# TOC

# Composer 24

# Introduction to Composer 24

Designed specifically for software teams, Composer delivers the first out-of-the-box development experience for embedded analytics. Choose Composer for:

- Its effortless authoring paradigm empowers developers to easily create, customize and embed data visualizations with complete control over the end user experience.
- The self-service embedded with a visualization can be tailored and configured to match the skill level of your end users, while enabling them to modify and share their own visualizations.
- Its internal query engine (z-Engine) and smart data connectors unlock unmatched modern data connectivity and query performance while also working with your existing tech stack.
- Its cloud-ready microservices architecture provides you with elastic scale so that your CFO loves you and your DevOps team respects you.

**Important:** We have changed Composer to a quarterly release schedule, and adopted a new version numbering system. Composer v8 is now Composer 22.4. Composer version names for Composer 7 and earlier remain unchanged. See Composer Release Vehicles.

**Important:** Composer v7, a passive support release, is supported with service packs that address only critical patches and security updates. Composer v7 is in passive support as of September 1, 2023. Security updates are not applicable to third-party dependencies. Changes to existing functionality, bug fixes, and workarounds are described in the Release Notes. Composer v7 is at end of life as of September 1, 2024. See Composer Release Vehicles.

**Important:** Older license keys are not compatible with Composer 6.9 and higher. If you are upgrading from any older Composer or Zoomdata release, see Request And Apply A New License Key.

If you are already familiar with Composer and want to read up on the latest updates, see: Composer 24 Release Notes Overview. For information about hardware and software requirements for Composer, see System Requirements.

For information about the Postgres metadata repository that Composer uses to store definitions and settings, see Metadata Repository.

When you are ready to get started with Composer, see Get Started With Composer 24.

For information about the job skills necessary to use Composer, see Composer Personas.

# Effortless Authoring

Composer's effortless authoring allows your developers to avoid heavy data modeling using a data authoring workflow that streamlines data connectivity, preparation and enrichment. You can rapidly build embedded analytics content with an easy-to-use visual authoring workflow and create a persistent look and feel

by seamlessly embedding analytics into your application with custom themes and extensibility that goes beyond just colors and styling.

You can build visuals once and reuse them to populate multiple dashboards, eliminating the need to recreate them over and over again. Finally, you can enjoy the ultimate end user control that a complete embedded analytics development environment provides to develop and manage content directly in your existing application's workflow.

See also:

- Configurable Data Sources
- Configurable Data Visualization
- Product Customizations
- Web-Based User Interface

# Embedded Self-Service

Using Composer's embedded self-service, your end users will enjoy the freedom to securely collaborate and share content on-demand or with automated scheduled delivery.

While other products take a one-size-fits-all approach, we empower your application teams to embed and customize end user self-service with embedded visualizations. Granular controls allow you to define levels of end user self-service and dashboard interactivity at the feature level. Precise controls over end user data access and governance ensure a secure discovery experience. You can provide end users with the freedom to visually author analytic content and perform ad hoc analysis all within your application.

In addition, Data DVR enables your end users to explore and interact with embedded content, including play, rewind, pause and fast forward of raw data and charts.

See also:

- Embeddable Analytics
- Data Playback And Live Mode

# Query Engine and Smart Data Connectors

Respecting the uniqueness of data stores, the internal query engine (also called z-Engine) and smart data connectors allow you to explore SaaS scale data sets with speed and scale including relational, NoSQL, multisource and other non-traditional data stores.

The query engine analyzes and optimizes queries and pushes processing down to the data store to enable the processing of thousands of concurrent user requests. Its data fusion technology empowers multisource analysis that allows end users to easily interact with visualizations from multiple data sources by virtually combining them so they appear to be from a single source.

Our broad set of smart data connectors for modern data stores such as search engines, streaming, and cloud data warehouses let you access all your data without the need to move or prepare the data in advance. Composer uses your existing data infrastructure and security framework, eliminating the need for redundant technology investments and on-going maintenance efforts by your team.

See also:

- Query Engine Microservice
- Data Connector Microservices
- Multisource Analysis
- Security
- Composer Data Protection Policy

# Microservices Architecture

Composer's microservice architecture is comprised of code that is adaptable and extensible. It runs on a wide variety of modern data platforms and architectures. It allows data store connectivity to run independently from the query engine and it allows you to build, deploy, and automate at cloud speed.

The horizontal scale provided with distributed microservices helps you avoid single points of failure and assures high availability with no proprietary hardware required. This elastic scale lets you optimize computer resources and avoid waste by scaling up and down resources such as CPU process power and RAM when and where you need it.

See also:

- Microservices Architecture
- Distributed Environment Support
- Flexible Deployment

# Microservices Architecture

The Composer platform is architected as a set of loosely-coupled Java microservices. Unlike traditional business intelligence software, which is deployed as a monolithic application, a microservices architecture allows for the following benefits:

- Faster deployment of new functionality
- Optimal resource management
- Faster recovery in some failure scenarios
- Greater deployment flexibility

Each Composer microservice runs in its own Java Virtual Machine (JVM) to optimize memory allocation and runtime attributes to meet its function and load. A failure in one microservice does not take down all Composer, and recovery is faster because only the failed microservice needs to restart. Deploying new functionality, such as an updated or custom data store connector, occurs by auto-discovery, and without requiring a server restart.

Separately, Composer centralizes its metadata store in a relational database and includes an internal messaging queue. The communication protocols used by the microservices include WebSockets (for real-time bidirectional communication) and HTTP/S.

For more information about individual microservices, see:

- Composer server microservice for its Web-Based User Interface
- Query Engine Microservice
- Data Connector Microservices
- Data Writer Microservice
- Service Monitor Microservice
- Configuration Microservice
- Service Discovery Microservice

- Screenshot Microservice
- Distributed Tracing For ComposerSymphony

The following links provide general information about all Composer microservices:

- ComposerSymphony Data Discovery Microservice Name Reference
- ComposerSymphony Microservice Startup Order
- Start ComposerSymphony Microservices
- Stop ComposerSymphony Microservices
- Restart ComposerSymphony Microservices
- Scaling ComposerSymphony Microservices
- Manage ComposerSymphony Microservices Using The Command Line Utility
- ComposerSymphony System Metrics

# Web-Based User Interface

Composer provides a single, unified web-based user interface (the Composer UI) for administrators, dashboard developers, analysts and casual users alike. The web application is compatible with modern browsers that support the HTML5 standard for web applications and WebSocket communication protocol.

Because it supports the HTML5 standard, the user interface responsively adjusts to a tablet form factor without loss of functionality. The application will function on a smartphone, however due to the smaller form factor, such devices don't lend themselves well to the out-of-the-box exploratory experience for which Composer was developed. Custom mobile apps can be developed using mobile frameworks such as Ionic or React Native.

The web browser (or any client application using the Composer JavaScript APIs) is responsible for establishing a two-way WebSocket communication channel between itself and the Composer query engine. The web browser is responsible for keeping the WebSocket connection alive and automatically reconnecting to the query engine when network problems occur.

Messages transmit over WebSockets using the lightweight JSON (JavaScript object notation) file format. There are several message types used by Composer.

| Message Type | Description |
|---|---|
| Query messages | Define the query request. Requests can be raw data requests or single or multidimensional data requests that include filters, sorts, and limits. |
| Data messages | Contain the results of a query execution. |
| Metadata messages | Contain information about the query status (progress, error, time window). |
| Control messages | Define control events, such as pause or play. |

The Composer web application delivers unique functionality to the end-user, such as:

- Data Sharpening™ to stream predictive results when working with very large data sets
- Smart loading of visuals and dashboards. Smart loading improves the performance for loading visuals on a dashboard.
- A time bar to simplify time-based analysis
- Data DVR to rewind, play, and fast forward data streams
- Live mode to visualize near real-time data streams
- Highly configurable visuals and dashboards that includes filtering and sorting the data in near real-time
- Search box and facets to maximize performance when connecting to search-enabled data platforms such as Apache Solr and Elasticsearch.

The web application pulls all these technologies and features together in a “non-blocking” data exploration experience.

There are two ways to understand the non-blocking UI. One is that data loads independently in each visual, so viewing the entire dashboard is not slowed down by the longest running query. The second is that at the same time that data rolls in, users can filter, sort, drill down, reformat, change visual styles, and otherwise manipulate visuals. The ability to interact with visuals while data loads is significant because it is what enables users to engage in speed-of-thought analysis against massive and live data sources. When a user switches direction to explore different data, Composer efficiently cancels previously running queries to conserve computing and network resources.

The value of the Composer non-blocking user experience should not be underestimated. Because many studies suggest that the human attention span for most tasks is less than 10 seconds, it’s important that your users are not stuck staring at a blank screen or a waiting hourglass. True data exploration only happens when users can rapidly and freely decide to explore where the data leads.

# Query Engine Microservice

The Composer query engine sits between the web application and the Composer data connectors.

The query engine has three primary roles:

1. It deconstructs and converts your query requests into distributed execution plans.
2. It optimizes the execution plans based on data platform capabilities, in-memory cached results, and the query engine capabilities.
3. It executes data functions that include:

i. Communicating with Composer data connectors to execute push-down queries

ii. Retrieving data from in-memory cached results, as appropriate

iii. Using in-memory processing to combine, append, or manipulate one or more data sets to produce only the values needed to fulfill your request.

The key capabilities of the query engine include push-down processing for select data sources, microqueries and Data Sharpening, adaptive caching, data playback and live mode, and multisource analysis.

# Push-Down Processing

Composer is built so users can interact directly with data down to the atomic row-level detail. Push-down processing is necessary to support this ad-hoc, interactive user experience on fresh data. As users explore the data and drill down to lower levels of detail, Composer continues to push processing down as new queries. The data store returns only the values that the query engine needs to populate the user's visuals. The Composer push-down architecture also avoids scaling up the query engine unnecessarily when complex processing can be better executed on high-performance database engines or scalable data platforms.

Composer's default processing strategy is to push down as much work to the underlying data sources as possible, for as many data sources as possible. Internal to the query engine is a query optimizer that evaluates each end-user request, and determines whether to submit all or part of the request to the target data stores. This includes pushing down filtering criteria, derived fields, custom metrics, and offset, limit, sort, and time bucketing operations.

The ability to push down filters means that the data platform engine doesn't need to scan large data sets unnecessarily. It also reduces the amount of data transferred over the network from the data source to Composer. Composer can push down all filters that a user requests in the UI.

Push-down of derived fields and custom metrics optimizes performance for the most resource-intensive operations. Composer always pushes down custom metric aggregations: min, max, sum, avg, count, distinct count, last value, and percentiles. Where advantageous, the query engine combines several simpler aggregates to compute more complex metrics.

Composer offers automatic time bucketing, which allows you to group and filter data by time categories such as current or prior week, month and year, rolling time periods, and so on. There is no need to pre-aggregate or model time buckets, freeing up technical personnel to work on other work. All that is needed is a date-

time field. The query engine does all the work of interpreting and converting user requests to one or more queries and pushing the whole operation to the data store.

The benefits of the Composer live-connect approach with push-down processing are:

- You always have access to fresh data from the data store
- Computational resources are scaled and managed where they make the most sense
- Network bandwidth is conserved
- It works very well for hybrid-cloud deployments, since there's no need for massive data movement between systems

Composer's implementation of push-down processing also allows users to explore down to the atomic, row level detail. In this way, Composer is unique in its ability to make the full breadth and depth of big data environments available for exploration.

# Microqueries and Data Sharpening

Microqueries and Data Sharpening™ are patented technologies that work together to let you interact with big data. The query engine invokes microqueries and Data Sharpening based on criteria such as the type of aggregate values requested and anticipated query run time. Microqueries and Data Sharpening are ideal for big data that is partitioned by date and that runs on a cluster with many processing cores. This functionality can be disabled when you set up the Composer data source configurations for your data store.

Microqueries, which are executed first, sample data across database partitions. The query engine submits a full long-running query that runs with the first set of microqueries. A progress indicator estimates the progress of the full long-running query. Both the full query and the microqueries run until the full query runs to completion or the user changes direction, at which point both the long-running query and microqueries are canceled to conserve processing and network resources.

Data Sharpening analyzes the sample data and streams predictive results to the your browser (or other client) over a WebSocket connection. Data Sharpening's predictive results may fluctuate a bit up or down until the final query is reported. However, the relative values of each group usually remain consistent as the data is sharpened. For example, the tallest bar in a bar chart at 10% completion will almost always remain the tallest bar at 100% completion. This means that you can be confident exploring data even as it streams live to the dashboard.

When you drill down, filter, change metrics or groupings, or perform any other action that changes data values, Composer cancels the full long-running query and microqueries to free up the data source engine for the next sequence of queries. Canceling active queries, however, is not trivial, and many JDBC drivers do not support it. In these cases, Composer's data connectors issue native API calls to complete the task.

# Adaptive Caching

The query engine optionally accelerates performance through adaptive caching. The cache eviction algorithm prefers the most frequently used data, with consideration of cache size and expiration times. Users with the appropriate privileges can configure cache sizes and time-to-live (TTL) schedules, and can forcibly clear caches.

Composer uses its cache to enhance performance in scenarios where large numbers of users are concurrently viewing the same shared visuals.

**Note:** Cached data is shared between users only if they have the same data access permissions and security context.

When caching is enabled, Composer does not requery the data source to obtain the data unless the cache is cleared or unless a refresh schedule is defined in the data source configuration. See Cache Tab and Trigger Refresh Jobs.

By default, data caching is enabled for all data sources. The Composer cache stores all the results of aggregated requests from your data source. When a visual is created the request is first sent to the Composer cache. If the required results are found in the cache, they are visualized on your visual.

Each visual that is cached has a key that uniquely identifies its content and how it can be reused. The key is calculated based on the request and user. The request provides the information related to data grouping and content, such as the data attributes and metrics. The user provides contextual information, like per-user security filters, user attributes, etc. This way all the security settings are taken into account when storing and using the cached visual. Information from the cached visual is shared between users only when they have the same data access permissions and security context.

The data cache optimizes data processing and avoids unnecessarily expensive queries. To avoid an expensive query, all or parts of multiple interactive data caches can be used to fulfill different user requests.

The cache uses normalized requests for its cache entry key. This key allows data caches to be safely used by multiple users with different security contexts and visual types. To serve requests that require different data, the query engine applies a number of possible transformations to the cached data. For example, columns can be dropped to enforce field-based security, additional filters can be applied to enforce row-level security, time windows can shift, or the data can be rolled up to higher levels of aggregation. Because there may be several ways to get the same result using different cache entries, the query engine evaluates different transformation approaches for complexity, and selects the simplest approach.

Not all data requests are cached. By default, cache entries are stored in the metadata repository, where they are retained after a service restart.

# Data Connector Microservices

Composer offers the widest set of connectors for modern data stores deployed on-premises or in the cloud, such as Hadoop, NoSQL, search engines, and modern data warehouses. In addition to the set of standard, out-of-the-box, data connectors that Composer provides, you can create custom data connectors.

Composer's standard, out-of-the-box, data connectors are smart, leveraging the unique capabilities of each data platform to take advantage of query expressiveness and to optimize performance. For example, Composer's data connectors leverage:

- Data partitions in Impala and other similar sources
- Faceted search when querying an unstructured data search engine such as Elasticsearch or Solr
- Native APIs for NoSQL databases.

The data connectors use native APIs either in whole or in part to interact with the target data store. For some modern data platforms, all user interactions are converted to native API calls. For data stores that can be queried using SQL over a JDBC driver, the data connector uses SQL when available and native API calls when necessary. For example, a data store may support query cancellation, but its JDBC driver may not. Composer data connectors are smart enough to enrich the data interaction experience beyond standard off-the-shelf functionality.

Composer's microservices-based architecture allows your developers to build connectivity to proprietary data platforms or to data for which a native connector is not currently available. Custom connectors can also be smart by making full use of the Composer query engine, including its Data Sharpening and live mode functionality as well as enabling custom security features such as user delegation and Kerberos authentication.

Custom connectors can also be used to extend Composer data connectors with additional business rules. For example, a custom connector can implement the logic necessary to query specific tables based on the end user's group membership.

Each Composer data connector deploys as a standalone Thrift server running in its own Java processing space and registers with Consul and Composer's Service Monitor as a Composer microservice.

For complete information about Composer data connectors, see Connect ComposerVisual Data Discovery To Data Stores.

# Data Writer Microservice

Composer offers a multipurpose Data Writer microservice that writes data to a relational database for an enriched analytic experience. Its current uses are to:

- Persist user-uploaded flat text or CSV files for reuse, performance, and functional improvements such as push-down processing and derived fields
- Simplify landing or persisting streaming data into a high-performance data platform for subsequent analysis
- Store user-generated keysets for “set analysis” to be used in single and multisource data environments.

The Data Writer microservice consists of a microservice that receives requests using a REST API, a message queue, and writable data connectors.

# User Uploaded Files

Users with privileges to create Composer data sources can use the Composer UI to upload flat files to Composer. The Data Writer microservice passes the text or CSV file to a message queue to manage backflow, and then writes the file contents to a database. Users then work with it as any other data source. Additional APIs are available to manage uploaded files.

# Landing Streaming Data

Any streaming engine, such as Kafka, Spark Streaming, Storm, Apex, Nifi, Kinesis, and others, can be used to process and land data in a persistent data storage environment. Alternatively, the Composer Data Writer microservice can receive live streaming data via the Composer REST API, pass it to an internal messaging queue for backflow management, and then write it to a database. After it is landed, data is accessible as any other data source for seamless live mode and historical analysis.

Composer took this approach because it provides greater functionality than connecting directly to a live stream, and requires far less administrative overhead and maintenance than a complex lambda architecture.

# Keysets: User-Directed Set Analysis

Composer offers an elegant approach to “set analysis” that allows users to explore key relationships within and between data, regardless of where the data is stored. The Data Writer microservice provides the backbone for this functionality by receiving an ordered and filtered set of “keys” from the web application, passing the keyset through the message queue, and storing it in a relational database for reuse by authorized persons. Users work independently to apply keysets to data stored on any platform. The user experience is swift and fluid, and the supporting architecture is so elegant and straightforward that no SQL or coding is ever required.

# Service Monitor Microservice

The Service Monitor microservice provides administrators with real-time views of microservice health, configuration, and logs. The Service Monitor is not installed as part of a default Composer installation, so it must be installed and configured before you can use it. Using the Service Monitor, you can:

- Review all Composer microservices and their log files
- Produce a diagnostics bundle to send to Composer Support
- Set the logging level and properties for each microservice.

For complete information, see Service Monitor Overview.

# Configuration Microservice

The Composer configuration microservice is packaged with the Spring Cloud Configuration server, which allows Composer to easily integrate with its Spring-based microservices. It provides the mechanism by which Composer property settings can be maintained using the Service Monitor. The property settings are persisted in a supported PostgreSQL data store or in a GitHub repository.

A `config-server-upload.jar` utility is available that can be used to migrate the microservice properties from your standalone Composer servers to the Composer configuration data in the PostgreSQL data store, where the configuration microservice can maintain them. See Migrate Properties To The Configuration Server.

For complete information, see Use The ComposerSymphony Data Discovery Configuration Microservice To Maintain Application Properties.

# Service Discovery Microservice

The Service Discovery microservice integrates Composer with the Spring Cloud Consul. It provides:

- External configuration capabilities for Composer via a key/value store
- A service discovery client backed by the Consul service registry.

# Screenshot Microservice

The Screenshot microservice allows you to view snapshots of your saved dashboards. See Set Up The Screenshot Microservice.

# Tracing Microservice

**Important:** The Composer Tracing Microservice (`zoomdata-tracing-server`) has been removed. See *Distributed Tracing For Composer*.

# Distributed Environment Support

You can deploy Composer microservices in a distribued environment. Such an environment ensures that you can minimize the downtime caused by:

- hardware or software failures due to excessive resource use or other events
- software upgrades or updates

You have two options for setting up a distributed Composer environment:

- **Load balancing:** Load balancing helps you scale Composer for hundreds of users. You can use load balancing both on-premises and with cloud deployments. A single set of microservices communicate with each other in a monolithic flow.

  For more information, see Configure The ComposerSymphony Server Behind A Load Balancer.

- **High Availability:** A high availability environment includes multiple Composer nodes, each with its own set of microservices. This ensures that a microservice is available at all times, somewhere in the cluster. Each microservice communicates with others using service discovery.

  Different numbers of different types of microservices can be defined for the Composer nodes, although at least two of each must be installed.

  A high-availability load balancer is required to distribute the network traffic across your user-facing Composer nodes. If only a single load balancer is deployed in a high availability environment, you will not be able to access any of the Composer nodes behind it if the load balancer should fail. Microservice load balancing and failover occur automatically within the Composer nodes themselves.

  For more information, see Configure A High Availability Environment.

If you have the configuration microservice configured and running, you can maintain properties for microservices of a given type in a single location in the Service Monitor. For example, if you have two query engine microservices running in your high availability environment, you can change the properties for both microservices in a single location, ensuring that the query engine microservices operate in the same manner across the product nodes. A `config-server-upload.jar` utility is provided that can be used to migrate the microservice properties from your standalone Composer servers to the Composer configuration data in the high availability PostgreSQL data store, where the configuration microservice can maintain them. For more information see Migrate Properties To The Configuration Server.

# Configurable Data Sources

Data queries in Composer use data source configurations to identify the data needed and the data store from which it should be obtained. A Composer data source configuration defines the following settings:

| Setting | Description |
|---|---|
| connection definition | The connection string that should be used to connect to your data store. |
| data store table, index, or data to use | The table, index, or other data in your data store that should be collected when the data source is used in a Composer query. |
| data customizations | Any customizations you want made to the data collected from the data store. This can include customizations to the field names, types, partition settings, distinct count activation for the data, filter display customizations, and other configurations such as the time pattern or granularity used for time fields or the aggregation used for numeric fields. |
| derived fields and custom metrics | Any derived fields or custom metrics you want calculated from the data collected from the data store. |
| refresh schedule | The schedule by which the data collected from the data store is refreshed. Derived field and custom metric data is also refreshed on this schedule, after the data store data is updated. |
| time bar settings | The default time bar settings that should be used for visuals and dashboards created using the data source. |

For more information about creating and managing data source configurations, see Manage Visual Data Discovery Data Sources.

In addition to standard data source configurations, Composer provides methods of analyzing data from multiple sources at the same time, including the ability to fuse data sources. See Multisource Analysis.

# Configurable Data Visualization

You can visualize data from your data stores in visuals and dashboards in the Composer UI. A dashboard is a container for one or more visuals. The visuals in a dashboard do not have to be related, but for easier data analysis, probably should be. See Multisource Analysis.

Data can be visualized using a wide variety of visual styles: arc gauges, bar charts, box plots, donuts, heat maps, KPI charts, line charts, maps, pie charts, scatter charts, tables, tree maps, and word clouds are all supported. Variations of these visual styles can also be produced. For example, you can produce standard bar charts, stacked bar charts, clustered bar charts, histograms, and multiple metric bar charts. See the ComposerSymphony Data Discovery Visual Metrics And Attributes Reference.

You can customize visuals in a variety of ways. When you define a data source configuration, you can specify some default global settings for visuals created using the data source. Read about the Global Settings Tab of a data source configuration.

After a visual is created in a dashboard, you can customize it in the following ways. Some of these customizations depend on the selected visual style or the data source configuration used for the visual.

| Customization | Description |
|---|---|
| Visual Colors | You can change the colors used in the visual. See Change Color Schemes and Change The Visual Color Metric. |
| Visual Style | You can change the visual style. See Change The Visual Style. |
| Names and Descriptions | You can change the title and description of dashboards and visuals. See Rename A Data Discovery Dashboard ,Descriptions , Modify Visual Names, Display Names, And Descriptions. |
| Data Sorting | You can sort visual data. See Sort And Limit Visual Data. |
| Data Filtering | You can filter visual data by specific field values or ranges. See Filter Data. |
| | You can filter visual and dashboard data by time ranges using the time bar. See Use The Time Bar |
| | You can filter visual data by unique data values stored in a keyset. See Use Keysets. |
| | You can filter dashboard data by cross-source linked fields from different data sources. See Use Cross-Source Links. |

In addition to these dashboard and visual configuration features, you can link dashboards and you can share dashboards with other users. You can also export visuals and dashboards in a variety of formats: PNG, PDF, CSV (visuals), and JSON (dashboards and visuals). When you export visuals and dashboards in JSON format, you can impor them into other Composer environments running the same version of Composer. See Manage Dashboards and Manage Visuals.

Finally, you can change the look of the UI altogether using themes. See Manage UI Themes.

# Multisource Analysis

Most organizations have data stored in disparate systems for a variety reasons. Regardless of how the data is stored, you may need to combine it for clarity or to analyze the relationships between data stored in different systems.

Composer dashboards can contain many visuals, and each visual can depict data from a different data store. In addition, Composer offers more practical approaches to multisource analysis through data fusion, cross-source filtering, and keysets.

- Fusion combines multiple data sources using one or more common keys so that they appear to be from a single source. The two most common reasons for using data fusion are to enrich or provide clarity to the data. One simple example is to use fusion to look up labels or other attributes. Depending on the size of the data being fused, data fusion can be resource-intensive. See Fuse Data Sources.

- Cross-source filtering allows business you to quickly apply common filters across visuals that are populated by different data sources. Cross-source filtering simplifies and accelerates exploration when a dashboard contains data that is logically related but physically in different systems. See Use Cross-Source Links.

- Keysets are an innovative way to perform multipass data analysis without requiring IT or developer intervention. You can simply create a keyset from one visual and apply that keyset as a filter on other visuals. Keysets can be applied to any visuals, and are usable across all data sources. See Use Keysets.

# Data Playback and Live Mode

There is one common attribute to all data streams: time. Any data source that has a date-time field can be playable in the Composer time bar (Data DVR). Historical data can be played back visually, just like replaying a movie. Data sources that host frequently updated data can be configured to play data in live mode. In live mode, Composer uses internal markers to automatically push incremental updates to visuals. Live mode updates are configurable for all levels of granularity available in your data source: from as frequently as once a second, to once a minute, hour, or day, to broader time frames, such as 30, 90, 180, or 365 days.

You do not need to force expensive full query refreshes, and since only newly arrived data is pushed to your visuals, network and other resources are conserved. Composer's built-in live mode functionality increases productivity and overall user satisfaction when working with very large and rapidly updated data.

Like Data Sharpening, the time bar streams data to the user over WebSocket connections. Within your WebSocket connection, Composer multiplexes commands from each visual over that connection so each visual's query request results in its own virtual communication channel that receives data streaming in one direction, and sends control commands, such as changing playback speed, pausing, or flipping the time window in the other direction.

For more information, see Live Mode And Historical Playback.

# Embeddable Analytics

Composer was built for extensibility and for integrating business intelligence into other applications. Integrations can be used to pass context between applications and Composer as part of your workflow or data exploration experience.

There are several options for embedding analytics:

- White-label for rebranding Composer to display another organization's logo, fonts, colors, etc. See Product Customizations.
- Data integration using the Composer API to run a query from your application and then display or use the data pulled by the query. The query definition is invoked by an *action* in the Composer UI and is based on the filters applied to a Composer visual and on the data and limit specifications in the application integration definition for the visual data source. The invoked action creates the query definition and sends it to your application. See Integrate Visual Data Into Your Applications.

The behavior of Composer visuals and dashboards when they are embedded into your applications is controlled by the interactivity settings for the visuals themselves. See Control How Users Interact With A Visual.

Composer REST APIs use common design practices to be easily navigable by any user. The APIs are documented with Swagger, allowing for interactive experiences when conducting experimentation and testing.

API documentation is provided with your Composer installation at this link: *`https://<composer-URL>/composer/swagger-ui.html`*.

**Important:** Some API endpoints are marked as `experimental` in the Swagger documentation we provide. These endpoints are in the early stages of design and are subject to change. We make no commitment to their stability and may remove them without notice. These experimental endpoints are not recommended for use in production.

# Product Customizations

You can customize Composer in the following ways.

| Customization | Description |
|---|---|
| Custom charts | Composer provides APIs that use web standards such as HTML, CSS, and JavaScript to expand the number and type of visuals available to you. The use of web standards allows developers to leverage popular JavaScript libraries to build custom charts, and then integrate them directly into Composer dashboards.<br><br>Custom charts make use of the same query engine that powers standard Composer visuals. Composer data connectors, microqueries with Data Sharpening, and live mode are all available with custom charts.<br><br>Custom charts are built using the Composer Command Line Interface (CLI). The custom chart CLI offers a flexible environment for creating, managing, and deleting custom charts without being connected to the client application. The CLI tool uses `Node.js` and is installed locally via `npm`. After it is installed and configured, the CLI behaves much like any other command line tool.<br><br>See Manage Custom Charts. |
| Custom connectors | See Manage Connectors and Connector Servers. |
| Derived fields | Derived fields extend each row with additional attributes or metric fields that can be used in filters and aggregations. Derived fields can be complex or as simple as concatenating text strings, such as first and last names. A full editor is available to simplify the development of complex row level expressions (RLE). Derived fields can be used in the creation of other derived fields and custom metrics.<br><br>See Maintain Derived Fields. |
| Custom metrics | Custom metrics are used to perform complex math across rows and at various levels of aggregation such as grand totals and grouped subtotals. For example, you can define custom metrics that calculate percentages of total values. Custom metrics are retained as formulas and can be used in other custom metrics.<br><br>See Maintain Custom Metrics. |
| Custom interactivity | Interactivity settings can be applied to visuals to control how users interact with a visual and, more specifically, how users can interact with the visuals after they are embedded in other applications.<br><br>See Control How Users Interact With a Visual. |
| Admin-defined functions | Composer provides a set of functions that you can use in expressions to build derived fields or custom metrics. However, you can use your own functions (for which there is no Composer equivalent) to extend your data analytics in Composer. Examples of this might be functions available within your data store either natively or as user-defined functions.<br><br>See Admin-Defined Functions. |
| UI white labeling and customization | When you install Composer, default UI settings are applied. Composer allows you to manage links to help content, customize copyright info, the default logo, and change or remove the link to terms of use. To match the style of your company, you can also |

| Customization | Description |
| --- | --- |
| | upload a custom *.css* file to modify the default Composer skin.<br><br>See Customize The Composer Visual Data Discovery User Interface and White Label The ComposerSymphony Interface. |
| Themes | When you install Composer, three themes for the UI are provided: **composer**, **modern** (light) and **dark**. You can change the theme as needed by your installation.<br><br>See Manage UI Themes. |

# Metadata Repository

Metadata is any data that is used for the configuration and runtime operation of the Composer application, with the exception of customer-managed data stores. The metadata repository is simply the relational database instance containing the Composer schemas. The primary metadata schema includes configurations for:

- dashboards, visuals, and filters
- data store connectivity
- data source configurations and statistics
- enterprise authentication, accounts, users, groups, and permissions.

The metadata repository also contains separate schemas for storing raw data, which supports upload and keyset functionality. Note that the metadata schema definitions and the stored data are internal to the Composer application; they are not intended for direct manipulation.

Composer uses relational database technology for metadata storage, and ships with an open-source PostgreSQL database.

**Important:** Composer uses a packaged PostgreSQL database instance to store its metadata. Use the provided instance due to the specific configuration and version combination:

If you would like to use another PostgreSQL instance, contact Technical Support for further guidance.

**Note:** New installations of Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

Internally, industry-standard technologies are used for database connectivity, object-relational mapping, and schema changes that are executed during Composer software upgrades. The database administrator can use well-known tools and procedures for backing up and restoring the entire metadata repository. The administrator is responsible for setting up authenticated access from the Composer server to the database.

# Security

Composer provides robust data security that ensures the "three As" of security -- proper authentication, authorization, and accounting of the visual analytics environment. Additionally, its architecture provides inherent data security.

# Authentication

Administrators can manage access to the application by creating user accounts in Composer, or by synchronizing with an authentication identity provider (idP) to take advantage of centralized user management and authentication.

Composer adheres to standards-based methods for defining and enforcing security. This includes Trusted Access, a default in-house developed security methodology that allows for machine-to-machine authorization of Composer resources when embedded in your application, using delegated authorization.

Other supported standard authentication protocols include Kerberos (SPNEGO), X.509, and SAML2 for single sign on, along with plug-ins for LDAP and SAML2 IdPs to facilitate user and permissions verification. Where available, Composer can authenticate as a microservice using Kerberos or LDAP on connections to data sources.

# Authorization

Composer's authorization security model allows administrators to configure user access to data sources, attributes, and records. Fine-grained access control is configured at the group level with permissions passed via inheritance to the groups members (users).

For data sources that support delegation, pass user credentials as a connection parameter. When enabled, the database authorization polices are enforced on queries so that they run with that user's privileges.

# Accounting

Advanced accounting permits logging of all data a user viewed while using Composer. This is performed by logging all WebSocket data transmitted to the user's browser. All user activity can be recorded in they Composer application logs in your instance.

# Inherent Data Security

Composer is inherently secure because there is no need to extract or move data out of secured platforms. Direct data connectivity, push-down processing, adaptive caching, Data Sharpening™, and standards-based authentication and authorization (including user delegation) make it possible to securely work with the most current data in your data stores. Restricting the movement of data is a critical requirement for organizations that must regulated and monitor access to sensitive information, and whose data is too big to move.

# Flexible Deployment

Composer microservices and related components can be deployed on-premise, in the cloud, or in hybrid environments, provided that the host server meets operating system and hardware requirements. Additionally, Composer can query supported data sources in the cloud or on-premises, provided network-level connectivity exists between the host and the data store. As a best practice, SSL should be used on connections between Composer and a data store, especially when traversing network enclaves.

As a Java web application, Composer can meet increasing user concurrency requirements through horizontal scaling. Horizontal scale-out is achieved by replicating Composer behind a load balancer with a common metadata store for all nodes. This configuration also provides a higher degree of application availability, as traffic will be routed to healthy nodes in the event of an application or hardware failure on another node.

As the Composer microservices architecture matures, more microservices will be deployed independently for greater availability and optimized load management.

# Composer Personas

The successful implementation and use of Composer requires the completion of tasks that require different job skills. Whether these tasks are performed by one person or by more than one person varies by organization. This topic describes the different skills needed.

- Somebody with the authorization and skills to install the Composer software as a standalone product, in a high availability environment, or behind a load balancer. This person must understand your server and database infrastructure. Typically, this is someone in Operations.
- Somebody who manages licensing of the product.
- One or more people to manage product security, including:
  - Your organization's authentication tools (X.509, SAML, Kerberos, or LDAP) and their integration into Composer
  - The Composer accounts, users, and groups required by your organization.
- One or more people to manage configuration of the product. Among other things, configuration can include tasks such as setting up logging, creating and managing custom charts and admin-defined functions, or tailoring the theme and other features of the UI as needed by your organization. Typically, this would be a system developer.
- Somebody who manages the Composer connectors and connector server definitions needed by Composer for your data stores. This person would also create the Composer connection definitions needed to access those data stores. Typically, this is a database administrator.
- Somebody to define Composer data source configurations from the data available in your data stores. This person must understand the data in your data stores and how it will be used. They must also understand who should have access to the data in each data source and whether access should be controlled by column or row. This person may be referred to as a *data author*.
- One or more people who can create meaningful dashboards and visuals that help your organization analyze the data you've collected. These people need to be able to specify who should have access to the dashboard data and the dashboard functions available for end users to interact with after the dashboard is embedded in your application. This person may be referred to as a *content author*.
- One or more people to embed your dashboards into your applications. Typically, this is an application developer.

# Composer Data Protection Policy

Composer recognizes The General Data Protection Regulation (GDPR). GDPR means the Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).

Personal Data means any information relating to (i) an identified or identifiable natural person and, (ii) an identified or identifiable legal entity (where such information is protected similarly as personal data or personally identifiable information under applicable Data Protection Laws and Regulations), where for each (i) or (ii), such data is Customer Data.

With respect to processing personal data, the Composer Customer (“Customer”) is the Controller and Composer is the Processor. The Customer shall use the Composer Managed Service (“Managed Service”) in accordance with the requirements of the relevant Data Protection Laws and Regulations. If Customer will be processing Personal Data using the Managed Service, Customer shall comply with Data Protection Laws and Regulations. Customer shall have sole responsibility for the accuracy, quality, and legality of Personal Data and the means by which Customer acquired Personal Data.

Composer shall treat Personal Data as Confidential Information and shall only process Personal Data as it relates to Customers use of the Managed Service.

Composer, to the extent legally permitted, will promptly notify Customer if Composer receives a request from a Data Subject (identified or identifiable person to whom Personal Data relates) to exercise the Data Subject's right of access, right to rectification, restriction of Processing, erasure (“right to be forgotten”), data portability, object to the Processing, or its right not to be subject to an automated individual decision making, each such request being a “Data Subject Request”. Taking into account the nature of the Managed Service, Composer will assist Customer by appropriate technical and organizational measures, insofar as this is possible, for the fulfillment of Customer’s obligation to respond to a Data Subject Request under Data Protection Laws and Regulations. In addition, to the extent Customer, in using the Managed Services, does not have the ability to address a Data Subject Request, Composer will upon Customer’s request provide commercially reasonable efforts to assist Customer in responding to such Data Subject Request, to the extent Composer is legally permitted to do so and the response to such Data Subject Request is required under Data Protection Laws and Regulations. To the extent legally permitted, Customer shall be responsible for any costs arising from the provision of such assistance.

Composer shall maintain appropriate technical and organizational measures for protection of the security (including protection against unauthorized or unlawful Processing and against accidental or unlawful destruction, loss or alteration or damage, unauthorized disclosure of, or access to, Customer Data), confidentiality and integrity of Customer Data. Composer regularly monitors compliance with these measures.