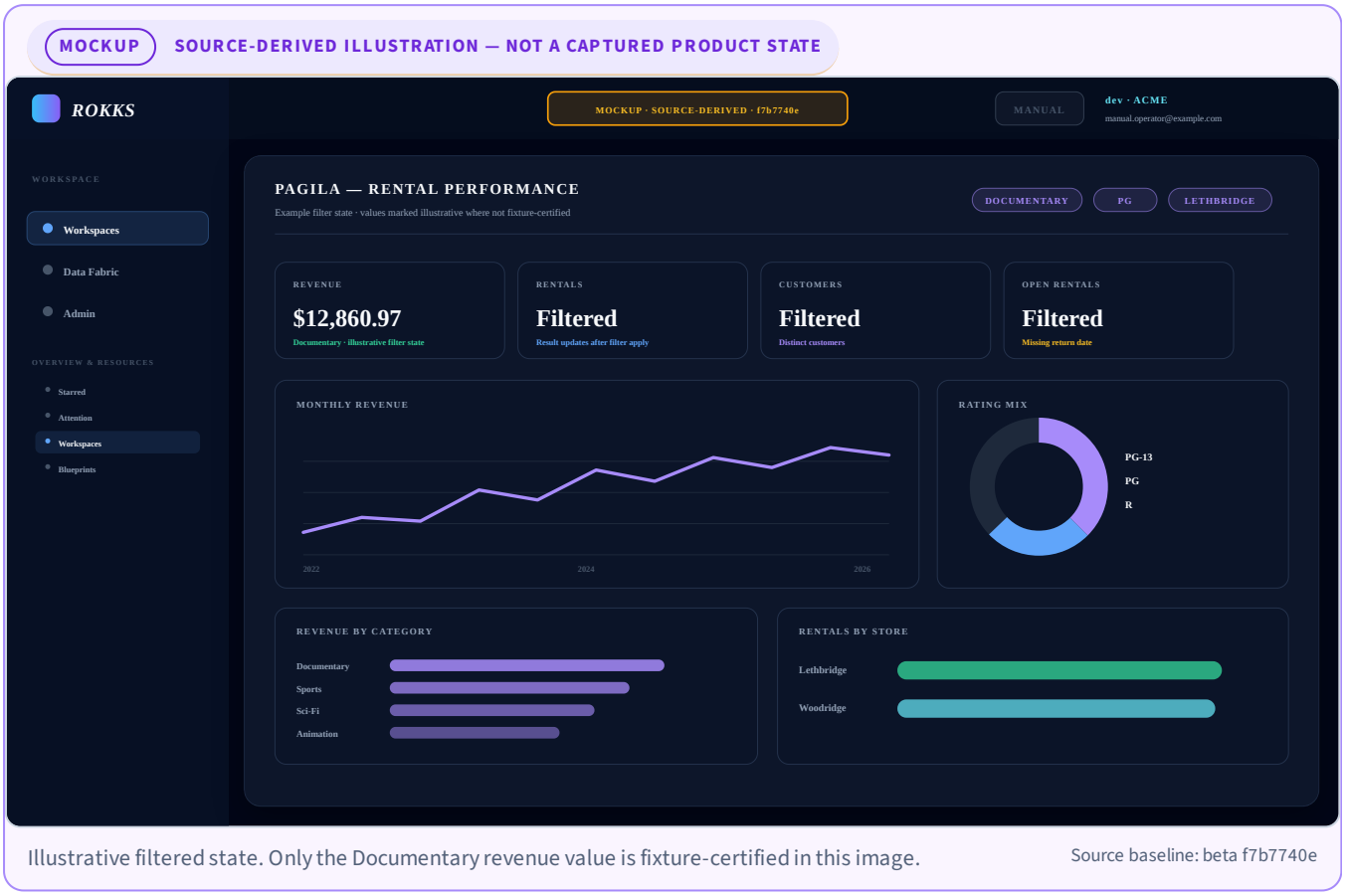
Proposed Pagila dashboard layout. Certified all-data totals come from the pinned public fixture. Source baseline: beta f7b7740e

Step 5: Exercise a dashboard filter

Click **Add or remove filter dimensions** and add `"films"."category_name"`, `"films"."rating"`, and `"stores"."city"` in **Filter Dimensions**. Close the picker and wait for recompilation. Open the `"films"."category_name"` chip and select **Documentary**. With no other filters active, the Revenue result should be **12,860.97 USD**. Click **Clear all** and confirm that the all-data values return.

You can then explore `"films"."rating"` and `"stores"."city"`, but treat combinations of multiple filters as exploratory until you verify their results against the loaded tables.

MOCKUP — verified capture pending. This source-derived asset deliberately shows an illustrative multi-filter state. Its own label identifies it as a mockup, and its combined-filter values are not acceptance figures.



Illustrative filtered state. Only the Documentary revenue value is fixture-certified in this image. Source baseline: beta f7b7740e

Expected results

Use these fixture-certified values for acceptance:

Check	Expected result
Constellation relationships	5
Categories rows	16
Stores rows	2
Customers rows	997
Films rows	958
Rentals rows	51,805
Payments rows	51,056
First rental timestamp	2022-02-14 15:16:03+00
Last rental timestamp	2026-07-28 22:25:29.109854+00

Check	Expected result
Distinct films	958
Distinct customers	997
Film categories	16
Stores	2
Open rentals (`return_date` empty)	241
Total revenue	170,962.39 USD
Highest-revenue category	Documentary — 12,860.97 USD
Rentals for Store 1	25,761
Rentals for Store 2	26,044

These aggregate checks are recomputed from the JSON tables through the five declared relationships. The JSON generator aggregates payments by `rental_id`, keeps the earliest matching `payment_date`, and omits the 749 rentals that have no payment from the `payments` table. The flat derivation records those same rentals as `0.00` with an empty payment date. Both fixtures calculate `rental_hours` for returned rentals and leave that field empty for open rentals.

Generic CSV practice

Use CSV to Dashboard when you specifically want to practice the regular-file **Source** → **Sample** → **Plan** → **Load** lifecycle with `pagila-rental-facts.csv`. That tutorial is separate from the Constellation acceptance path above.

Every image in this manual carries its own rendered provenance label. Read **BETA PRODUCT**, **LIVE PRODUCT**, **MOCKUP**, or **CONCEPTUAL ILLUSTRATION** on the individual asset instead of inferring provenance from a page-wide statement. The fixture totals above come from the pinned public JSON and its deterministic derivation records, not from image pixels.

Common mistakes

- Do not upload confidential data for this exercise. Use the supplied public Pagila fixture.
- Do not describe the six-table Constellation as the full Pagila schema. It is a reduced relational projection with complete rental and derived payment facts for this fixture.
- Do not look for imported Constellation tables under **Local Tables**. Inspect them from the File Areas **Output tables** list, Analytic Warehouse, or a catalog-backed editor.
- Do not confuse a document-field size policy with the file upload maximum. The source lives in blob storage and must fit the separately configured upload limit.
- Do not treat AREA_ETL_SYNC **DONE** as proof that its child load succeeded. Require child `FILE_ETL` **DONE** and **6/6 loaded**.
- Do not mark `rentals.return_date` or `rentals.rental_hours` as required. Both are empty for the 241 open rentals; `payments.payment_date` is populated on every payment row because unpaid rentals are omitted from that table.

- Do not compare a multiply filtered mockup with an all-category certified total. Clear unrelated filters before checking Documentary revenue.
- Do not create several widgets before testing the first query. Verify Monthly Revenue first, then add one widget at a time.

Attribution and license

Pagila © Devrim Gündüz. Pagila originated as a PostgreSQL port of the Sakila sample database, initially developed by Mike Hillyer. The pinned upstream is `pagila-v4.0.0` at commit `481abd8fd518fec9abeba14db9ed1a2895c9bd33`, distributed under the MIT License.

Related

- File Areas
- Transform Plans
- Job Queue
- Widget Builder
- CSV to Dashboard

CSV to Dashboard

Goal

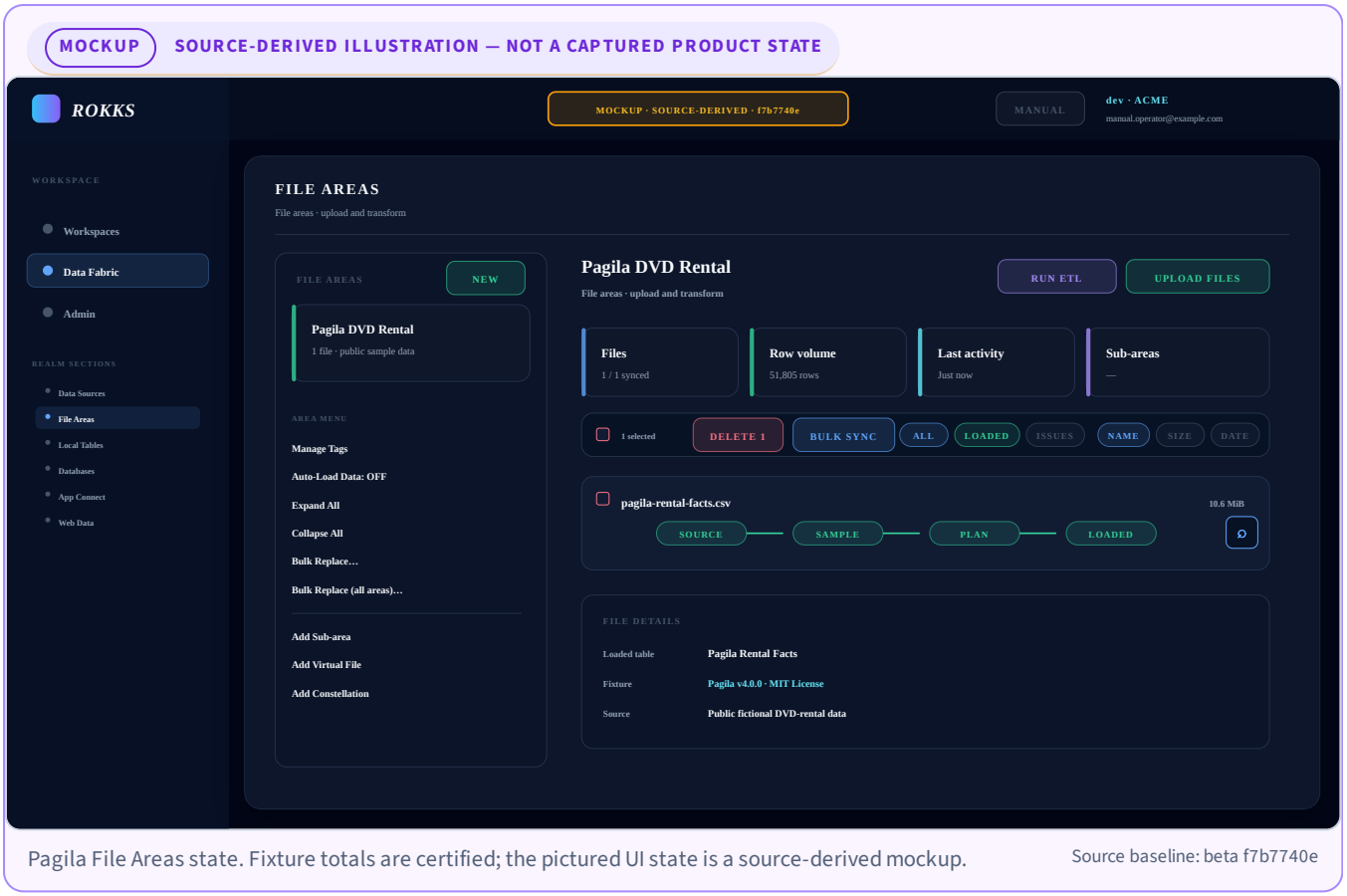
Load a flat CSV file, review its per-file transform plan, and create a dashboard widget from the resulting governed source. This tutorial uses the public Pagila rental facts CSV as a deterministic sample; the relation-aware Pagila maiden flight uses the six-table JSON instead.

Download the CSV source and derivation record with the sample. It pins 11,138,814 bytes, 51,805 rows, 21 columns, and SHA-256 `0e7a033d85e4e5657e2fefd5e81f68c7c067de92a43e3dae4b96ad75aef4a2ee`.

Steps

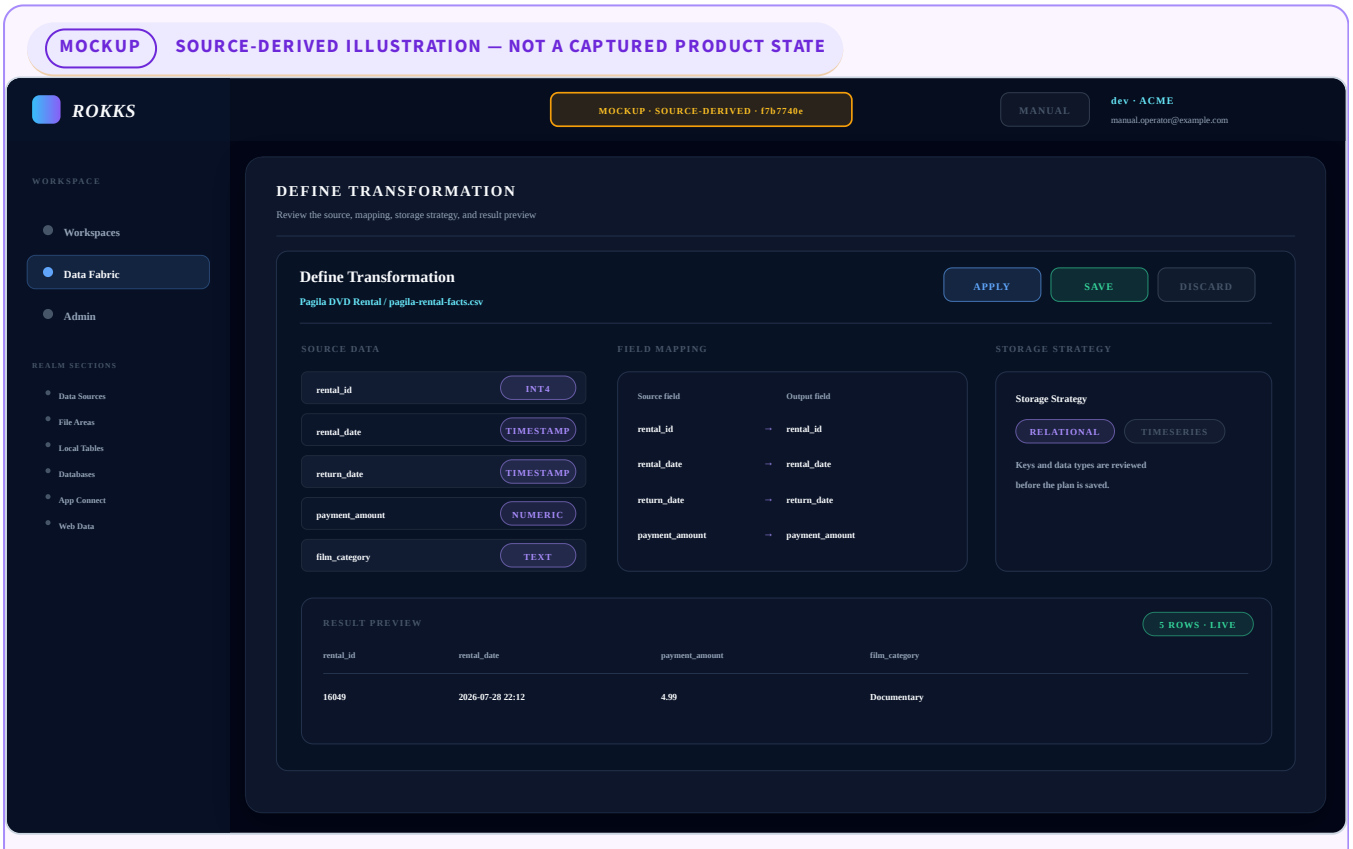
1. Open **Data Fabric** → **File Areas** and create a regular File Area named **Pagila CSV Practice**.
2. Choose **Upload Files** and upload `pagila-rental-facts.csv`.
3. Wait for **Source** and **Sample**, then open **Plan**. Review the 21 flattened fields and correct field names, types, units, nullability, and keys.
4. Run **Load** and wait for the lifecycle to reach **Loaded**. If you can access **Job Queue**, require the `FILE_ETL` job to reach **DONE**.
5. Open **Workspaces**, enter or create **ACME Analytics**, and create a practice dashboard.
6. Choose **Add column**, then use the empty slot labelled **Add or drag widget here**. The slot creates a new widget directly in edit mode.
7. In the **Data Model** canvas, select the governed table produced by the file load.
8. Choose a line chart and configure `SUM(payment_amount)` by month of `rental_date`.
9. Choose **Run Query** and inspect the preview.
10. Wait for the first instant save to finish, then choose **Close**. The Widget Builder has no separate Apply or Save action.

> **MOCKUP — flat CSV tutorial.** This source-derived asset shows the regular File Area used by this exercise. It is not evidence that the separate Constellation maiden flight was loaded.



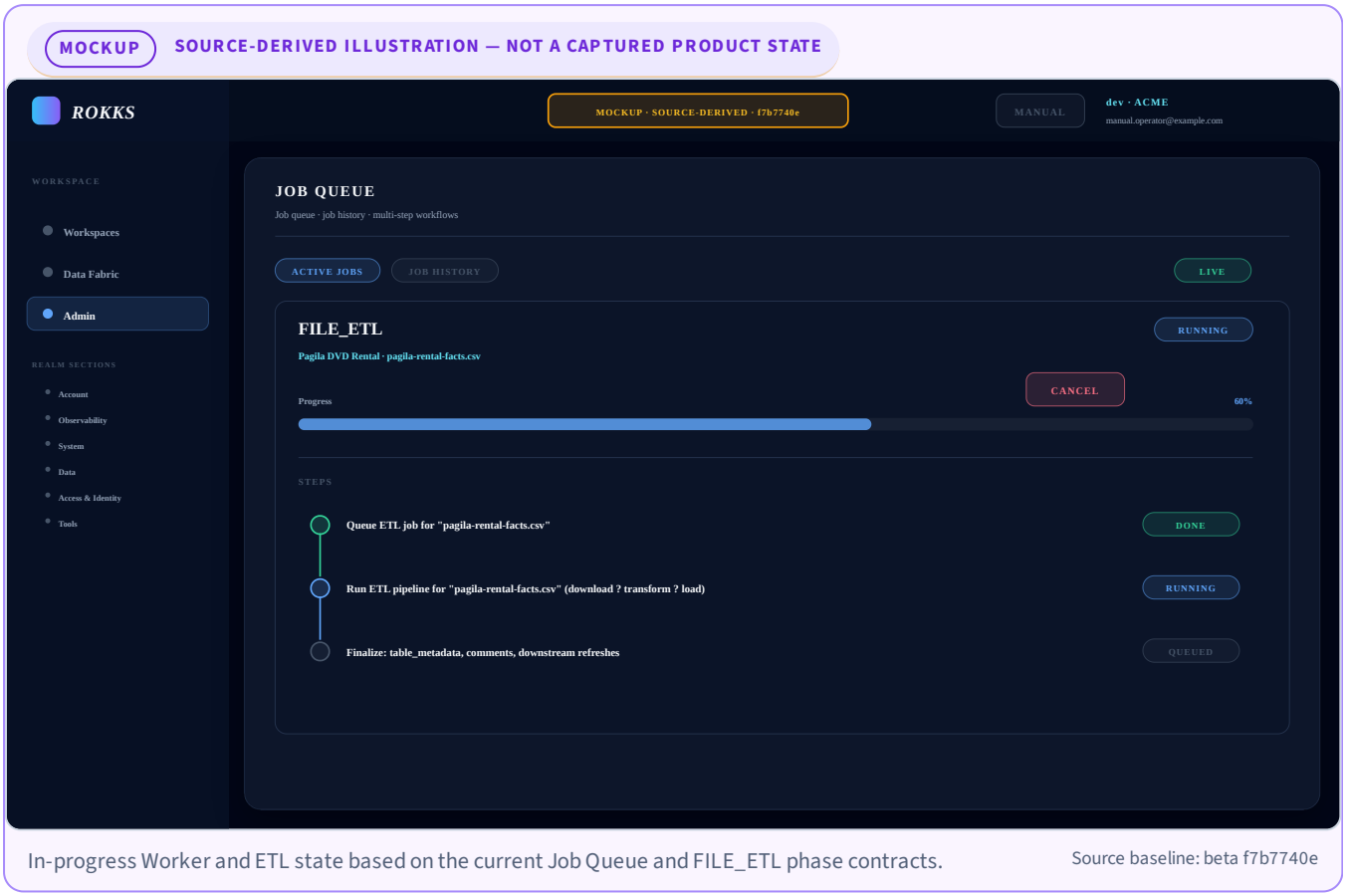
Pagila File Areas state. Fixture totals are certified; the pictured UI state is a source-derived mockup. Source baseline: beta f7b7740e

MOCKUP — flat CSV tutorial. This source-derived asset shows the per-file **Define Transformation** controls for the flattened CSV, not the six-node Constellation editor.



Pagila transform mapping based on the current File Areas plan surface. Source baseline: beta f7b7740e

MOCKUP — flat CSV tutorial. This source-derived asset shows a flat-file `FILE_ETL` job. Read its own **MOCKUP** label; it is not a configured product capture.



In-progress Worker and ETL state based on the current Job Queue and FILE_ETL phase contracts. Source baseline: beta f7b7740e

Common mistakes

Do not skip the transform review. A wrong type in the source will show up later as missing chart options, bad sorting, or failed filters.

Do not use production data for a first learning run unless your organization has approved that workflow.

Do not use this flattened table as the relation-aware maiden-flight model. Complete the JSON Constellation workflow when you need separate tables, declared relationships, and qualified fields.

Related

- File Areas
- Transform Plans
- Widget Builder
- Pagila Maiden Flight

Web API Stream to Alert

Goal

Create a web data stream, sync it into Rokks, and add an alert that watches the resulting data.

Steps

1. Open **Web Data** and create a connection.
2. Add a stream for one endpoint.
3. Test a sample response.
4. Review the generated transform plan.
5. Configure a sync schedule.
6. Wait for Dispatcher to submit the first scheduled sync, then confirm in the Job Queue that it finishes successfully.
7. Build a small dashboard widget from the stream.
8. In the Widget Builder, create and run a continuous-analytics rule that emits the signal you want to monitor.
9. Open **Workspaces** → **Attention**, choose **Manage rules**, then choose **Add rule**. This action remains unavailable until at least one continuous signal exists.
10. Select the existing signal, configure its condition, threshold, duration, cooldown, severity, and delivery channels, then choose **Save**. The alert editor has no test action.

Manual Web API **Sync** and **Full Sync** actions are currently unavailable in the UI and Sidekick. This tutorial therefore uses the first Dispatcher-scheduled run as the end-to-end validation.

Common mistakes

Do not build the widget or alert before the first scheduled sync has completed and its output has been validated.

Do not try to create an alert directly from a source field. Alert Rules watch existing continuous-analytics signals; create and run the signal first.

If the alert does not trigger, check the stream job first. A rule can be correct while the source data is stale, empty, or loaded into a different tenant than the one you are testing.

Related

- Web API Connections
- Web API Streams
- Alerts

App Connect Dashboard

Goal

Use the Airtable connector to select tables for a constellation, sync them, and build a dashboard from the resulting catalog sources.

Steps

1. Open **App Connect**.
2. Select **Airtable**, choose **New**, and enter the connection name and Personal Access Token.
3. Choose **Save & Test**.
4. Open the constellation panel and select the tables required for the dashboard.
5. Assign the appropriate fact roles. The connector snapshots Airtable fields and types automatically; there is no field-type or key editor in this workflow.
6. Choose **Save & Sync** and monitor the resulting work.
7. Open a dashboard.
8. Add widgets from the synced sources.
9. Review filters and wait for each automatic persistence or compile indicator to finish; the dashboard has no separate Save action.

Common mistakes

Do not select every Airtable table on the first pass. Start with the tables required by the business question.

Do not ignore relationships. Linked records often explain how dashboards should group or filter data.

After the first dashboard works, add only one improvement at a time. For example, add another table, validate the widget, then add the next source. This makes constellation mistakes easier to find.

Related

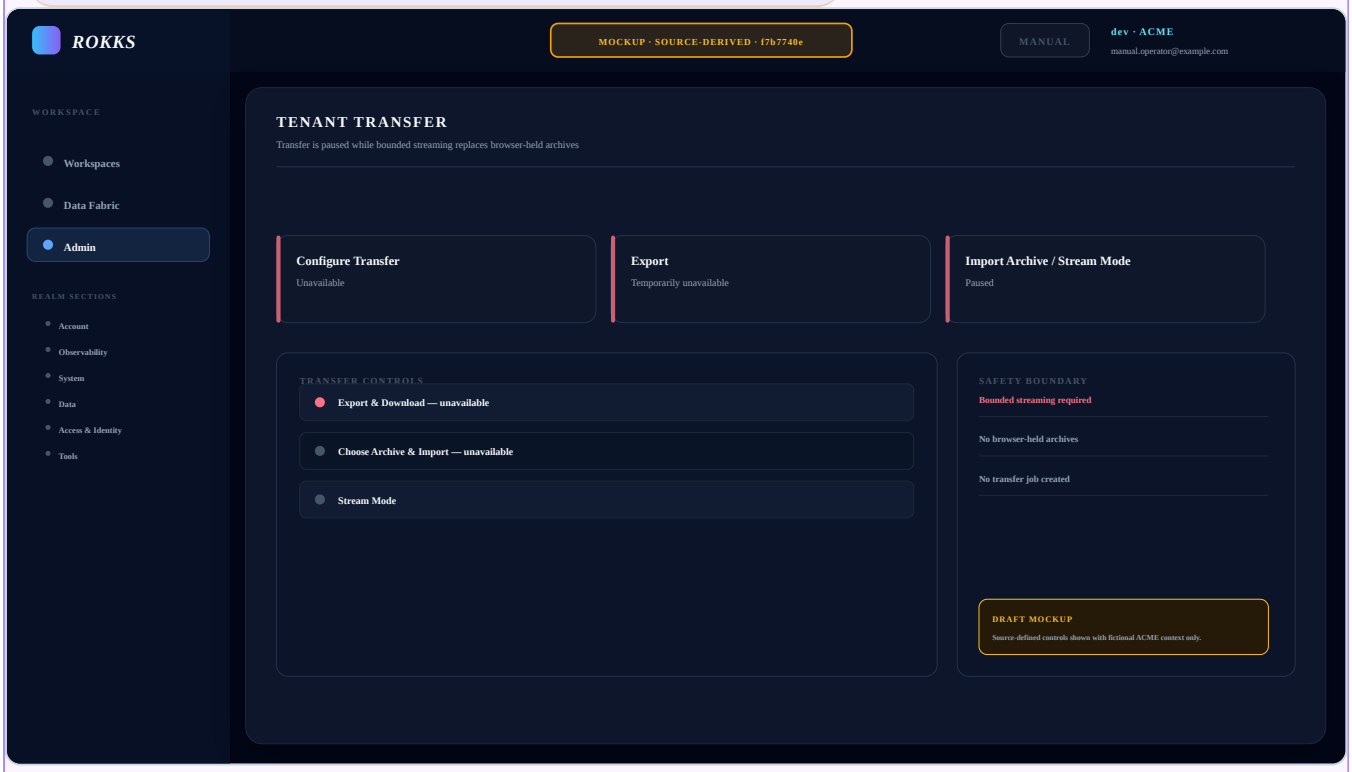
- App Connect
- Data Source Connector
- Connector Lifecycle

Tenant Onboarding

Goal

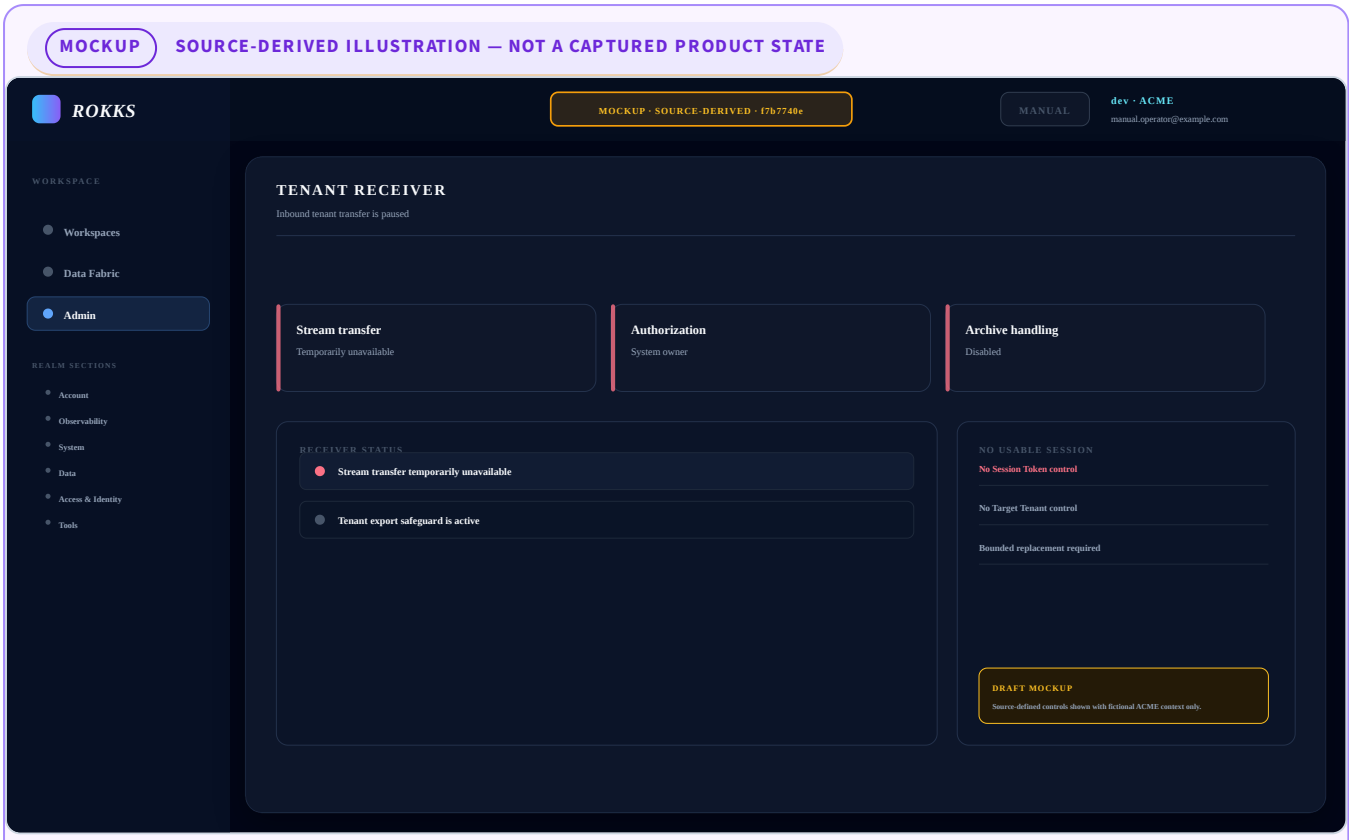
Create a clean tenant workspace, assign access, load initial data, and confirm users can build or view dashboards.

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Tenant Transfer illustration showing that export, import, and streaming are paused.

Source baseline: beta f7b7740e



Tenant Receiver illustration showing the current fail-closed streaming state. Source baseline: beta f7b7740e

Steps

1. Open **Tenants**.
2. Create the tenant with a clear business name.
3. Assign users or groups according to your access policy.
4. Select the new tenant.
5. Add initial data sources in the Data Fabric.
6. Create or copy starter dashboards.
7. Confirm jobs finish successfully.
8. Ask a pilot user to sign in and verify the workspace.
9. Add alert rules or schedules only after the core workflow works.

Common mistakes

Do not onboard a tenant in the wrong environment. Check both selectors before creating data.

Do not start with too many dashboards. A small, verified starter set is easier for users to trust.

Do not skip the pilot login. A workspace that looks correct to an administrator can still have missing user access, hidden dashboards, or an initial scope that does not contain the intended tenant.

Related

- Tenants
- Environment and Tenant Scope

- Backup and Restore

SECTION 9

End User Reference

6 topics

Feature Matrix

Summary

This matrix lists the major public capabilities covered by the user manual and integrator guide.

Fields / Parameters

Area	Type	Description
Data Fabric	User feature	Files, web APIs, App Connect, local tables, schema review, and warehouse outputs.
Dashboards	User feature	Collections, widgets, charts, tables, filters, drilldowns, styles, and history.
Sidekick	User feature	Context-aware assistant for dashboard creation, refinement, and explanation.
Android app	Mobile feature	Native dashboard, alerts, and settings companion for read-focused mobile workflows.
Operations	Cross-workflow feature	Job Queue, monitoring, Service Stack configuration, local document-backup inspection, and updates live under Admin; alert rules and breached-widget attention live under Workspaces. Deployment backup/recovery is operator-managed; UI export is not currently connected.
Tenants	Admin feature	Isolated business workspaces and tenant-level lifecycle actions.
Integrator APIs	Integration feature	Authenticated, scoped route families and job observation.

Example

```
Need to import data? Start in Data Fabric.
Need to monitor a failed sync? Start in Job Queue.
Need to explain deployment behavior? Start in Integrator Guide.
```

Related

- Platform Overview
- Widget Capabilities
- Mobile Capabilities
- Worker Job Types

Worker Job Types

Summary

Worker jobs execute background work that should survive browser closure or take longer than a quick request.

Registration describes a handler, not who may invoke it. The Worker separately checks the exact job type, authenticated role or service identity, environment, tenant, and system scope.

Fields / Parameters

Family	Type	Description
Data load	Job family	File loads, area syncs, PDF extraction, and virtual file processing.
Connector sync	Job family	Web API and application connector synchronization.
Dashboard	Job family	Widget compilation, widget deletion, dashboard generation, and refinement.
AI generation	Job family	Transform plans, stream plans, and assistant-supported generation.
Tenant operations	Job family	Import, purge, transfer preparation, and remapping. The export handler is registered but paused.
Maintenance	Job family	Logical view refresh, retention policy, materialization, migrations, and cleanup. Availability varies by exact job type.

Availability notes

Job type or class	Current submission policy
`WIDGET_COMPILE`, `REFRESH_LOGICAL_VIEWS`, `WAREHOUSE_PUBLISH`	Tenant-member authoring jobs. `WAREHOUSE_PUBLISH` accepts exactly the environment, tenant, and persisted warehouse query id; the Worker owns graph validation, SQL expansion, target-catalog CREATE-vs-SWAP attestation, staged publication, and durable receipts. Successful publication hands the server-owned source query to the same tenant optimizer scope.
`DERIVED_TABLE_REFRESH`	Tenant member for `refresh`; tenant operator for `drop`.
Object-destructive soft-delete jobs and `WIDGET_DELETE`	Tenant operator.

Job type or class	Current submission policy
`TENANT_PROVISION`, `TENANT_PURGE`	System owner.
`WEBAPI_SYNC`, `MATVIEW_REFRESH`	Dispatcher-only over the HTTP mesh path. `MATVIEW_REFRESH` accepts only environment, tenant, and persisted query id; the Worker reloads the exact catalog and derives the physical target/cascade.
`OPTIMIZER_INDEX_BUILD`	SQL Gateway only. The payload is exactly environment, tenant, physical table, ordered physical columns, and the deterministic index name. The Worker validates before persistence, deduplicates by exact environment + tenant + table + index name, and reloads the exact parent/chunk catalog. Fresh usable prefix coverage returns `already-covered` without DDL or an ownership claim. Otherwise the Worker emits server-generated online DDL without a wall-clock transport timeout. Cancellation is honored before that non-transactional DDL boundary; after DDL starts the Worker waits for the terminal outcome, reconciles the result, and re-reconciles a fresh catalog before accepting its durable ownership receipt.
`MIRROR_SQLITE_TO_POSTGRES`	Dispatcher, or a system owner with explicit system scope.
`TENANT_EXPORT`	Registered under tenant-operator policy, but execution is paused. New submissions return HTTP `409` / `WORKER_JOB_EXECUTION_PAUSED` before persistence; after generic run-lifecycle bookkeeping, recovered rows fail before export peer resolution or tenant-data access/effects.
`TENANT_IMPORT`	Registered under tenant-operator policy, but execution is paused. New submissions return HTTP `409` / `WORKER_JOB_EXECUTION_PAUSED` before persistence; after generic run-lifecycle bookkeeping, recovered rows fail before import peer resolution, archive download, or tenant-data access/effects.
`CHECKPOINT_PURGE`	Registered but dormant: it has no HTTP submitter and automatic scheduling is paused.
`HARD_PURGE_ENTITY`	Registered but dormant: submission and recovered-row execution are paused pending atomic retained-version deletion.

The Worker has no generic manual index-analysis, create, apply, or drop job. Observer-driven analysis and its owned index lifecycle belong to SQL Gateway's autonomous optimizer and use server-owned statements and telemetry rather than client SQL. `OPTIMIZER_INDEX_BUILD` is the optimizer's narrow internal effect protocol, not a client-accessible SQL surface. `WIDGET_COMPILE` separately retains its intentional compile-time `idx_auto_*` lifecycle.

System-scoped job types additionally require a system owner and the explicit system-scope header. HTTP callers cannot select the Worker's internal `PHYSICAL` phase. A service's mTLS identity does not grant access beyond its exact service-to-job allowlist.

Example

```
QUEUED → RUNNING → DONE
    └──────────────────────────→ CANCELLED
               └──────→ ERROR
```

Job identifiers use the `j-` prefix, for example `j-7f3a91c2-d4e`. `PENDING` is a step state inside a job; a newly accepted job itself is `QUEUED`.

After a Worker restart, an interrupted job may be re-enqueued from the beginning of its handler. This is restart recovery, not mid-handler resume or exactly-once execution.

Job progress and result projections are not bulk-data channels. The retained tenant-export browser marker is a defense-in-depth refusal, not a transfer mechanism. Export and import require a dedicated authorized, size-bounded, backpressured, cancellable streaming architecture before either can be enabled.

Alert notification is not a Worker job family. Sentinel emits browser events and delivers configured external channels directly when an alert changes state.

Related

- Worker Jobs
- Job Queue
- API Overview

Widget Capabilities

Summary

Widgets turn governed data into dashboard content such as charts, tables, scorecards, maps, and operational lists.

Fields / Parameters

Capability	Type	Description
Metric selection	Data	Numeric or calculated values used in charts and scorecards.
Dimension selection	Data	Categories, dates, or grouping fields.
Timeline	Data	Time range filtering and time-aware display.
Filtering	Interaction	User or dashboard controls that narrow data.
Sorting	Interaction	Ordered results for tables and ranked visuals.
Formatting	Presentation	Labels, units, conditional styles, and data-state colors.
Preview	Validation	A live result check while configuring a widget; edits to an existing dashboard widget save instantly.

Example

```
A revenue trend widget uses a source, a revenue metric, a month dimension, a
timeline, and optional filters.
```

Related

- Widget Builder
- Charts
- Tables

Mobile Capabilities

Summary

The Android app is a native mobile viewer for Rokks dashboards, alerts, and scoped account settings.

Fields / Parameters

Capability	Type	Description
Server selection	Setup	User enters the approved Rokks application URL.
Mobile sign-in	Authentication	Supported mobile authentication methods are discovered from the server.
Environment selection	Scope	User chooses an allowed environment.
Tenant selection	Scope	User views data in one active tenant at a time.
Workspaces	Viewer	Lists accessible workspaces, then their dashboards and reports.
Dashboard detail	Viewer	Renders supported widgets in mobile-friendly cards with server-defined filter and time controls.
Cached dashboard	Offline support	Shows last rendered dashboard data with stale-state context.
Alerts tab	Notifications	Shows accessible events grouped by Today/Earlier; tapping an unread card marks it read.
Native push	Conditional	Token registration and receiving code are present, but deployment support and Android notification permission are required; this beta has no in-app permission prompt.
Settings	Account	Server, tenant, app information, and sign-out controls.

Example

```
An ACME operator opens the Android app, selects the production environment and the
ACME tenant, opens a workspace dashboard, and reviews the Notifications inbox.
```

Related

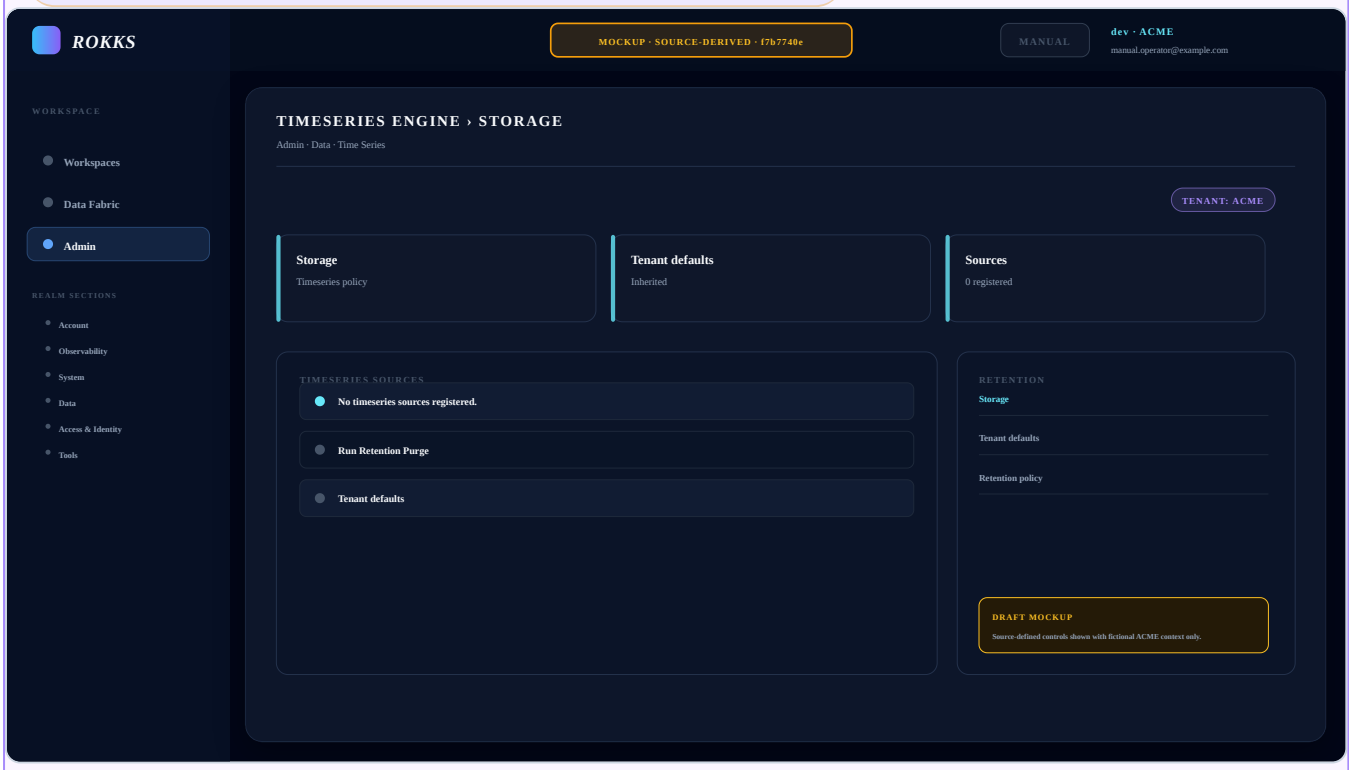
- Android App
- Android Client Integration
- Feature Matrix

Data Types

Summary

Data types tell Rokks how a field can be sorted, filtered, aggregated, displayed, and used in alerts.

MOCKUP SOURCE-DERIVED ILLUSTRATION — NOT A CAPTURED PRODUCT STATE



Time Series illustration showing storage, tenant defaults, and the empty source state. Source baseline: beta f7b7740e

Fields / Parameters

Type	Type	Description
Text	Field type	Names, categories, statuses, and labels.
Number	Field type	Measures that can be summed, averaged, ranked, or compared.
Timestamp	Field type	Dates and times used for timelines, ordering, and schedules.
Boolean	Field type	True or false flags.
Enum	Field type	Controlled sets of known values.
JSON-like value	Field type	Nested or semi-structured content that may need transformation before analysis.

SQL preview responses preserve SQL `NULL` as JSON `null`. PostgreSQL non-finite floating-point values use the strings `"NaN"`, `"Infinity"`, and `"-Infinity"` because JSON numbers cannot represent them truthfully.

Example

```
Use a timestamp for order date, a number for revenue, and text or enum for order
status.
```

Related

- Transform Plans
- Schema Editor
- Tables

Errors

Summary

Rokks errors are most useful when read with the current environment, tenant, resource, and job context.

Fields / Parameters

Category	Type	Description
Authentication	Access	The user must sign in again or lacks permission.
Scope	Access	The selected environment or tenant is missing or wrong.
Source	Data	The external provider, file, or stream returned unexpected data.
Transform	Data	Raw values did not match the configured mapping or type.
Query	Dashboard	A widget query could not run or returned invalid columns.
Job	Background	A long-running operation failed in one of its steps.
Service	Operations	A platform service or dependency is unavailable.

For SQL Query Stream v1, use the stable `code` instead of matching message text. Before response streaming, HTTP `400/405/406/413/415/417` identify request-contract failures; `503` means execution or lifecycle capacity is unavailable, and `504` means the initial-header or bounded database phase deadline expired. After streaming starts, the final `terminal.error` carries the bounded code, message, optional position, and retryability signal. A missing/invalid terminal is a protocol failure and provisional rows are discarded.

Example

```
If a Web API sync fails, open the stream, then the related job, then the monitoring
page if the error points to a service.
```

Related

- Troubleshooting
- Job Queue
- Monitoring

EDITION RECORD

About this manual

This PDF and the online manual use the same English MDX source. The online edition remains canonical for deep links and later corrections.

WEB PRODUCT COMMIT

`f7b7740e7371bad8317a547dd2608ac2706fefd7`

WEB BETA VERIFIED

2026-09-12T17:42:57.314Z

ANDROID BETA COMMIT

`ed5d9494be9e5d3919d0c34d9d45440457e96ca3`

ANDROID APK SHA-256

`c232d94c4a42bfb33b8b7c11881228c70339afb5c5af7495f24debb5d553d152`

UI CENSUS SHA-256

`8cadbae757a0cc817f762c213eb1e2ebce028c08801e573c7ee33593c9e49279`

SCREENSHOT MANIFEST SHA-256

`9ac7063ee141412fa166ef6fe74d65ad506471a33170f6a8cf6adf8baae73746`

Sample data attribution

Tutorial examples use Pagila, a fictional PostgreSQL DVD rental database. Pagila is distributed under the MIT License. Source revision`481abd8fd518fec9abeba14db9ed1a2895c9bd33`; reduced six-table Constellation checksum`83281cc7936bf37a8f9a232ab1f468b4b4fe5664edb6a01d5b176dcb00c7ae4a`; deterministic flattened CSV derivation-source checksum `0e7a033d85e4e5657e2fefd5e81f68c7c067de92a43e3dae4b96ad75aef4a2ee`.

Pagila source and licence

Visual labels

BETA PRODUCT

A capture from the configured beta product.

LIVE PRODUCT

A capture from the released product.

MOCKUP

A source-authentic illustration, not a configured product capture.

CONCEPTUAL ILLUSTRATION

An explanatory diagram rather than a UI capture.