TOC

Logi Symphony 24

Introduction to Symphony 24

Symphony is an embeddable business intelligence platform for data exploration, visual analytics, and creating & sharing dashboards, reports, and more. You can deploy it as the central data portal for your organization, or integrate it into an existing website as part of a custom or embedded BI solution. Symphony is flexible enough to adapt to your needs and to every type of user.

Symphony delivers the first out-of-the-box development experience for embedded analytics. Choose Symphony 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.

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

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

Effortless Authoring

Symphony'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

Self-Serve Business Intelligence

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

Symphony gives you the ability to perform data exploration and analysis, run ad hoc queries, and create your own dashboards and reports, all without having to involve your organization's technical or IT staff.

Using the interactive screens, you can easily explore available data sources, use intuitive operations to visualize your data, and arrange it in dashboard, multi-page report, scorecard, and small multiple views. Apply formulas using a familiar toolbar interface, access other analysis options with a single click, or ask the Assistant for help. Smart defaults and automatic data preparation are used throughout Symphony to provide a streamlined workflow, which means there are a lot fewer steps needed to progress from your data to desired visuals.

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. Symphony 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
- Data Discovery Security
- Data Protection Policy

Microservices Architecture

Symphony'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

Fully Open APIs

Symphony is built on a completely open API platform designed for extensibility, integration, and embedding. Every capability you see in the user interface from administration through to analysis and views can be embedded, and is powered by public .NET, REST, and JavaScript APIs, which you have access to as well. You can insert your own custom extensions into virtually any part of the workflow using the same plug-in architecture used for built-in components, including data providers, transforms, formulas, visualizations, export providers, and much more. Tailor to your needs using custom CSS and JavaScript, the familiar syntax of DundasScript in calculations and advanced formulas, or the built-in JavaScript editor with pop-up help and suggestions.

HTML5, Mobile & Touch

The client-side of theSymphony is based entirely on the latest web standards including HTML5, JavaScript, and CSS. There is no requirement for browser plug-ins, or for users to install software. This means Symphony works and looks the same on a desktop as it does on a mobile device such as a tablet or smartphone. The user interface fully supports touch-based gestures so you can analyze data or design dashboards and reports directly on an iPad, or administer from your phone. All you need is a standards-compliant browser for your device.

Data Visualizations

Symphony provides a wide array of data visualization types you can use to analyze and view your data, including a complete chart library, maps, tables, gauges, relationship diagrams and more. Visualizations are highly interactive and responsive, with intuitive animated transitions, and with performance and support for ad hoc analysis in mind. We've programmed our visualizations with styles and behavior that automatically conform to established best practices in data visualization, and smart visualizations can make choices and recommendations for you as you work with your data, including one-click advanced analytics. Each visualization is rich with features and customization options for optimal gaining and sharing of insights.

ETL Layer & In-Memory Analytics

Extract-Transform-Load (ETL) capability is built directly into Symphony via our data cube layer. A wide variety of transforms are available for you to add and connect together to perform data cleansing, integrate data from different sources, access machine learning and other advanced analytics with R or Python, and any other data preparation you may need. Data cubes can also store data input through a spreadsheet-like interface or through custom forms on dashboards. The output of a data cube is a reusable data model you can prepare with predefined formatting and metadata and share with others. Connect to your data live, store it as needed in our internal data warehouse for improved performance, or build it as an in-memory model to enable even faster analytical query results.

Administration & Configuration

Symphony provides a flexible data security model, which includes complete built-in support for optional multi-tenant / SaaS deployment. You can use whatever form of authentication you need for seamless integration with your own applications or systems, with a variety of standard protocols ready to use. For scalability, load balancing is supported through the use of application and data processing server groups, and containerization and Kubernetes support for scaling on cloud or server cluster deployments. Authorized users can access administrative tasks with one click from Symphony's menu, using intuitive interfaces to manage the complete set of configuration settings, logs, licensing, user account management, and other application functions, without additional tools.

Microservices Architecture

This applies to: *Visual Data Discovery*

The Symphony 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 Symphony 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 Symphony, 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, Symphony 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:

- Symphony 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 Symphony

The following links provide general information about all Symphony microservices:

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

Web-Based User Interface

This applies to: *Visual Data Discovery*

Symphony provides a single, unified web-based user interface (the Symphony 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 Symphony 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 Symphony JavaScript APIs) is responsible for establishing a two-way WebSocket communication channel between itself and the Symphony 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 Symphony.

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 Symphony 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, Symphony efficiently cancels previously running queries to conserve computing and network resources.

The value of the Symphony 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

This applies to: *Visual Data Discovery*

The Symphony query engine sits between the web application and the Symphony 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 Symphony 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

Symphony 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, Symphony 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 Symphony 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.

Symphony'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 Symphony. Symphony 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. Symphony 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.

Symphony 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 Symphony 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

Symphony's implementation of push-down processing also allows users to explore down to the atomic, row level detail. In this way, Symphony 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 Symphony 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, Symphony 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, Symphony'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.

Symphony 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, Symphony 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 Symphony cache stores all the results of aggregated requests from your data source. When a visual is created the request is first sent to the Symphony 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

This applies to: *Visual Data Discovery*

Symphony 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 Symphony provides, you can create custom data connectors.

Symphony'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, Symphony'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. Symphony data connectors are smart enough to enrich the data interaction experience beyond standard off-the-shelf functionality.

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

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

Data Writer Microservice

This applies to: *Visual Data Discovery*

Symphony 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 Symphony data sources can use the Symphony UI to upload flat files to Symphony. 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 Symphony Data Writer microservice can receive live streaming data via the Symphony 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.

Symphony 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

Symphony 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

This applies to: *Visual Data Discovery*

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 Symphony installation, so it must be installed and configured before you can use it. Using the Service Monitor, you can:

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

For complete information, see Symphony Data Discovery Service Monitor.

Configuration Microservice

This applies to: *Visual Data Discovery*

The Symphony configuration microservice is packaged with the Spring Cloud Configuration server, which allows Symphony to easily integrate with its Spring-based microservices. It provides the mechanism by which Symphony 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 Symphony servers to the Symphony 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 Symphony Data Discovery Configuration Microservice To Maintain Application Properties.

Service Discovery Microservice

This applies to: *Visual Data Discovery*

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

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

Screenshot Microservice

This applies to: *Visual Data Discovery*

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

Tracing Microservice

This applies to: *Visual Data Discovery*

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

Distributed Environment Support

This applies to: *Visual Data Discovery*

You can deploy Symphony 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 Symphony environment:

- **Load balancing:** Load balancing helps you scale Symphony 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 Symphony Server Behind A Load Balancer.

- **High Availability:** A high availability environment includes multiple Symphony 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 Symphony 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 Symphony nodes. If only a single load balancer is deployed in a high availability environment, you will not be able to access any of the Symphony nodes behind it if the load balancer should fail. Microservice load balancing and failover occur automatically within the Symphony nodes themselves.

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 Symphony servers to the Symphony 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

This applies to: *Visual Data Discovery*

Data queries in Symphony use data source configurations to identify the data needed and the data store from which it should be obtained. A Symphony 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 Symphony 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, Symphony 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

This applies to: *Visual Data Discovery*

You can visualize data from your data stores in visuals and dashboards in the Symphony 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 Symphony environments running the same version of Symphony. See Manage Dashboards and Manage Visuals.

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

Multisource Analysis

This applies to: *Visual Data Discovery*

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.

Symphony dashboards can contain many visuals, and each visual can depict data from a different data store. In addition, Symphony 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

This applies to: *Visual Data Discovery*

There is one common attribute to all data streams: time. Any data source that has a date-time field can be playable in the Symphony 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, Symphony 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. Symphony'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, Symphony 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

This applies to: *Visual Data Discovery*

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

There are several options for embedding analytics:

- White-label for rebranding Symphony to display another organization's logo, fonts, colors, etc. See Product Customizations.
- Data integration using the Symphony 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 Symphony UI and is based on the filters applied to a Symphony 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 Symphony 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.

Symphony 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 Symphony installation at this link: `<symphony-URL>/discovery/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

This applies to: *Visual Data Discovery*

You can customize Symphony in the following ways.

Customization	Description
Custom charts	Symphony 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 Symphony dashboards. Custom charts make use of the same query engine that powers standard Symphony visuals. Symphony data connectors, microqueries with Data Sharpening, and live mode are all available with custom charts. Custom charts are built using the Symphony 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. 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. 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. 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. See Control How Users Interact With a Visual.
Admin-defined functions	Symphony 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 Symphony equivalent) to extend your data analytics in Symphony. Examples of this might be functions available within your data store either natively or as user-defined functions. See Admin-Defined Functions.

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

Metadata Repository

This applies to: *Visual Data Discovery*

Metadata is any data that is used for the configuration and runtime operation of the Symphony application, with the exception of customer-managed data stores. The metadata repository is simply the relational database instance containing the Symphony 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 Symphony application; they are not intended for direct manipulation.

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

Important: Symphony 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 Symphony (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 Symphony 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 Symphony server to the database.

Data Protection Policy

This applies to: *Visual Data Discovery*

Symphony 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 Symphony Customer ("Customer") is the Controller and Symphony is the Processor. The Customer shall use the Symphony 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.

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

Symphony, to the extent legally permitted, will promptly notify Customer if Symphony 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, Symphony 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, Symphony will upon Customer's request provide commercially reasonable efforts to assist Customer in responding to such Data Subject Request, to the extent Symphony 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.

Symphony 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. Symphony regularly monitors compliance with these measures.

Add an SSL Certificate

This applies to: *Visual Data Discovery*

Symphony supports SSL certificates so that a secure connection between the Symphony server and the browser can be established. In particular, Symphony supports two common formats for SSL certificates - JKS and PKCS12. For information on creating a keystore or Certificate Signing Request (CSR) for use with Symphony, see *https://www.digicert.com/kb/code-signing/java-code-signing-guide.htm*.

To enable HTTPS and a secure browser connection, the SSL certificate needs to be copied into the appropriate Symphony directory and the proper parameters be added to the `zoomdata.properties` configuration file.

Perform the following steps:

1. From your terminal, SSH to your Symphony server.
2. Stop Symphony microservices. See Stop Symphony Microservices.
3. Copy your SSL keystore file to the `/etc/zoomdata` or `/opt/zoomdata/conf` directory. To obtain the SSL keystore file, work with your website domain provider.
4. Use the following command to access and open the `zoomdata.properties` file:

   ```
   vi /etc/zoomdata/zoomdata.properties
   ```

   If the `.properties` file does not exist, this command will create the file.
5. Add the following lines to the `zoomdata.properties` file:

   ```
   server.port=8443
   server.ssl.enabled=true
   server.ssl.key-store=/etc/zoomdata/<keystore_name>
   server.ssl.key-store-password=<your_keystore_password>
   ```

   Replace the placeholders `<keystore_name>` and `<your_keystore_password>` with your keystore details. Symphony supports the JKS and PKCS12 certificate formats.
6. Save and exit the `.properties` file.
7. Start Symphony microservices. See Start Symphony Microservices.

After the Symphony server has successfully restarted, open a new browser and check for a secure connection (that is, HTTPS).

Revert the SSL Certificate to the Default Version

If you need to remove the SSL certificate, you must edit the `zoomdata.properties` file so that the SSL certificate is reverted back to the default version. Edit the `zoomdata.properties` file as follows:

```
server.ssl.key-store=/opt/zoomdata/conf/keystore
keystorePass=changeit
```

Remember to save and exit the `.properties` file. Then restart Symphony microservices. See Restart Symphony Microservices.

This reverts the SSL connection to the original self-signed certificate preinstalled with Symphony.

Disable the SSL Certificate in Symphony Data Discovery

This applies to: *Visual Data Discovery*

You can disable the SSL certificate in Symphony by adding a parameter to the `zoomdata.properties` file located in the `/etc/zoomdata` directory. The purpose of the parameter is to disable a redirect by Spring Boot to SSL and enable you to use the HTTP port. Take the following steps to disable the SSL Certificate:

1. From your terminal, SSH to your Symphony Server.
2. Stop Symphony microservices. See Stop Symphony Microservices
3. Use the following command to access and open the `zoomdata.properties` file:

   ```
   vi /etc/zoomdata/zoomdata.properties
   ```

   If the `.properties` file does not exist, this command creates the file.
4. Add the following parameters into the file as new lines:

   ```
   server.port=8080
   server.ssl.enabled=false
   ```

5. Save and exit the `.properties` file.
6. Start Symphony services. See Start Symphony Microservices .

After the Symphony Server has successfully restarted, you can open a new browser window and log in. You should no longer be redirected to an SSL connection.

If you have configured your firewall (see next section) then use the following URL format:

```
http://<Symphony-IP-address>/discovery
```

Otherwise, use the following URL format:

```
http://<Symphony-IP-address>:8080/discovery
```

Set Up the Screenshot Microservice

This applies to: *Visual Data Discovery*

The Screenshot microservice allows you to view snapshots of your saved dashboards. Review the following topics:

- Screenshot Microservice Prerequisites
- Install The Screenshot Microservice
- Test The Screenshot Microservice
- Troubleshoot Screenshot Microservice Problems
- Screenshot Microservice Upgrade Notes

Screenshot Microservice Prerequisites

This applies to: *Visual Data Discovery*

Before you can install and use the Screenshot microservice, consider the following browser, memory, thread count, and software prerequisites.

Browser Requirements

The Screenshot microservice uses headless Google Chrome. Chrome-based screenshots include the entire dashboard and can be exported in PNG or PDF formats.

Memory Configuration Considerations

If your use of Symphony includes scheduling dashboard reports, you may need to update this memory setting. The default memory configuration is suitable for handling up to 10 concurrent dashboard reports (different dashboard reports scheduled for the same time). If you need to schedule more concurrent dashboard reports, increase the memory allocation to about one gigabyte (1GB) more for every 15 concurrent reports. See Configure Memory Settings.

Thread Count Considerations

If your use of Symphony includes scheduling dashboard reports, update the Screenshot microservice `pool.thread.size` setting so it is greater than the number of concurrent reports (see Screenshot-service.properties Properties). This setting controls the thread count for Screenshot microservice requests.

Obtain the Software

Before you can install the Screenshot microservice, contact Symphony Technical Support to obtain a download link for the Screenshot microservice installation software. Be sure you specify the operating system you are using so the appropriate software is provided.

Install the Screenshot Microservice

This applies to: *Visual Data Discovery*

Install the Screenshot microservice

1. Download the Screenshot microservice package using the link provided by Symphony Technical Support.
2. Install the software using the appropriate command below, modifying `<filename>` to match the installation package provided by Symphony Technical Support:

   In CentOS environments:

   ```
   sudo yum localinstall <filename>.rpm
   ```

   In Ubuntu environments:

   ```
   sudo dpkg -i <filename>.deb
   ```

3. Install the correct version of ChromeDriver. Refer to the ChromeDriver documentation for more information.

   Run the script `install-dependencies.sh` (located in the `/opt/zoomdata/docs/screenshot-service/` installation directory) or enter the following commands on the command line (for all operating systems):

   ```
   CHROMEDRIVER_LATEST_VERSION=$(curl -s https://chromedriver.storage.googleapis.com/LATEST_RELEASE)
   curl -s -o /tmp/chromedriver.zip \
   "https://chromedriver.storage.googleapis.com/${CHROMEDRIVER_LATEST_VERSION}/chromedriver_linux64.zip" sudo unzip
   /tmp/chromedriver.zip -d /usr/bin/
   rm -f /tmp/chromedriver.zip
   ```

4. Optionally, modify the `zoomdata.properties` file to enable and set up the Screenshot microservice. In addition to enabling the Screenshot microservice, you can also set the time period for capturing screenshots of your visuals to be displayed on the library page.

i. To create screenshots in the background, set the `screenshot.daemon.enabled` property to `true`:

ii. Specify which types of screenshots you want to enable. Set the `screenshots.dashboards.enabled` property to `true` if you want to enable capturing and displaying the screenshots for the dashboards.

When both `screenshot.daemon.enabled` and `screenshots.dashboards.enabled` properties are enabled, screenshots are created automatically when a dashboard is created or updated and at the rate specified by the `screenshot.daemon.schedule.rate` property (set in the next step of this procedure). If either the `screenshot.daemon.enabled` or `screenshots.dashboards.enabled` properties is disabled, screenshots are not created automatically, but you can still create a screenshot manually using the API.

5. Specify the frequency at which the screenshots are refreshed by configuring the property `screenshot.daemon.schedule.rate=<n>h` in `zoomdata.properties`. The default frequency is every 24 hours, but you can set your own frequency (in hours) by replacing `<n>` with your desired frequency.

6. Enable and start the Screenshot microservice. If you are using `systemctl`, run the following commands:

```
sudo systemctl enable zoomdata-screenshot-service
sudo systemctl start zoomdata-screenshot-service
```

If you are not using `systemctl`, adjust these commands accordingly. Additional information on restarting microservices is provided in Restart Symphony Microservices.

7. Watch the `/opt/zoomdata/logs/screenshot-service.log` file. The microservice is running successfully when the log displays a line similar to this:

```
"Started ScreenshotServiceApplication in 12.184 seconds"
```

Test the Screenshot Microservice

This applies to: *Visual Data Discovery*

Test the Screenshot microservice

1. Open a web browser to Symphony. Obtain the ID for a dashboard to use for testing. Log in to Symphony and open a dashboard. On the browser address bar copy the portion of the URL after the + sign. In the following example, you would copy `5ad8d1fa60b2894b38f0933b`:

   ```
   https://vm:8443/composer/visualization/5ad50156e4b07727dcb11098+5ad8d1fa60b2894b38f0933b
   ```

2. Use Postman or cURL to issue a PUT request to Symphony to request a screenshot of the dashboard in PNG format. In the following example, replace <username> with your user name, <password> with your password, <server> with your server IP address or name, <port> with your port number, and <dash-id> with the dashboard ID you obtained in the previous step.

   ```
   curl --insecure -X PUT -u <username>:<password> https://<server>:<port>/composer/api/
   screenshot/<dash-id>?"width=1000&format=PNG" > test.png
   ```

3. If the `test.png` file is created successfully, the Screenshot microservice is operating correctly.

Troubleshoot Screenshot Microservice Problems

This applies to: *Visual Data Discovery*

Common issues can be resolved by editing the Screenshot microservice properties file in `/etc/zoomdata/screenshot-service.properties` and, in some cases, the `etc/zoomdata/zoomdata.properties` file. Changing values in these files requires a restart of the associated microservice. See Restart Symphony Microservices. See also Icons Not Reverting To Defaults After Screenshot Microservice Disabled

Timeouts

Symphony and the Screenshot microservice include default timeouts for dashboards to render. If you will be requesting screenshots of dashboards that takes longer than this default, you can increase the default timeout in the properties file.

The Screenshot microservice may time out on dashboards that take too long to draw. If a selected Symphony dashboard takes more than the default timeout, then the default timeout setting in the properties file can be increased.

Determine how much additional time you need, in seconds, for the dashboards to load or render and then add properties, as described below, to the properties file.

Increase the default timeout

1. On the Symphony server, edit `/etc/zoomdata/screenshot-service.properties`.
2. Update the following properties, specifying an appropriate number of seconds for `<nnnn>`:

i. `screenshot.webdriver.timeout=<nnnn>`

ii. `export.dashboard.screenshot.timeout.seconds=<nnnn>`

3. Save the `screenshot-service.properties` file.
4. On the Symphony server, edit `/etc/zoomdata/zoomdata.properties`.
5. Update the following property, specifying an appropriate number of milliseconds for `<nnnn>`:
   `screenshot.service.http.client.read.timeout.milliseconds=<nnnn>`
6. Save the `zoomdata.properties` file.
7. Restart the Symphony and Screenshot microservices. See Restart Symphony Microservices.

Self-signed Certificate

If Symphony is running with a self-signed certificate, the Screenshot microservice must be configured to accept the lower-security certificate.

Configure the Screenshot microservice to accept the lower-security certificate:

1. On the Symphony server, edit `/etc/zoomdata/screenshot-service.properties`.
2. Update the following property as follows:

```
driver.options=--headless,--disable-gpu,--hide-scrollbars,--no-sandbox,--allow-insecure-localhost
```

3. Save the properties file.
4. Restart the Screenshot microservice. See Restart Symphony Microservices.

   This will pass the options to the ChromeDriver. The option list includes the default options normally passed to the driver, with the additional `--allow-insecure-localhost` option.

Screenshot Microservice Upgrade Notes

This applies to: *Visual Data Discovery*

After upgrading, the screenshots on the Home page may look different. This occurs if there has been a change in the screenshot aspect ratio. To make the screenshots look correct, make sure that the `screenshot.daemon.enabled` property in the `zoomdata.properties` file is set to `true`. Remember to restart the Symphony server microservice after the change (see Restart Symphony Microservices).

After you have updated the properties file, you can update the screenshots in one of the following ways:

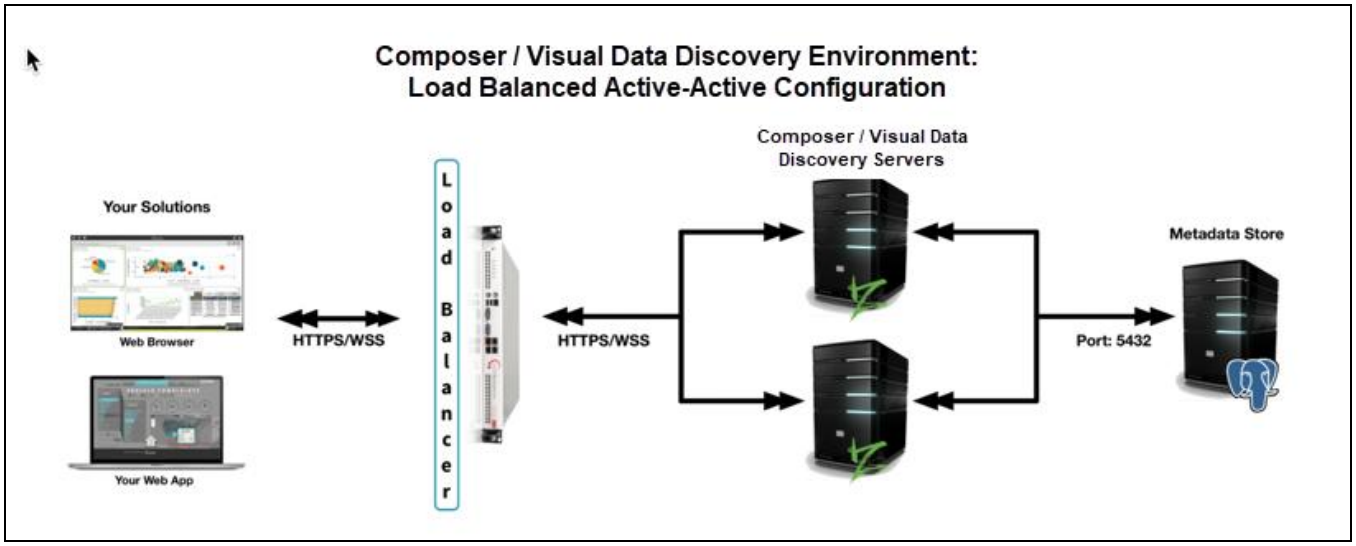
- In the `zoomdata.properties` file, modify the `screenshot.daemon.schedule.rate` property and set it to a more frequent refresh rate.
- Execute a cURL call to update a screenshot for a specific dashboard.
- Execute a cURL call to upload a custom image for a specific dashboard. Otherwise, the screenshots will be updated when the refresh screenshot procedure runs according to the configured refresh rate.

Configure the Symphony Server Behind a Load Balancer

This applies to: *Visual Data Discovery*

Load balancing helps you to scale Symphony to hundreds of users. You can use load balancing both on-premises and with cloud deployments, meaning you can use load balancing for various environments. When the server is load balanced within your network environment, it includes native SSL support and can proxy WebSocket traffic.

The following diagram depicts a classic load balancing setup.



Symphony has tested active-active load balancing configuration, which is the particular setup used in the steps below. In addition, the instructions provided below take into account that Symphony, when it runs standalone, uses its own dedicated PostgreSQL server as the metadata store (in other words, PostgreSQL was installed as part of the Symphony installation process). If you are running in a high availability environment, you will need to use a high availability (clustered) PostgreSQL data store.

Important: Configuring Symphony in a distributed environment requires a multi-node license. Be sure you have obtained this before you start. Contact your insightsoftware Technical Support representative for assistance.

Complete the following steps to configure load balancing in your environment.

Step 1: Identify or Install the Postgres Metadata Store

All Symphony installations require a Postgres metadata store. If you install Symphony on a single server using the supplied bootstrap installation script, this metadata store is installed for you. If you install Symphony manually, you must also manually set up the metadata store.

The metadata store should be centrally installed, accessible by all Symphony nodes. In a high availability environment, it can be clustered.

When setting up a distributed environment, you need to determine whether the metadata store is installed. If you have already installed Symphony and are now trying to set up a distributed environment, there is a good chance that the metadata store was installed with your existing Symphony instance or instances.

If this is the first time you have installed Symphony, and you want to set up a distributed environment, a new metadata store is required.

- If the metadata store has not already been installed, manually install it now. Complete the following sub-steps. Then identify its host name and port.

A. ***Create the Metadata Store User & Stores*** *- Necessary for all installations.*

The instructions to set up PostgreSQL as Symphony's metadata store differ depending on the Linux operating system used by the target server. Select a topic below:

- PostgreSQL Setup for CentOS Environments
- PostgreSQL Setup for Ubuntu Environments

PostgreSQL Setup for CentOS Environments

Note: New installations of Symphony (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.

- Add the PostgreSQL Yum repository to CentOS by running this command calling the appropriate PostgreSQL version:

```
sudo yum -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-
latest.noarch.rpm
```

- Install the PostgreSQL client and server packages by running these commands:

```
sudo yum -y install epel-release yum-utils
sudo yum-config-manager --enable pgdg12
sudo yum install postgresql12-server postgresql12
```

- After installation, initialize the PostgreSQL database:

```
sudo /usr/pgsql-12/bin/postgresql12-setup initdb
```

- Start and enable the PostgreSQLmicroservice:

```
sudo systemctl enable --now postgresql-12
```

- Confirm that the service started without errors:

```
sudo systemctl status postgresql-12
```

  If necessary, start it:

```
sudo systemctl start postgresql-12
```

- If you have a running firewall and remote clients should be able to connect to the PostgreSQL metadata store, modify the firewall to allow the PostgreSQL service:

```
sudo firewall-cmd --add-service=postgresql --permanent
sudo firewall-cmd --reload
```

- If the PostgreSQL database is operating in a cluster, repeat steps 3-6 for each instance of the database.

- Set up the PostgreSQL Admin user and password:

```
sudo su - postgres
~]$ psql -c "alter user postgres with password 'StrongPassword'"
ALTER ROLE
```

PostgreSQL Setup for Ubuntu Environments

Note: New installations of Symphony (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.

- If this is a new server instance, update your current system packages:

```
sudo apt update
sudo apt -y install vim bash-completion wget
sudo apt -y upgrade
```

  A reboot is necessary after an upgrade.

```
sudo reboot
```

- Import the GPG key and add the appropriate PostgreSQL version repository to your Ubuntu machine. Run the following commands:

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add
echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" |sudo tee
/etc/apt/sources.list.d/pgdg.list
```

  The added repository contains many different packages and third-party add-ons, including: `postgresql-client`, `postgresql`, `libpq-dev`, `postgresql-server-dev`, and `pgadmin packages`.

- Update the package list and install the PostgreSQL server and client packages:

```
sudo apt update
sudo apt -y install postgresql-12 postgresql-client-12
```

  The PostgreSQL microservice is started and will start with every system reboot.

- If you have a running firewall and remote clients should be able to connect to the PostgreSQL metadata store, modify the firewall to allow the PostgreSQL service port:

```
sudo ufw allow 5432/tcp
```

- Test the PostgreSQL connection.

  a. During installation, a user named `postgres` is created automatically with full superadmin access to your entire PostgreSQL instance. Before you switch to this account, your logged in system user should have sudo privileges:

  ```
  sudo su - postgres
  ```

  b. Replace the `postgres` password with a strong password:

  ```
  psql -c "alter user postgres with password 'StrongAdminP@ssw0rd'"
  ```

  c. Start PostgreSQL using this command.

  ```
  $ psql
  ```

  d. Get connection details as shown below.

  ```
  postgres=# \conninfo
  You are connected to database "postgres" as user "postgres" via socket in "/var/run/postgresql" at port "5432".
  ```

  e. Create a test database called `mytestdb` to see if everything is working.

  ```
  postgres=# CREATE DATABASE mytestdb;
  CREATE DATABASE
  postgres=# CREATE USER mytestuser WITH ENCRYPTED PASSWORD 'MyStr0ngP@SS';
  CREATE ROLE
  ```

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE mytestdb to mytestuser;
GRANT
```

You can list the created databases by running:

```
postgres=# \l
```

f. Connect to your test database.

```
postgres-# \c mytestdb
You are now connected to database "mytestdb" as user "postgres".
```

B. ***Change Metadata Store Authentication to MD5*** *-- Not necessary when installing a high availability environment in the cloud.*

If you installed Symphony's metadata store on a server running CentOS or RedHat, complete the configuration steps below. If the server is running Ubuntu, ignore these instructions.

Note: New installations of Symphony (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.

Change authentication for your metadata store to MD5

- Edit the `pg_hba.conf` file for the appropriate version of PosgreSQL.

```
sudo vi /var/lib/pgsql/12/data/pg_hba.conf
```

- Change METHOD to MD5.

```
# IPv4 local connections:
host all all 127.0.0.1/32  md5 # IPv6 local connections: host all all ::1/128       md5
```

- Restart PostgreSQL. In CentOS environments, run:

```
sudo systemctl restart postgresql-12
```

C. ***Configure the Metadata Store for SSL*** *- Not necessary if installing a high availability environment in the cloud.*

If you have specified SSL connections for the metadata store JDBC connections in `zoomdata.properties` file, the root CA certificate that is used for the PostgreSQL database must be added to the `/opt/zoomdata/.postgresql` directory. This directory does not exist by default and will need to be created. Complete the following steps.

- Change to the `/opt/zoomdata` directory as a superuser:

```
sudo cd /opt/zoomdata
```

- Create a `.postgresql` subdirectory.

```
sudo mkdir .postgresql
```

- Copy the root CA certificate for the PostgreSQL database into the new directory:

```
sudo cp root.ca /opt/zoomdata/.postgresql/root.ca
```

D. ***Configure the Metadata Store for a Distributed Environment***

The Symphony PostgreSQL data store must be configured so it is available to all Symphony instances in a distributed environment. For more information about PostgreSQL high availability clustering, see PostgreSQL documentation on high availability environments.

> **Note:** New installations of Symphony (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.

Configure the PostgreSQL data store so it is available to all instances

- Edit the `postgresql.conf` file using the appropriate version and paths:

```
vi /var/lib/pgsql/12/data/postgresql.conf
```

- Set the following property in `postgresql.conf` and save the file.

```
listen_address='*'
```

- Edit the `pg_hba.conf file`:

```
vi /var/lib/pgsql/12/data/pg_hba.conf
```

- Add the following to the `pg_hba.conf file`:

```
# TYPE DATABASE USER ADDRESS METHOD
# IPv4 local connections
host all all 0.0.0.0 /0 md5
```

- Save the `pg_hba.conf` file.
- Restart the PostgreSQL service:

```
sudo service postgresql-12 restart
```

Step 2. Install Symphony On Your Distributed Servers

Intra-service communication must be enabled by making any necessary networking (ports) and firewall changes.

Deploy multiple Symphony nodes (or instances) in a distributed environment, complete the following steps for each instance

1. For each instance, run the following commands to set up the environment variables for the installation:

```
export ZOOMDATA_POSTGRES_HOST=<postgres-host>
export ZOOMDATA_POSTGRES_PORT=<postgres-port>
export ZOOMDATA_POSTGRES_USER=<postgres-db-username>
export ZOOMDATA_POSTGRES_PASS=<postgres-db-password>
```

   where:

i. `<postgres-host>` and `<postgres-port>` are the host name and port number of the Symphony PostgreSQl metadata store

ii. `<postgres-db-username>` and `<postgres-db-password>` are the user name and password required to access the Symphony PostgreSQL metadata store.

2. In each instance, run the following command to set up the environment variables for specific enterprise data connector (EDC) packages:

```
export ZOOMDATA_EDC_PACKAGES='<edc>,<edc>[,<edc>]...'
```

   where `<edc>` is the name of the data connector you'd like to install. You must install the PostgreSQL connector because it connects to the metadata store. Data connector names are the same as their connector microservice names without the `zoomdata-edc-` prefix. See *Composer V24 Data Connector Reference*.

3. Run the bootstrap installation script after exporting the environment variables in the previous steps.

```
curl -O https://composer-repo.logianalytics.com/<v.r>/bootstrap-zoomdata.run
sudo -E /bin/sh bootstrap-zoomdata.run
```

   where `<v.r>` is the Symphony version and release.

4. After the installation, ensure that the following property files are correctly set up on each node. Add or update the properties as necessary. In each, the IP address, port, user name, and password of the PostgreSQL metadata store should be specified.

   In the `zoomdata.properties` file:

```
spring.datasource.url=jdbc:postgresql://<IP-address>:<port>/zoomdata
spring.datasource.username=<db-username>
spring.datasource.password=<db-password>
```

```
keyset.destination.params.jdbc_url=jdbc:postgresql://<IP-address>:<port>/zoomdata-keyset
keyset.destination.params.user_name=<db-username>
keyset.destination.params.password=<db-password>
keyset.destination.schema=public
upload.destination.params.jdbc_url=jdbc:postgresql://<IP-address>:<port>/zoomdata-upload
upload.destination.params.user_name=<db-username>
upload.destination.params.password=<db-password>
```

In the `query-engine.properties` file:

```
spring.qe.datasource.jdbcUrl=jdbc:postgresql://<IP-address>:<port>/zoomdata-qe
spring.qe.datasource.username=<db-username>
spring.qe.datasource.password=<db-password>
```

5. Ensure that port 8080 is open on all your back-end servers to support load balancing. If not, run the following command:

```
sudo iptables -I INPUT 1 -p tcp --dport 8080 -j ACCEPT
sudo service iptables save
```

6. Repeat these steps for every Symphony instance (node) in your Symphony cluster.

Step 3. Set Up a Load Balancer

Set up one or more load balancers in your environment. For a simple load balanced configuration, only one is needed. In a high availability configuration, however, more than one load balancer is recommended; if only a single load balancer is deployed in a high availability environment, you will not be able to access any of the Symphony nodes behind it if the load balancer should fail.

The following steps provide an example of setting up HAProxy as a load balancer. Regardless of what kind of load balancer you use, ensure that port 443 is open on it.

Set up an HAProxy load balancer

1. On your machine, run the following command to install HAProxy:

```
sudo yum install haproxy
```

2. Navigate to the HAProxy folder.

```
cd /etc/haproxy
```

3. Create a certificate or copy an existing certificate to the `/etc/haproxy` folder. If you need to create a certificate, run the following commands:

```
sudo openssl genrsa -out ca.key 1024
sudo openssl req -new -key ca.key -out ca.csr
sudo openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
sudo vi cert.pem #Create and save an empty file
sudo chmod a+w cert.pem
sudo cat ca.key ca.crt > cert.pem
```

4. In the same folder, replace the contents of the `haproxy.cfg` file with the contents of the Symphony haproxy configuration file. In the file, replace the `<node1-ip>` and `<node2-ip>` with the IP addresses of your servers. If you have more than two servers, add additional lines for each server.
5. Save your changes and exit the file.
6. Start the HAProxy microservice

```
sudo service haproxy restart
```

7. Use the following command to configure the HAProxy microservice to start automatically in CentOS environments:

```
sudo systemctl enable haproxy
```

8. Ensure that port 443 is open on your load balancer. If not, run the following command:

```
sudo iptables -I INPUT 1 -p tcp --dport 443 -j ACCEPT
sudo service iptables save
```

For assistance with configuring SAML for use with HAProxy, contact insightsoftware Technical Support.

Step 4. Copy Specialized Files

If you have any specialized files for Symphony, such as vocabulary files, manually copy them to each instance of Symphony in your distributed environment.

Back Up the Metadata Store

This applies to: *Visual Data Discovery*

Before upgrading your Symphony server, insightsoftware recommends that you back up your PostgreSQL metadata store. The metadata includes refresh schedule data, object information for your environment (such as sources, dashboards, and visual definitions), and aggregated result sets. After the metadata store is backed up, perform the upgrade. If problems arise, you can restore the metadata store from your backup copy, if necessary.

This topic describes how to back up the metadata store. For information about restoring the metadata store, see Restore The Metadata From The Metadata Store Backup.

Back up the metadata store

1. From your terminal, SSH to your server.
2. Stop all microservices. For appropriate commands based on your operating system, see Stop Symphony Microservices.
3. Navigate to the `/etc/zoomdata` directory and create a backup folder:

```
mkdir backups
```

4. Navigate to the backups directory.
5. Perform an SQL dump of the databases by entering the following commands:

```
sudo -u postgres pg_dump zoomdata > zoomdata
sudo -u postgres pg_dump zoomdata-upload > zoomdata-upload
sudo -u postgres pg_dump zoomdata-keyset > zoomdata-keyset
sudo -u postgres pg_dump zoomdata-qe > zoomdata-qe
```

6. Restart all microservices. For appropriate commands based on your OS, see Start Symphony Microservices.

For more information on backup and restore processes for PostgreSQL, refer to the PostgreSQL documentation.

Restore the Metadata From the Metadata Store Backup

This applies to: *Visual Data Discovery*

This topic describes how to restore the metadata store from a backup copy. For information about backing up the metadata store, see Back Up The Metadata Store.

Prior to restoring the metadata, ensure that the target PostgreSQL data store is clean.

Restore the metadata store

1. From your terminal, SSH to your Symphony server.
2. Stop all Symphony microservices. For appropriate commands based on your OS, see Stop Symphony Microservices.
3. Navigate to your backup directory and enter the following commands:

```
sudo -u postgres psql zoomdata < zoomdata
sudo -u postgres psql zoomdata-upload < zoomdata-upload
sudo -u postgres psql zoomdata-keyset < zoomdata-keyset
sudo -u postgres psql zoomdata-qe < zoomdata-qe
```

4. Restart all Symphony microservices. For appropriate commands based on your OS, see Restart Symphony Microservices.

For more information on backup and restore processes for PostgreSQL, refer to the PostgreSQL documentation.

Symphony Data Discovery Service Monitor

This applies to: *Visual Data Discovery*

Symphony includes a Service Monitor microservice, which can be used to:

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

Note: Symphony recommends that you contact Symphony Support for assistance with the installation and use of the Service Monitor.

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

Prerequisites

- Symphony and its microservices must be version 3.5 or later.
- All microservices must have service discovery enabled.
- The Symphony Service Monitor must be manually installed, configured and started.

The following topics provide complete information about the Symphony Service Monitor:

- Install And Configure The Symphony Service Monitor
- Access The Symphony Service Monitor
- Service Monitor Views
- Download The Diagnostics Bundle
- Maintain Application Properties

Install and Configure the Symphony Service Monitor

This applies to: *Visual Data Discovery*

The Symphony Service Monitor microservice is not installed as part of a default Symphony installation. The microservice name is `zoomdata-admin-server`.

Install, configure, and start the Service Monitor

1. Open your SSH client.
2. Use the following command to install the Symphony Service Monitor in a CentOS environment:

   ```
   sudo yum install zoomdata-admin-server -y
   ```

   Use the following command to install the Symphony Service Monitor in an Ubuntu environment:

   ```
   sudo apt-get install zoomdata-admin-server
   ```

3. After the Service Monitor is installed, you must specify a user name and password in its properties file. The properties file is called `admin-server.properties` and can be found in the `/etc/zoomdata/` directory (Linux) or the `<install-path>/conf-modify/` directory (Windows). If the properties file is not there, create it. The properties that must be defined are:

i. `monitor.user.name=<username>`

ii. `monitor.user.password=<password>`

   Edit the properties file with a text editor and substitute a Service Monitor user name for `<username>` and its associated password for `<password>`. The user name and password can be any user name and password you want.

   When you have finished, save the file.

4. Add the following properties to the `zoomdata.properties` file, located in the `/etc/zoomdata` directory (Linux) or the `<install-path>/conf-modify/` (Windows). These properties ensure that the Service Monitor has access to the Symphony server actuator endpoints.

i. `actuator.user.name=<Composer-admin-username>`

ii. `actuator.user.password=<Composer-pswd>`

iii. `actuator.logging.external-file=<log-file-path>`

Edit the properties file and substitute the valid user name and password of a Symphony administrator for `<Composer-admin-username>` and `<Composer-pswd>`. If the default Symphony log file path is not used for your installation, substitute your custom log file path for `<log-file-path>`. The default log file path is `/opt/zoomdata/logs/zoomdata.log` for Linux and `<install-path>/logs/zoomdata.log` for Windows.

> **Note:** Setting these properties exposes valid Symphony credentials as plain text in both the properties file and as tags in the Symphony Consul. Anyone in your network with the ability to communicate directly with the Consul API or view the Consul UI will be able to see these values.

When finished, save the file.

5. Start the microservice. For example, use the following command to start the Symphony Service Monitor using `systemd` in a CentOS or Ubuntu environment:

```
sudo systemctl start zoomdata-admin-server
```

See also Start Symphony Microservices.

Access the Symphony Service Monitor

This applies to: *Visual Data Discovery*

After installing the Service Monitor and restarting its microservice, you can navigate to its UI in a web browser using default port 8050. For example, if running locally, the Symphony instance would be **http://localhost:8080**, so the Service Monitor URL would be **http://localhost:8050**. If your Symphony instance is IP address URL 10.2.3.24, the Service Monitor URL would be **http://10.2.3.24:8050**.

The user ID and password you should use to log into the Service Monitor are defined in properties you set up during the Service Monitor installation. See Install And Configure The Symphony Service Monitor.

Service Monitor Views

This applies to: *Visual Data Discovery*

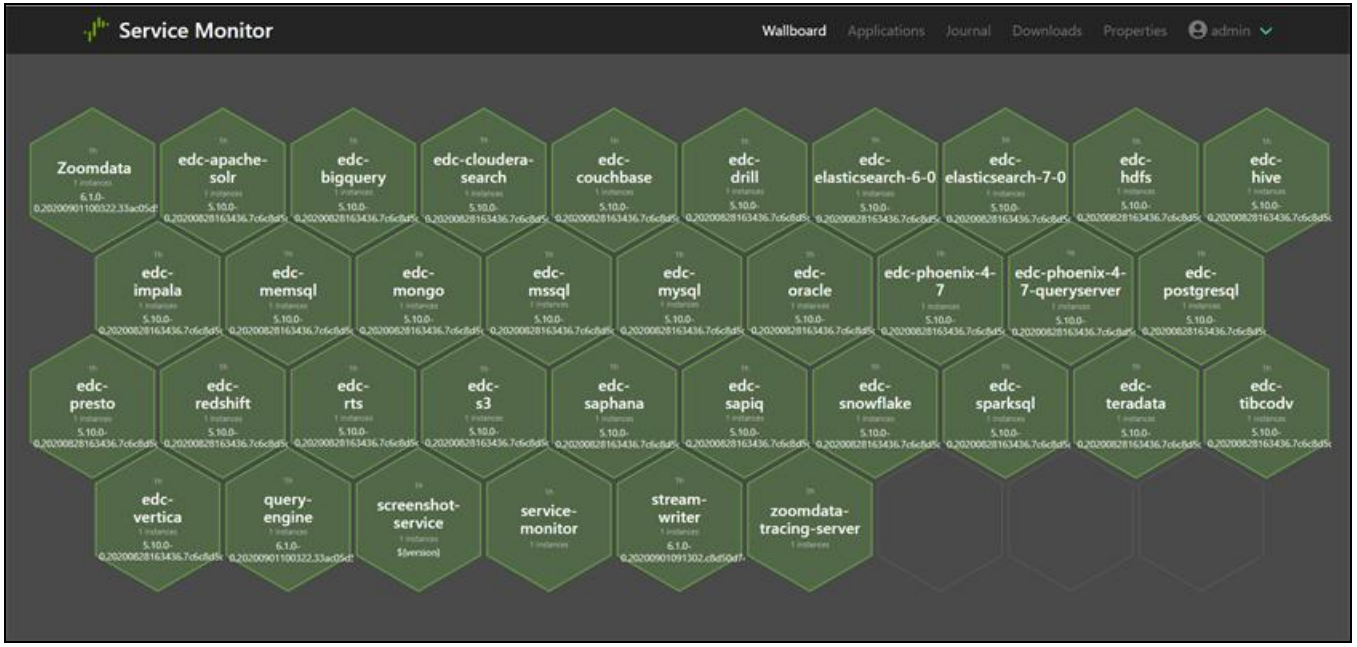
The Symphony Service Monitor UI includes the following views, each accessible via a menu option in the main menu on the Service Monitor UI page:

Each view is described below.

- Wallboard View
- Applications View
- Journal View
- Downloads View
- Properties View

Wallboard View

The Wallboard view provides shows all the Symphony microservice types used in your installation. You can access it by selecting the **Wallboard** menu option.

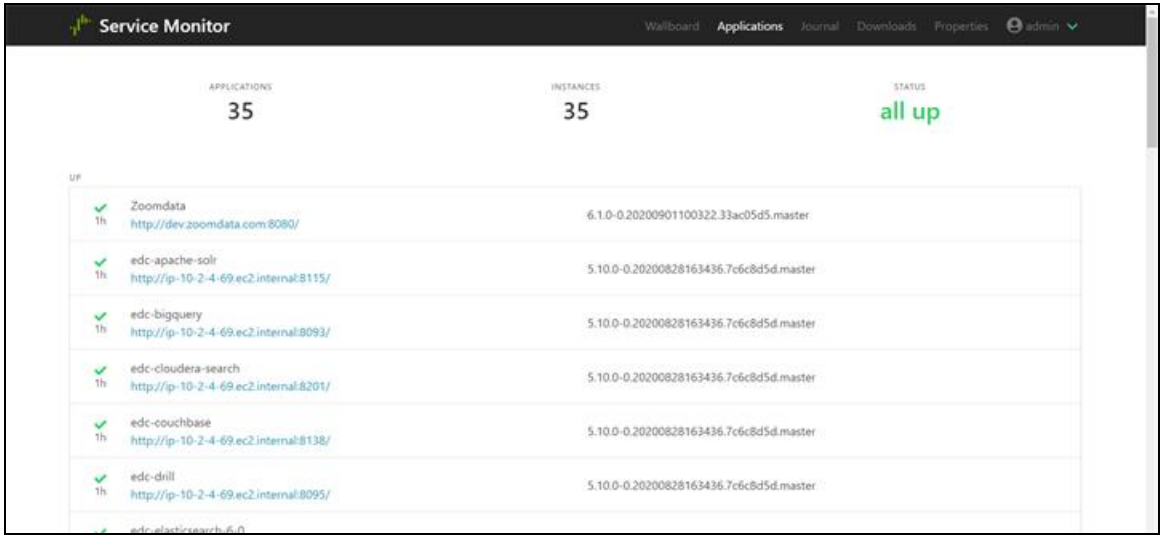


The Wallboard view shows how many instances of a Symphony microservice are running and the length of time they have been running.

Select a microservice type to obtain detailed information about it. The detailed information available varies based on the microservice type, but may include metrics, health, environment, configuration properties, scheduled tasks, logging threads, audit log, and web mappings and HTTP traces for the microservice type.

Applications View

Select **Applications** to access the Applications view.

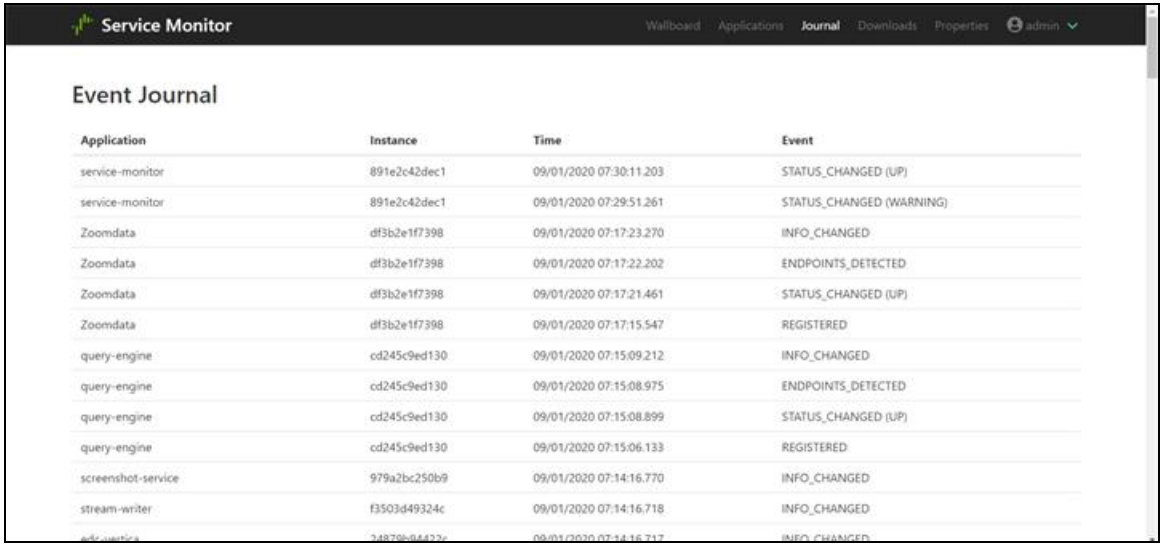


The Applications view shows all the Symphony microservice types used in your installation. It also identifies the instance URLs, and indicates how many instances there are of each and how long they have been running. If more than one instance of a microservice is running, you can expand the microservice type to see the specific instance URLs.

If you select the URL for a Symphony microservice instance, you will launch the Symphony UI for that instance. If you select the URL for a Service Monitor instance, you will launch the Service Monitor for that instance. If you select the URL for an instance of any other microservice, a page of metrics and other information appears for that instance of the microservice.

Journal View

Select **Journal** to access the Journal view.



The Journal view allows you to review the journal entries for each Symphony microservice type.

Downloads View

Select **Downloads** to access the Downloads view.

The Downloads view can be used to collect a diagnostics bundle for you to send to Symphony Support when necessary. See Download The Diagnostics Bundle for more information.

Properties View

Select **Properties** to access the Properties view.

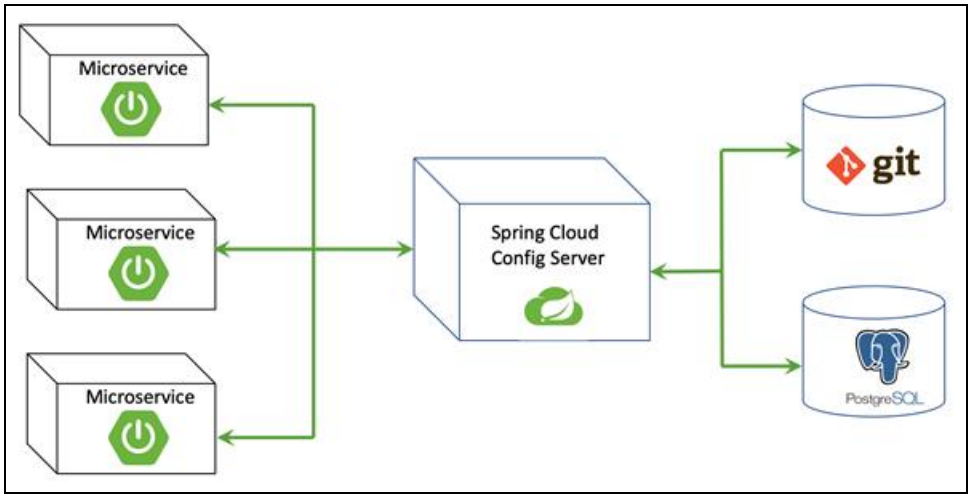
This view allows you to review and maintain the properties for each of the other Symphony microservices. It requires that the Symphony configuration microservice be installed and started first. See Use The Symphony Data Discovery Configuration Microservice To Maintain Application Properties.

Use the Symphony Data Discovery Configuration Microservice to Maintain Application Properties

This applies to: *Visual Data Discovery*

You can use the Service Monitor to review and maintain the properties for each of the other Symphony microservices. It provides a centralized location where Symphony configuration properties can be maintained. However, it requires that the Symphony configuration microservice be configured and started first.

The Symphony configuration microservice is packaged with the Spring Cloud Configuration server, which allows Symphony to easily integrate with its Spring-based microservices and provides the mechanism by which Symphony property settings can be persisted in a supported PostgreSQL metastore or in a GitHub repository. The following diagram depicts this relationship.



After the Symphony configuration microservice is installed and started, the properties can be maintained on the Service Monitor's Properties tab.

If you have the configuration microservice configured and running in a high availability environment, 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 Symphony servers to the Symphony 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.

See the following topics:

- Set Up The Configuration Microservice Metadata Store Or Repository
- Configure And Start The Configuration Microservice
- Maintain Application Properties

Set Up the Configuration Microservice Metadata Store or Repository

This applies to: *Visual Data Discovery*

Before you can install and start Symphony's configuration microservice, you must set up a PostgreSQL metastore or a GitHub repository to store Symphony property metadata. A separate PostgreSQL database or a separate GitHub repository must be configured.

PostgreSQL Database Setup Notes

If you elect to persist property metadata to a PostgreSQL metastore:

1. Configure a separate database and make it accessible to the connection user account:

   ```
   CREATE DATABASE <composer-config> WITH OWNER <db_username>;
   ```

   where `<composer-config>` is the name of the PostgreSQL database and `<db_username>` is the connection user account name.

2. Add the following properties to the Symphony`config-server.properties` file, located in the `/etc/zoomdata` directory:

   ```
   # metadata storage settings
   spring.datasource.url=jdbc:postgresql://localhost:5432/<composer-config>
   spring.datasource.username=<db_username>
   spring.datasource.password=<db_password>
   ```

   Substitute the connection user account name and password you set up in Step 1 for `<db_username>` and `<db_password>`. Substitute the name of the PostgreSQL database for `<composer-config>`.

3. Save the properties file. You will restart the configuration microservice when you configure it. See Configure And Start The Configuration Microservice.

GitHub Repository Setup Notes

If you elect to persist property metadata to a GitHub repository:

1. Add the following properties to the Symphony`config-server.properties` file, located in the `/etc/zoomdata` directory:

   ```
   # metadata storage settings
   spring.cloud.config.server.git.uri=<repo_uri>
   ```

```
spring.cloud.config.server.git.skipSslValidation=true
spring.cloud.config.server.git.username=<repo_username>
spring.cloud.config.server.git.password=<repo_password>
```

Substitute the repository user account name and password for `<repo_username>` and `<repo_password>`. Substitute the URI of the repository for `<repo_uri>` (for example, `https://example.com/my/repo`).

Additional and advanced configuration information can be found in Spring.io's documentation.

2. Save the properties file. You will restart the configuration microservice when you configure it. See Configure And Start The Configuration Microservice.

Configure and Start the Configuration Microservice

This applies to: *Visual Data Discovery*

Install, configure, and start the Symphony configuration microservice

1. Verify that you have set up a PostgreSQL metastore or a GitHub repository to store the property metadata. See Set Up The Configuration Microservice Metadata Store Or Repository.

2. Open the SSH client associated with your Symphony instance.

3. Configure all installed and enabled Symphony microservices to use the Symphony configuration microservice. Run the following script:

```
for i in $(systemctl list-unit-files | grep zoomdata | grep enabled | awk '{print $1}'|sed -e
's/\.service/.properties/g' -e 's/zoomdata\-//g');
do
echo "config-server.enabled=true">>/etc/zoomdata/$i;
done
```

4. Each Symphony microservice supports two configuration properties related to the configuration microservice:

i. `config-server.enabled`: Enables or disables integration with the configuration microservice. Valid values are `true` (enable integration) and `false` (disable integration). The default is `true`.

ii. `config-client.retry.max-attempts`: Sets the maximum number of attempts that should be made to connect to the configuration microservice. The default is 20. The Symphony microservice will fail if the number of attempts to connect to the configuration microservice exceeds this value. This property is useful in situations where the configuration microservice starts with a delay. If your Symphony microservice fails while waiting for the configuration microservice and you want to give it more time, increase this value.

   Update these properties, as appropriate, for each microservice.

5. Install, enable and start the Symphony configuration microservice. Enter the following commands:

```
sudo yum install zoomdata-config-server \
&& systemctl enable zoomdata-config-server \
&& systemctl start zoomdata-config-server
```

> **Note:** After starting the configuration microservice with a valid database configuration, the microservice should connect to the database and create a properties table.

6. Restart all the other Symphony microservices. Enter the following command:

```
sudo systemctl restart $(systemctl list-unit-files | grep zoomdata | grep enabled | awk '{print $1}')
```

See also Restart Symphony Microservices.

Maintain Application Properties

This applies to: *Visual Data Discovery*

After the Symphony configuration microservice has been configured and started, you can use the Properties tab in the Service Monitor to maintain the configuration properties of Symphony's other microservices. For information about specific Symphony properties and property files, see Configuration Property Files.

Maintain configuration properties using the Service Monitor

1. Using a web browser, navigate to the Service Monitor for the Symphony instance. Its default port is 8050. For example, if running locally, the Symphony instance would be **http://localhost:8080**, so the Service Monitor URL would be **http://localhost:8050**. If your Symphony instance is at IP address 10.2.3.24, the Service Monitor URL would be **http://10.2.3.24:8050**.

   A login screen will appear.

2. Log into the Service Monitor using the Service Monitor user name and password you defined when the Service Monitor was installed. See Install And Configure The Symphony Service Monitor.

3. Select **Properties** on the main menu bar.

   The Properties page appears.

4. Select a Symphony microservice in the **Service** drop-down list. The properties for the microservice are listed. The screenshot above lists all the properties in the `zoomdata.properties` file used by the `zoomdata` microservice.

   See ComposerSymphony Data Discovery Microservice Name Reference for a list of Symphony's microservices. If a microservice is not listed in the **Service** drop-down list, that Symphony microservice is not yet set up to use the configuration server or the microservice is down.

5. Locate the microservice property in the list that you want to change.

i. The list of properties can be sorted alphabetically by key or value. Select the **Key** or **Value** list heading to sort the properties in ascending or descending order.

ii. You can locate a property in the list by typing all or part of its name in the **Search** box at the top of the page.

6. Change the value of the property in the **Value** box associated with the microservice property you located.

   Note: Be careful changing properties. Some properties should not be changed, except with the help of Symphony Technical Support. The results of property changes could be unpredictable, if they are not made correctly.

See Configuration Property Files for a list of the property files and links to descriptions of the properties for which changes are approved.

7. Select **Save Properties** to save your property changes.

8. Select **Apply Updates** to apply the property changes to your running Symphony instance. Symphony dynamically stops and restarts the appropriate Symphony microservices.

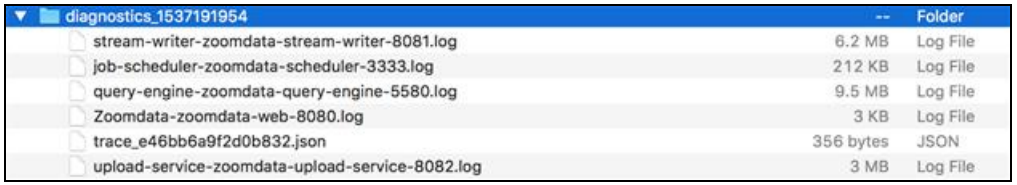
Diagnose Problems

This applies to: *Visual Data Discovery*

Symphony provides a "one-click" diagnostics bundle for the Symphony platform as part of the Symphony Service Monitor.

The diagnostics bundle is primarily useful as an easy way to send Symphony the information needed to help you debug a problem. The bundle is a zip file containing all the current log files from all Symphony's microservices. If distributed tracing is enabled and properly configured, the diagnostics bundle may also contain a JSON trace file with the trace ID you specify.

An example of the uncompressed contents of the diagnostic bundle might look like this:



Prerequisites

Before you can download the diagnostics bundle, the following conditions must be met:

- Symphony and its microservices must be version 3.5 or later.
- All microservices must have service discovery enabled.
- The Symphony Service Monitor must be manually installed, configured and started. See Install And Configure The Symphony Service Monitor.

After the Service Monitor is installed, you can download the diagnostics bundle. See Download The Diagnostics Bundle.

Download the Diagnostics Bundle

This applies to: *Visual Data Discovery*

After the Symphony Service Monitor microservice is installed and running, you can use it to download a diagnostic bundle. If you want to include trace details in the diagnostics bundle, be sure to collect tracing information as described in Distributed Tracing For Symphony.

Download a diagnostics bundle

1. Using a web browser, navigate to the Service Monitor. Its default port is 8050. For example, if running locally, the Symphony instance would be **http://localhost:8080**, so the Service Monitor URL would be **http://localhost:8050**. If your Symphony instance is at port 10.2.3.24, the Service Monitor URL would be **http://10.2.3.24:8050**.

   A login screen will appear.

2. Log into the Service Monitor using the Service Monitor user name and password you defined when the Service Monitor was installed. See Install And Configure The Symphony Service Monitor.

3. Select **Downloads** on the main menu bar.

   A screen appears that you can use to download the diagnostics bundle:

4. Optionally enter a trace ID in the Trace ID box.

   If the tracing service is not configured, the Trace ID box will be disabled, with a relevant error message. When this happens, the diagnostic bundle contains only log files.

5. Select **Download Diagnostic Bundle**.

   If you did not specify a trace ID, the zip file that is produced contains only log files for all configured Symphony microservices. If you specify a valid trace ID, the trace JSON file will also be included as part of the bundle.

   If you specify an invalid trace ID, a relevant error message displays.

In some cases, a previously valid trace ID may be reported as not found. Traces are best retrieved as soon as an error is encountered.

For more information about distributed tracing microservices, see Distributed Tracing For Symphony.

Distributed Tracing for Symphony

This applies to: *Visual Data Discovery*

All Symphony microservices support distributed tracing using the OpenTelemetry (OTel) specifications. Integrate with OTel javaagent or any other OTel compliant agent. Install a tracing server of your choice, an OTel collector, and configure the microservices to trace Symphony front end and back end requests.

Tracing collects information about requests. Use it to associate logs of the other microservices with the collected information.

Install a Tracing Server

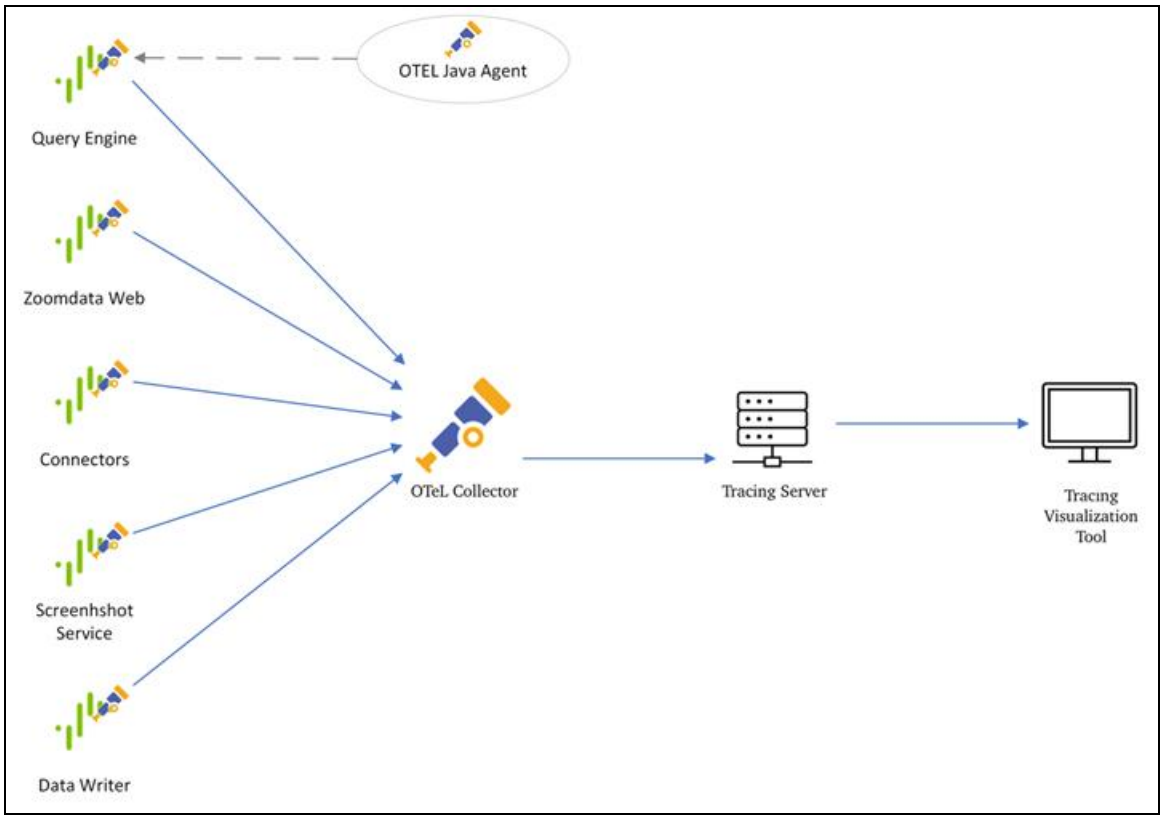
Symphony integrates with any tracing server that supports OpenTelemetry (OTel) specifications and its export protocols (otlp, zipkin, jeager).

After you have installed a tracing server, install the otel collector.

Install and Configure OTel Collector

Install an OTel collector to act as a proxy between Symphony microservices and your tracing server, both to simplify configuration and improve security by passing information through the proxy. See https://opentelemetry.io/docs/collector/ and https://opentelemetry.io/docs/collector/getting-started/.

After installation, configure your collector or collectors to export tracing information to the tracing server.



Configure the Collector

Provide the appropriate configuration information for your collector. The configuration information consists of three main sections, receivers, processors, and exporters. See https://opentelemetry.io/docs/collector/configuration/.

```
receivers:
       otlp:
               protocols:
                       grpc:

processors:
       batch:

exporters:
```

```
        otlp:
                endpoint: zoomdata-tempo:4317
                tls:
                        insecure: true

service:
        pipelines:
                traces:
                        receivers: [otlp]
                        processors: [batch]
                        exporters: [otlp]
```

- **Receivers**: Configure available protocols and endpoints for the receiver to receive tracing information from Symphony's microservices. insightsoftware recommends you use the OTlp protocol.
- **Processor**: Enable additional processing of tracing information. In this example, batch processing is enabled to optimize network communication.
- **Exporters**: Configure export information. Provide a destination URL and other connection parameters to communicate with the tracing server. insightsoftware recommends you use the OTlp protocol.

After you have installed and configured the OTel collector, install and configure a java agent and Symphony's microservices.

Configure Symphony Microservice Tracing

You must install and configure the OTel java agent to trace Symphony microservices. Optionally, next configure the Symphony front end for tracing. After these steps are complete and you've restarted the microservices, you can extract the traceID in a REST request or from a web socket outbound message.

- Install The Java Agent
- Configure The Collector
- Configure The Symphony Front End
- Extract The TraceID

Important: insightsoftware recommends you use the OTel java agent.

Install the Java Agent

1. Download the java agent to the server running the Symphony microservice. https://github.com/open-telemetry/opentelemetry-java-instrumentation/releases/latest/download/opentelemetry-javaagent.jar

2. Define the path to the java agent for Symphony. Select the option that works best for your environment.

   a. Specify the path in `SERVICE_NAME.jvm` (such as `zoomdata.jvm` or `edc-postgresql.jvm`) in the `/opt/zoomdata/conf/` folder, such as `-javaagent:/path/to/java/agent/opentelemetry-javaagent.jar`. See ComposerSymphony Data Discovery Microservice Name Reference.

   b. Define an environment variable, `_JAVA_OPTS`, that contains the path, such as `_JAVA_OPTIONS="-javaagent:/path/to/java/agent/opentelemetry-javaagent.jar"`.

Note: Some application performance monitoring tools provide their own java agents that are built on top of the OpenTelemetry java agent, or is compliant with the OTLP protocol. Use at your own discretion; the behavor may be different from the OTel java agent.

Configure the Java Agent

After you've installed the java agent, specify backend configuration properties for the agent using required and optional configuration properties. Include in the SERVICE_NAME.jvm or use environment variables. See https://opentelemetry.io/docs/instrumentation/java/automatic/agent-config/ and https://opentelemetry.io/docs/reference/specification/sdk-environment-variables/.

Required Configuration Properties

Property	Recommended Value	Description
OTEL_TRACES_EXPORTER	otlp	Defines the protocol to use to export tracing information.
OTEL_METRICS_EXPORTER	none	Disable metrics export using opentelemetry.
OTEL_LOGS_EXPORTER	none	Disable logs export using opentelemetry.
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	http://OTEL_COLLECTOR_HOST:4317	Defines the URL for the otel collector or tracing server.
OTEL_SERVICE_NAME	▪ zoomdata ▪ query-engine ▪ edc-postgresql	Define the names of the services.

Property	Recommended Value	Description
	■ edc-mssql ■ screenshot-service ■ data-writer	
OTEL_INSTRUMENTATION_SPRING_BOOT_ACTUATOR_AUTOCONFIGURE_ENABLED	false	Required, for now, to enable tracing. This may be updated after future otel releases.

Configure the Symphony Front End

After you've provided back end configuration properties for tracking, provide any needed variables for the Symphony front end as well.

Enable tracing and provide environment variables by specifying the fe.env configuration property for the zoomdata-web microservice using the following format: `fe.env=ENV_VARIABLE1=value1;ENV_VARIABLE2=value2;ENV_VARIABLE3=value3`.

The default value of `OTEL_SDK_DISABLED` is `false` in the OpenTelemetry SDK. When used in your Symphony environment, installation changes this value to `true` for front end support. To enable tracing, specify `OTEL_SDK_DISABLED=false` in the environment variable.

Required Configuration Properties

Property	Recommended Value	Description
OTEL_TRACES_EXPORTER	otlp	Defines the protocol to use to export tracing information.
OTEL_METRICS_EXPORTER	none	Disable metrics export using opentelemetry.
OTEL_LOGS_EXPORTER	none	Disable logs export using opentelemetry.
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	http://OTEL_COLLECTOR_HOST:4317	Defines the URL for the otel collector or tracing server.
OTEL_SERVICE_NAME	■ zoomdata ■ query-engine ■ edc-postgresql	Define the names of the services.

Property	Recommended Value	Description
	■ edc-mssql ■ screenshot-service ■ data-writer	
OTEL_INSTRUMENTATION_SPRING_BOOT_ACTUATOR_AUTOCONFIGURE_ENABLED	false	Required, for now, to enable tracing. This may be updated after future otel releases.

OpenTelemetry is executed in the user's browser and all tracing information is sent to the collector from the browser. The `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` must be reachable by the browser to execute tracing collection. Depending on your environment, you may need to configure cors.

After you have installed and configured the java agent, restart all microservices.

Extract the traceID

When tracing is configured and enabled, you can extract the `traceID` in a REST request or from a web socket outbound message. `traceID` is embedded in the `traceparent` property. The structure is `traceparent` is `{version}-{traceId}-{spanId}-{sampleDecision}`.

Manage Symphony Microservices

This applies to: *Visual Data Discovery*

You can manage Symphony microservices in CentOS environments or in Ubuntu 18, 20, or 22 environments. You can also manage the microservices using the Symphony CLI.

See the following links:

- Start Symphony Microservices
- Stop Symphony Microservices
- Enable ComposerSymphony Microservices
- Disable Symphony Microservices
- Restart Symphony Microservices
- Manage Symphony Microservices Using The Command Line Utility
- Downloading And Installing The Solution Package

Scaling Symphony Microservices

This applies to: *Visual Data Discovery*

Symphony microservices allow you to scale your Symphony installation, giving you expanded performance gains. As you roll out your microservices changes, you'll need to:

- Estimate User and Microservice Loads
- Add or Remove Nodes in a High Availability Environment
- Configure Throughput for Microservices
- Load Monitor Microservices
- Scale Microservices Up
- Scale Microservices Down

All microservices, except composer web, support horizontal scaling. Composer web supports vertical scaling only.

Estimate User and Microservice Loads

The load used by each microservice depends on the type of interactions your users have while using Symphony. Some actions load only the composer web component, while others may load multiple components.

Some examples of actions that load microservices are included in the table below.

Action	Microservices Loaded
Open the home page	composer web
Open a dashboard	composer web, connector, query engine
Open the visual Gallery	composer web
Create a source	composer web, connector, query engine
Create an uploaded source	composer web, connector, data writer, query engine
Upload new data via API	composer web, data writer
Execute scheduled dashboard report	composer web, connector, screenshot service, query engine

Action	Microservices Loaded
Live mode	composer web, connector, query engine

Note: Horizontal scaling up of a microservice doesn't provide doubled performance gains due to sharing of PostreSQL resources and overlapping of microservices.

Start your scale planning based on an approximate number of services and expected number of users. See Server Size Guidelines for more information on services and user estimation.

Add or Remove Nodes in a High Availability Environment

To scale a microservice up or down, you may need to Add Nodes to an Existing High Availability Installation or Remove Nodes from a High Availability Environment. Before you add or remove nodes, you should understand the actual load on your microservices to make the decision of scaling up or down. See Symphony System Metrics.

Configure Throughput for Microservices

All microservices have a configuration property, `server.jetty.max-threads`, you can use to limit throughput. Once configured, you can more accurately monitor the load on each microservice.

Set your estimated concurrent users for each service at +20%, based on your calculations made using Server Size Guidelines. To manage configuration properties of `server.jetty.max-threads`, see Configure and Start the Configuration Microservice.

For example, if you have three instances of the query engine microservice and you expect 150 concurrent users, each instance is expected to handle 50 concurrent users. To accommodate this demand, set `server.jetty.max-threads` to (`60 = 150 / 3 & 120%`) for each instance of query engine.

Load Monitor Microservices

With throughput defined for your microservices, you can measure their use to decide what services to scale up or scale down.

Symphony microservices exposes several different metrics to help you understand the current load on each service. The main outputs to monitor include:

- `jetty_threads_busy` shows a current count of concurrent users.
- `jetty_threads_config_max` shows the maximum allowed concurrent users. This should match the value you set in `server.jetty.max-threads`.

These show a moment in time, which can vary depending on use spikes that may not actually require scaling up or down. Smooth out the data by looking at a specific length of time, such as five minutes, and monitor the average load across all instances of the same microservice.

Average over five minutes:

```
avg by instance_type (jetty_threads_busy) over 5m / avg by instance_type (jetty_threads_config_max)
```

In Prometheus:

```
sum by (job)(sum_over_time(jetty_threads_busy[5m]))
/ sum by (job)(count_over_time(jetty_threads_busy[5m]))
/ avg by (job) (jetty_threads_config_max)
```

Scale Microservices Up

The recommended load threshold for all microservices is 90%. After setting up your thresholds, monitor them manually, or use a metric monitoring tool such as Prometheus to define alerts at that threshold.

```
sum by (job)(sum_over_time(jetty_threads_busy[5m]))
/ sum by (job)(count_over_time(jetty_threads_busy[5m]))
/ avg by (job) (jetty_threads_config_max)
> 90
```

For more information on scaling up your environment, see Add Nodes to an Existing High Availability Installation.

Scale Microservices Down

When monitoring microservices for scaling up, you define and set alerts around defined load thresholds for types of services.

In contrast, when you monitor services to scale them down, monitor free resources, the amount of free threads dedicated to the system. The recommended threshold is dedicate two times the resources of a single instance `(2 * resources)`.

Average over five minutes:

```
sum by instance_type (jetty_threads_config_max) - avg by instance_type (jetty_threads_busy) over 5m
```

In Prometheus:

```
(sum by (job) (jetty_threads_config_max) - (sum by (job)(sum_over_time(jetty_threads_busy[5m])) / sum by (job)(count_over_
time(jetty_threads_busy[5m])))) / avg by (job) (jetty_threads_config_max) > 2
```

Start Symphony Microservices

This applies to: *Visual Data Discovery*

This topic describes how to start Symphony microservices in CentOS environments or in Ubuntu 18, 20, or 22 environments.

The list of Symphony microservices can be found in ComposerSymphony Data Discovery Microservice Name Reference. The order in which microservices should be started is described in ComposerSymphony Microservice Startup Order.

You can also start Symphony microservices using the CLI. See Manage Symphony Microservices Using The Command Line Utility.

To start all Symphony microservices, enter the following command:

```
sudo systemctl start $(systemctl list-unit-files | grep zoomdata | grep edc | awk '{print $1}')
```

To start a specific Symphony microservice, enter the following command:

```
sudo systemctl start <service>
```

Replace `<service>` with the name of the Symphony microservice. See Composer Microservice Name Reference.

Stop Symphony Microservices

This applies to: *Visual Data Discovery*

This topic describes how to stop Symphony microservices in CentOS environments or in Ubuntu 18, 20, or 22 environments.

The list of Symphony microservices can be found in ComposerSymphony Data Discovery Microservice Name Reference.

You can also stop Symphony microservices using the CLI. See Manage Symphony Microservices Using The Command Line Utility.

To stop all Symphony microservices, enter the following command:

```
sudo systemctl stop $(systemctl list-unit-files | grep zoomdata | awk '{print $1}')
```

To stop a specific Symphony microservice, enter the following command:

```
sudo systemctl stop <service>
```

Replace `<service>` with the name of the Symphony microservice. See Composer Microservice Name Reference.

Disable Symphony Microservices

This applies to: *Visual Data Discovery*

This topic describes how to disable Symphony microservices in CentOS environments or in Ubuntu 18, 20, or 22 environments.

The list of Symphony microservices can be found in ComposerSymphony Data Discovery Microservice Name Reference.

You can also disable Symphonys microservices using the CLI. See Manage Symphony Microservices Using The Command Line Utility.

To disable all Symphony microservices, enter the following command:

```
sudo systemctl disable $(systemctl list-unit-files | grep zoomdata | awk '{print $1}')
```

To disable a specific Symphony microservice, enter the following command:

```
sudo systemctl disable <service>
```

Replace `<service>` with the name of the Symphony microservice. See ComposerSymphony Data Discovery Microservice Name Reference.

Restart Symphony Microservices

This applies to: *Visual Data Discovery*

This topic describes how to restart Symphony microservices in CentOS Stream 9 environments or in Ubuntu 18, 20, or 22 environments.

The list of Symphony microservices can be found in ComposerSymphony Data Discovery Microservice Name Reference.

You can also restart Symphony microservices using the CLI. See Manage Symphony Microservices Using The Command Line Utility.

To restart all Symphony microservices, enter the following command:

```
sudo systemctl restart $(systemctl list-unit-files | grep zoomdata | grep edc | awk '{print $1}')
```

To restart a specific Symphony microservice, enter the following command:

```
sudo systemctl restart <service>
```

Replace `<service>` with the name of the Symphony microservice. See Composer Microservice Name Reference.

Manage Symphony Microservices Using the Command Line Utility

This applies to: *Visual Data Discovery*

You can use the `zdmanage` CLI command to manage the various microservices that are deployed in the Symphony environment. This command line utility is automatically installed when installing your software using the automated installation script. This utility script wraps underlying UNIX commands to help you perform common management operations such as stopping and starting Symphony microservices.

Prerequisites

To use the command line utility tool, you need root access.

Using the Command Line Utility Tool

The `zdmanage` command line utility is located in the following directory:

```
/opt/zoomdata/bin/zdmanage
```

The `zdmanage` CLI command syntax is as follows:

```
zdmanage services <command> <service_name>
```

The following commands (`<command>`) are supported:

Command	Action
`list`	Displays all the Symphony microservices installed in your deployment.
`status`	Provides a status of all Symphony microservices in your environment.
`start`	Starts Symphony microservices. The order in which microservices should be restarted is described in ComposerSymphony Microservice Startup Order.
`stop`	Stops Symphony microservices.
`restart`	Restarts Symphony microservices.
`enable`	Enables Symphony microservices automatically when the OS starts up.
`disable`	Shuts down Symphony microservices automatically when the OS starts up.
`configure`	Opens the specified Symphony property file.

For `<service_name>`, specify the microservice name or `all` (to apply the command to all Symphony microservices). A complete list of the Symphony microservices can be found in ComposerSymphony Data Discovery Microservice Name Reference.

Common Commands

Show all Symphony microservices installed:

```
sudo /opt/zoomdata/bin/zdmanage services list
```

Provide a status of all installed Symphony microservices:

```
sudo /opt/zoomdata/bin/zdmanage services status all
```

Show the status of the Symphony web microservice:

```
sudo /opt/zoomdata/bin/zdmanage services status zoomdata
```

Stop a specific Symphony microservice:

```
sudo /opt/zoomdata/bin/zdmanage services stop [zoomdata-service_name]
```

Symphony Monitoring Solution

This applies to: *Visual Data Discovery*

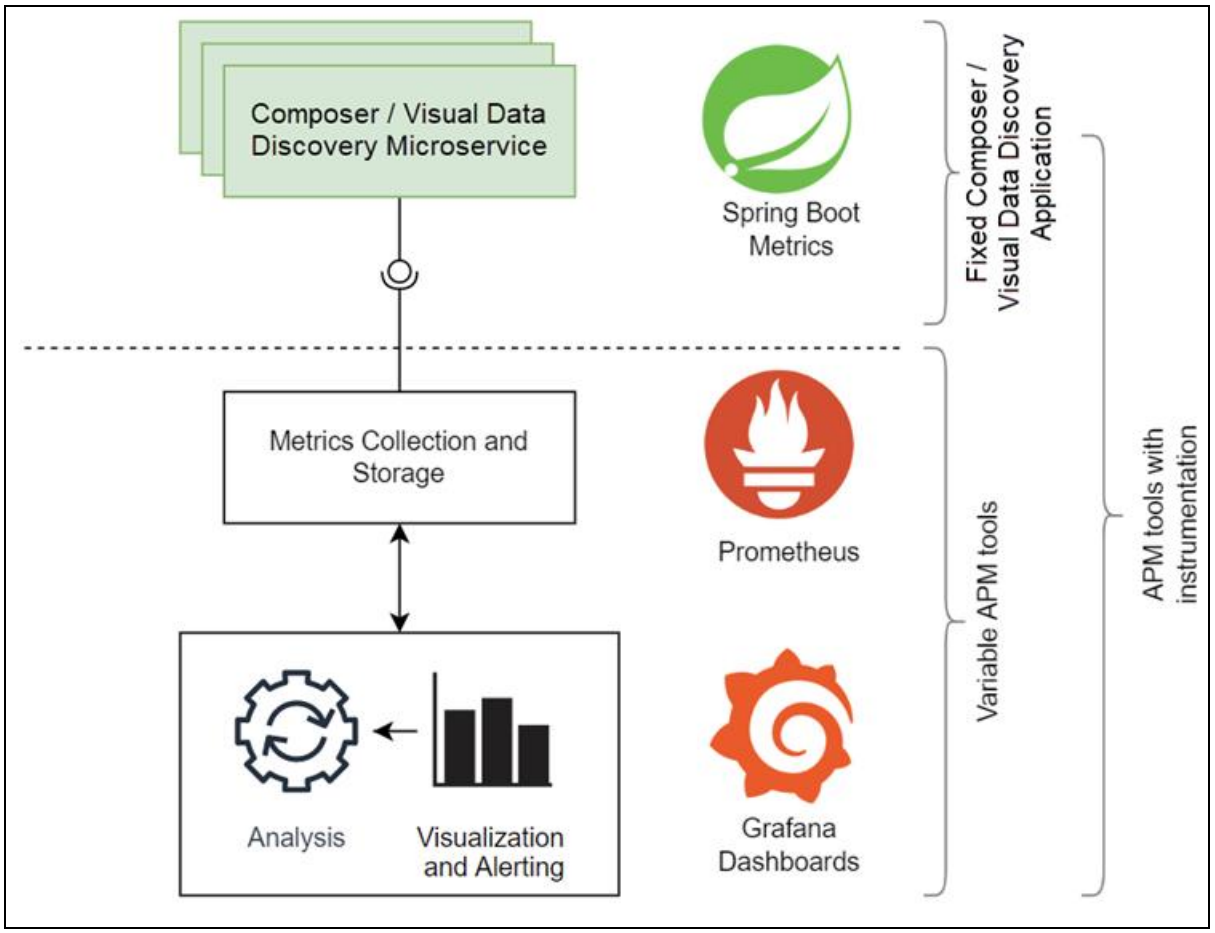
Symphony system metrics are published by all Symphony microservices. While these measurements provide some visibility of the operating state of the services, the metric data becomes more useful once it can be stored and viewed. In order to provide Symphony deployments with observability into the health and performance of the running system, you can use a monitoring solution. This topic describes an example monitoring solution that we make available for download with each release of Symphony.

Monitoring Solution Components

The monitoring solution consists of several components that are built up as layers.

- Generic operating and Symphony specific metrics form the fixed foundation layer and are published in various formats.
- Subsequent layers provide storage and visualization capabilities to support further analysis.

Note: Subsequent components added to the metrics foundation can be implemented as provided in the monitoring solution package, or you can substitute them with preferred components that were already deployed.



The example solution is built using the following industry-standard, open-source tools:

1. Prometheus service collects and stores metrics in a time series database.
2. Grafana supports further analysis by providing visualization of stored metrics on pre-built application-specific dashboards.

We provide these components in a tarball containing docker-compose files. We include the configuration files required for monitoring Symphony services, along with ready to use dashboards.

Downloading and Installing the Solution Package

1. Contact your SymphonyCustomer Support representative to download the Solution Package: Customer Support.
2. When you get the tarball, extract the tarball to the directory you create.

```
mkdir composer-monitoring
tar -xvf composer-monitoring-X.Y.Z.tar.gz -C composer-monitoring
```

3. Open the READ.me file and follow the directions included.

Configure Symphony

This applies to: *Visual Data Discovery*

The Symphony server uses multiple configuration files to ensure successful deployment of Symphony in your operating environment. Each Symphony component has configuration files including a `<component_name>.properties` file and a `<component_name>.jvm` file.

Use the Symphony configuration files to make changes to the Symphony server and how it functions. The configuration files need to be created after the Symphony server has been installed to override Symphony's default configuration. To modify the settings in these files, edit them in the `/etc/zoomdata` directory, as described in Edit A Data Discovery Configuration File.

A full list of the configuration file names can be found in Configuration Property Files.

Every `<component_name>.jvm` file should include an `Xmx` JVM option to control the maximum heap memory limit for the application. See Configure Memory Settings.

Edit a Data Discovery Configuration File

This applies to: *Visual Data Discovery*

You can edit the properties for your Symphony configuration in the configuration files. For a complete list of configuration files, see Configuration Property Files.

> **Note:** insightsoftware discourages changing properties in the `/opt/zoomdata/conf` directory (Linux) or `<install-path>/conf` (Windows). Copy the files you want to change to the `/etc/zoomdata` directory (Linux) or `<install-path>/conf-modify` (Windows) and change them there. This will ensure that your changes are not overwritten when Symphony is next upgraded.

Quickly determine what changes you've made to a properties file using `diff` in Linux. For example:

```
diff /opt/zoomdata/conf/edc-<connector-name>.properties /etc/zoomdata/<edc-<connector-name>.properties
```

or

```
diff /opt/zoomdata/conf/zoomdata.properties /etc/zoomdata/zoomdata.properties
```

For Windows environments, use your preferred diff utility to compare the differences between your original and updated property files.

Edit a Symphony configuration file in Linux:

1. From your terminal, SSH to your Symphony server.
2. Stop the appropriate Symphony microservice (Symphony, Screenshot, or the specific connector server).

   For the appropriate Linux command, see Stop Symphony Microservices.
3. Use the following command to access and open the configuration file:

   ```
   vi /etc/zoomdata/<config-file-name>
   ```

   For example:

   ```
   vi /etc/zoomdata/zoomdata.properties
   ```

   Names of valid configuration files are listed in Configuration Property Files.

If the configuration file does not exist, this command creates it.

> **Note:** If you are not logged in as a root user, then you need to enter `sudo vi /etc/zoomdata/zoomdata.properties` to create the desired file.

Make sure the file is readable by the `zoomdata` microservice account.

4. Add the new variable or property into the file on a new line or edit an existing one, as needed. See:

i. Zoomdata.properties Properties

ii. zoomdata.jvm Options

iii. Query Engine Properties

iv. screenshot-service.properties Properties

v. Connector Properties and Property Files

5. Save and exit the configuration file.

6. Restart Symphony microservices.

   For the appropriate Linux command line, see Restart Symphony Microservices.

Symphony System Metrics

This applies to: *Visual Data Discovery*

Symphony exposes metrics for discovering and monitoring Symphony microservices. Discover system metrics using one of several available tools.

Discovering Metrics

Connect to the composer-webservice actuator endpoints at: `http://[host_ip]:[port]/composer/actuator`.

Available endpoints include:

```
"prometheus": {
       "href": "http://127.0.0.1:8080/composer/actuator/prometheus",
       "templated": false
},
       "metrics-requiredMetricName": (
              "href": "http://127.0.0.1:8080/composer/actuator/metrics/{requiredMetricName)",
              "templated": true
       },
       "metrics":{
              "href": "http://127.0.0.:8080/composer/actuator/metrics",
              "templated": false
       },
```

Connect to all other microservices actuator endpoints at: `http://[host_ip]:[port]/actuator`.

Available endpoints include:

```
{
       "_links": {
              "self": {
                     "http://127.0.0.1:8105/actuator",
                     "templated": false
              },
              "metrics-sorted": (
                     "href": "http://127.0.0.1:8105/actuator/metrics-sorted",
                     "templated": false
              },
              "health-path": (
                     "href": "http://127.0.0.1:8105/actuator/health/{*path}",
```

```
                "templated": true
            },
```

For a list of all metrics exposed by Composer, connect to: `http://[host_ip]:[port]/actuator/metrics`.

Monitoring Microservices: Prometheus

Symphony metrics data is formatted so you can access that data through a Prometheus server. Navigate to the Prometheus composer-web actuator endpoint to see your data: `http://[host_ip]:[port]/composer/actuator/prometheus`.

Example Prometheus actuator response:

```
# HELP jvm_classes_unloaded_classes_total The total number of classes unloaded since the Java virtual machine has started
execution
# TYPE jvm_classes_unloaded_classes_total counter jvm_classes_unloaded_classes_total{service_instance="Zoomdata:ip-127-0-0-
0.ec2.internal:8080",} 63.0
# HELP process_files_max_files The maximum file descriptor count
# TYPE process_files_max_files gauge process_files_max_files_{service_instance="Zoomdata:ip-127-0-0-0:ec2.internal:8080",}
10240.0
# HELP jvm_gc_memory_allocated_bytes_total Incremented for an increase in the size of the young generation memory pool
after one GC to be
# TYPE jvm_gc_memory_allocated_bytes_total counter jvm_gc_memory_allocated_bytes_total{service_instance="Zoomdata:ip-127-0-
0-0.ec2.internal:8080",} 4.1484812288E10
# HELP websocket_sessions_active
# TYPE websocket_sessions_active gauge websocket_sessions_active{service_instance="Zoomdata:ip-127-0-0-
0.ec2.internal.8080",} 2.0
# HELP zoomdata_sessions_count_last_minute
# TYPE zoomdata_sessions_count_last_minute gauge zoomdata_sessions_count_last_minute{service_instance=Zoomdata:ip-127-0-0-
0.ec2.internal:8080",} 0.0
```

Configure Prometheus

Install Prometheus. See https://prometheus.io/docs/prometheus/latest/getting_started/ for more information.

Build your `prometheus.yml` as shown in the example here to connect to the composer-webservice. Replace `[username]`, `[password]` , `[host_ip]` and `[port]` with your appropriate values.

```
scrape_configs:
       - job_name: 'composer-web'
              metrics_path: 'composer/actuator/prometheus'
```

```
            scrape_interval: 5s
            basic_auth:
                    username: [username]
                    password: [password]
            static_configs:
            - targets: ['[host_ip]:[port]']

    - job_name: 'query-engine'
            metrics_path: 'actuator/prometheus'
            scrape_interval: 5s
            static_configs:
            - targets: ['[host_ip]:[port]']
```

Monitoring Microservices: Statsd and Graphite

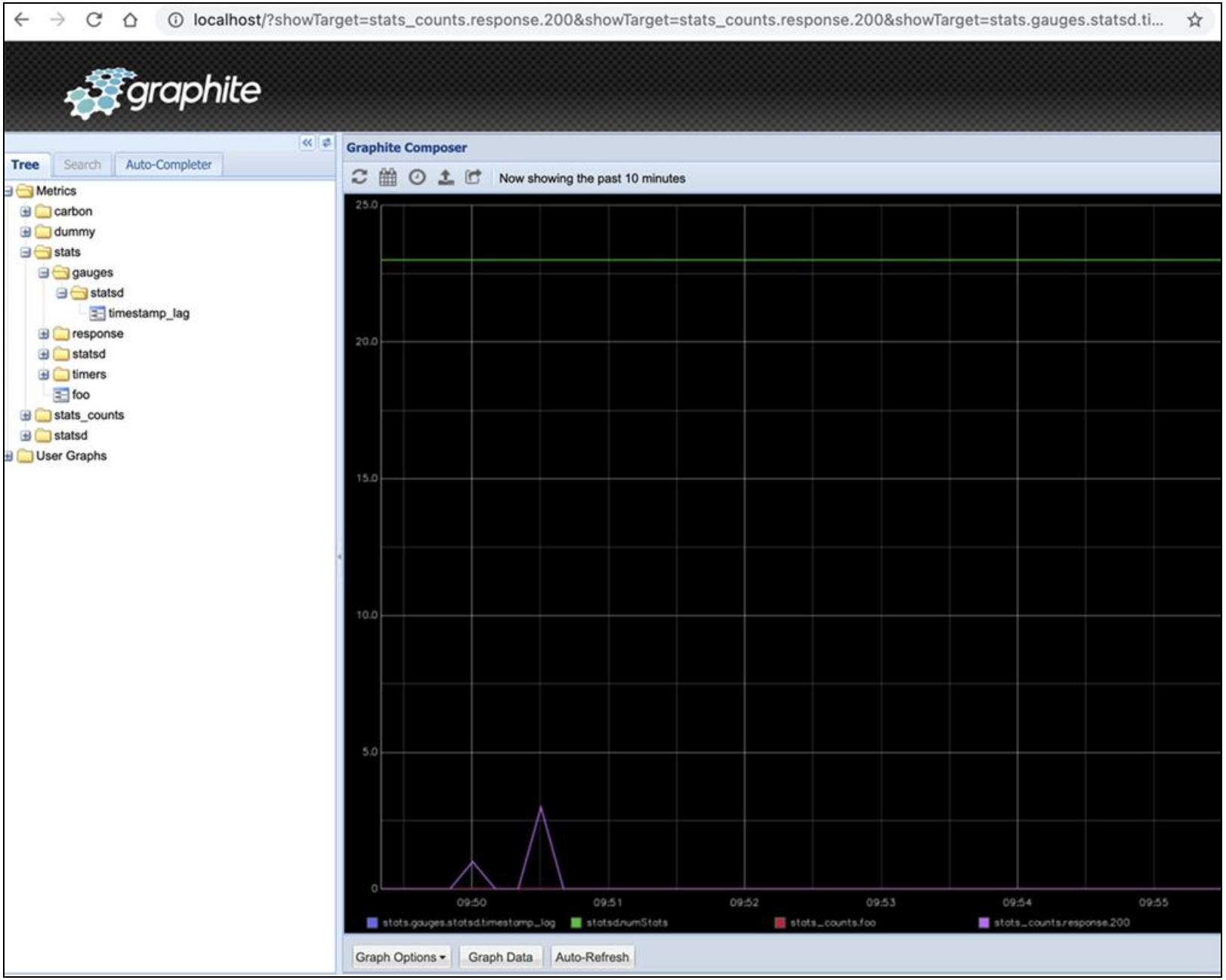
You can also collect Symphony metrics using the network daemon statsd. This network daemon runs on the Node.js platform to listen for statistics sent over UDP or TCP and sends aggregate information to one or more backend services. Unlike Prometheus, statsd does not collect metric data from specific endpoints.

Use in conjunction with Graphite to store numeric time-series data and render data graphs on demand.

Setup and Configure Statsd and Graphite

Install statsd and Graphite to connect to and access system metrics.

Example of a Graphite dashboard with metrics collected by statsd:


localhost/?showTarget=stats_counts.response.200&showTarget=stats_counts.response.200&showTarget=stats.gauges.statsd.ti...
graphite
Tree
Search
Auto-Completer
Metrics
carbon
dummy
stats
gauges
statsd
timestamp_lag
response
statsd
timers
foo
stats_counts
statsd
User Graphs
Graphite Composer
Now showing the past 10 minutes
25.0
20.0
15.0
10.0
5.0
0
09:50
09:51
09:52
09:53
09:54
09:55
stats.gauges.statsd.timestamp_lag
statsd.numStats
stats_counts.foo
stats_counts.response.200
Graph Options
Graph Data
Auto-Refresh

Configuration Property Files

This applies to: *Visual Data Discovery*

Symphony uses the configuration files in the following table to ensure successful deployment in your operating environment. You can edit many of the properties and options in these files. You can edit them in `/etc/zoomdata` as described in Edit A Data Discovery Configuration File.

Note: insightsoftware discourages changing properties in the `/opt/zoomdata/conf` directory (Linux) or `<install-path>/conf` (Windows). Copy the files you want to change to the `/etc/zoomdata` directory (Linux) or `<install-path>/conf-modify` (Windows) and change them there. This will ensure that your changes are not overwritten when Symphony is next upgraded.

Quickly determine what changes you've made to a properties file using `diff` in Linux. For example:

```
diff /opt/zoomdata/conf/edc-<connector-name>.properties /etc/zoomdata/<edc-<connector-name>.properties
```

or

```
diff /opt/zoomdata/conf/zoomdata.properties /etc/zoomdata/zoomdata.properties
```

For Windows environments, use your preferred diff utility to compare the differences between your original and updated property files.

File	Defines....
`admin-server.jvm`	JVM options related to the Symphony Service Monitor.
`admin-server.properties`	Configuration properties related to the Symphony Service Monitor.
`config-server.jvm`	JVM options related to the Symphony configuration microservice.
`config-server.properties`	Configuration properties related to the Symphony configuration microservice.
`consul.json`	Settings related to the Symphony Service Discovery microservice. For a description of these settings, see https://www.consul.io/docs/agent/options.
`data-writer-postgresql.jvm`	JVM options related to the Data Writer microservice's Postgres relational database.
`data-writer-postgresql.properties`	Configuration properties related to the Data Writer microservice's Postgres relational database.
`edc-<connector>.jvm`	JVM options related to the specific Symphony connector microservice. For example, `edc-impala.jvm` contains JVM options for the Symphony Cloudera Impala connector microservice. See Connector Properties And Property Files.

File	Defines....
`edc-<connector>.properties`	Configuration properties related to the specific Symphony connector microservice. For example, `edc-impala.properties` contains configuration properties for the Symphony Cloudera Impala connector microservice. See Connector Properties And Property Files.
`query-engine.jvm`	JVM options related to the Symphony query engine. See Manage The Symphony Query Engine.
`query-engine.properties`	Configuration properties related to the Symphony query engine. See Query Engine Properties.
`screenshot-service.jvm`	JVM options related to Symphony Screenshot microservice processing.
`screenshot-service.properties`	Configuration properties related to Symphony Screenshot microservice processing. See Screenshot-service.properties Properties.
`zoomdata.jvm`	JVM options (e.g. memory configuration) and Java system properties (e.g. timezone, temp directory, etc.) related to Symphony. See Zoomdata.jvm Options.
`zoomdata.properties`	Configuration properties related to Symphony. See Zoomdata.properties Properties.

zoomdata.properties Properties

This applies to: *Visual Data Discovery*

The zoomdata.properties file can be edited in the `/etc/zoomdata` directory. Each property is described in the table below.

Some situations where the `zoomdata.properties` file needs to be updated include:

- Add An SSL Certificate
- Disable The SSL Certificate In Symphony Data Discovery
- Set Up The Screenshot Microservice
- Create A Symmetric Key To Encrypt Data Source Passwords
- About Scheduled Dashboard ReportsAbout Scheduled Data Discovery Dashboard Reports
- Set Up and Use the Data Gateway Service

For information on editing configuration files, see Edit A Data Discovery Configuration File.

Property	Default Value	Description
access.control.allow.origin	*	By default, CORS is set to --- in the Symphony Server. You can set CORS to restrict access: `access.control.allow.origin=<user-defined>` For more information, see Enable Composer Visual Data Discovery Component Access From Other Sites Using Cross-Origin Resource Sharing (CORS).
logs.dir	*<ZD_install_directory>* /logs	Path to Symphony logs. The placeholder *<ZD_install_directory>* is replaced with the actual location where Symphony is installed. Verify that this log directory has all the necessary permissions and that the owner of the directory is set to `zoomdata`. The `/home` directory cannot be used for logging. Example: `logs.dir=/opt/<composer>/logs)`

Property	Default Value	Description
data-gateway.client-api.enabled	False	Set to true to enable use of the Data Gateway API and Data Gateway Service. See Set Up and Use the Data Gateway Service.
saml.maxAuthAge	86400	Sets the timeout for SAML, in seconds. The default is 24 hours. Example: `saml.maxAuthAge=86400`
server.compression.enabled	true	Enables gzip compression for http requests. Example: `server.compression.enabled=true`
server.port	8080	The default server port, which is set to use http. Prior releases used `http.port`
server.servlet.context-path	/composer	Example: `server.servlet.context-path=/composer`
server.session-timeout	1800 seconds	Sets when your Symphony session will timeout (in seconds). Example: `server.session-timeout=1800` If you alter this value, also alter the value of the zoomdata.server.ws.idle.timeout property to match it.
source.attribute.values.limit	1000	Sets the limit for the number of attribute values that can be displayed in the Filter list. Example: `source.attribute.values.limit=1000`
spring.servlet.multipart.max-file-size	500Mb	Example: `spring.servlet.multipart.max-file-size=500Mb`
spring.servlet.multipart.max-request-size	500Mb	Example: `spring.servlet.multipart.max-request-size=500Mb`
zoomdata.server.ws.idle.timeout	1800000 ms	Idle time that allows the WebSocket to be still valid. If you alter this value, also alter the value of the server.session-timeout property to match it.
http.response.header.content-security-policy	frame-ancestors *;	Add appropriate values to override the default values and support your business needs, such as `default-src 'self'`, `script-src 'self'`, and more. Secure your implementation by including values for these resources using appropriate source URLs. Example: `frame-ancestors *; default-src 'self'; script-src`

Property	Default Value	Description
		`'self' 'nonce-composerScript' https://*.storage.example.com; style-src 'self' 'unsafe-inline' https://*.storage.example.com; img-src 'self'; connect-src 'self'; font-src 'self'; base-uri 'self';`
Source Metadata Properties		
zoomdata.source.refresh.metadata.cache.timeout.minutes	10080 (one week)	Sets the default time-to-live for cache values, in minutes.
zoomdata.source.refresh.values.maxDistinctValues	100000	Sets the number of queried distinct values that can be stored in cache.
Encryption Properties		
security.encryption.algorithm		The encryption algorithm used for file encryption.
security.encryption.key.algorithm		The algorithm type of the encryption key used for file encryption.
Keystore Properties		
keystore.location	classpath:security/zoomkeystore.jks	Symphony uses symmetric encryption. You can point to a new keystore to strengthen security. Example: `keystore.location=classpath:security/zoomkeystore.jks` See Create A Symmetric Key To Encrypt Data Source Passwords for further guidance.
keystore.password	zoomkey	Lets you set up a unique password for the keystore. Example: `keystore.password=zoomkey`
keystore.key.alias	zoomkey	Example: `keystore.key.alias=zoomkey`
keystore.key.password	zoomkey	Example:`keystore.key.password=zoomkey`
Server SSL Properties		
server.ssl.key-store	<Symphony_install_directory>/conf/keystore	Sets the path for the keystore location. Example: `server.ssl.key-store=HOME/conf/keystore`
server.ssl.key-store-password	changeit	Stores the keystore password. Example: `server.ssl.key-store-password=<YourPassword>`

Property	Default Value	Description
SAML Configuration Properties		
saml.artifactBindingDefault	true	Example: `saml.artifactBindingDefault=true`
saml.useMultiValueList	true	Example: `saml.useMultiValueList=true`
saml.stringDelimiter	,	Example: `saml.stringDelimiter=;`
Kerberized PostgreSQL Properties		
spring.datasource.url	jdbc:postgresql://<IP_address>:<port>/zoomdata	The URL of the `zoomdata` database in the PostgreSQL metadata store. Example: `spring.datasource.url=jdbc:postgresql://localhost:5432/zoomdata`
spring.datasource.username	zoomdata	The user name for the `zoomdata` database in the PostgreSQL metadata store. Example: `spring.datasource.username=zoomdata`
spring.datasource.password	---	The password associated with the user name for the `zoomdata` database in the PostgreSQL metadata store.
keyset.destination.params.jdbc_url	jdbc:postgresql://<IP_address>:<port>/zoomdata-keyset	The URL of the `zoomdata-keyset` database in the PostgreSQL metadata store. Example: `keyset.destination.params.jdbc_url=jdbc:postgresql://10.2.1.4:5432/zoomdata-keyset`
keyset.destination.params.user_name	zoomdata	The user name for the `zoomdata-keyset` database in the PostgreSQL metadata store. Example: `keyset.destination.params.user_name=zoomdata`
keyset.destination.params.password	---	The password associated with the user name for the `zoomdata-keyset` database in the PostgreSQL metadata store.
upload.destination.params.jdbc_url	jdbc:postgresql://<IP_address>:<port>/zoomdata-upload	The URL of the `zoomdata-upload` database in the PostgreSQL metadata store. Example: `upload.destination.params.jdbc_url=jdbc:postgresql://10.2.1.4:5432/zoomdata-upload`
upload.destination.params.user_name	zoomdata	The user name for the `zoomdata-upload` database in the PostgreSQL metadata store. Example: `upload.destination.params.user_name=zoomdata`

Property	Default Value	Description
upload.destination.params.password	---	The password associated with the user name for the `zoomdata-upload` database in the PostgreSQL metadata store.
Source Sampling Properties		
source.sampling.rows	1000	Example: `source.sampling.rows=1000`
source.attribute.values.limit	1000	Example: `source.attribute.values.limit=1000`
Logging Properties		
logging.unified.host	127.0.0.1	Sets the host IP address for Fluentd server message logging. For more information, see Set Up Unified Logging Using Fluentd. Example: `logging.unified.host=123.4.5.6`
logging.unified.level	OFF	Sets the log level for messages logged to the Fluentd server. The following options are available for this property: TRACE, DEBUG, INFO, WARN, ERROR, and OFF. If set to OFF, Fluentd unified logging is disabled. For more information, see Set Up Unified Logging Using Fluentd. Example: `logging.unified.level=INFO`
logging.unified.port	24224	Sets the port for Fluentd server message logging. For more information, see Set Up Unified Logging Using Fluentd. Example: `logging.unified.port=1234`
logging.unified.tag	zoomdata-server	Sets the microservice tag name for messages logged to the Fluentd server. This is important because the tag identifies the microservice to which the log messages apply. Valid values are `query-engine`, `zoomdata-server`, `stream-writer`, `upload-service`, and `edc-<connector-name>` (where `<connector-name>` is one of the names listed in Connector Properties And Property Files). For more information, see Set Up Unified Logging Using Fluentd. Example: `logging.unified.tag = zoomdata-server`
syslog.host	127.0.0.1	Sets the host IP address for Syslog server message logging. Example: `syslog.host=127.0.0.1`
syslog.log.level	OFF	Sets the syslog log level for messages logged to the Syslog server. The following options are available for this property: TRACE, DEBUG, INFO, WARN, ERROR, and OFF.

Property	Default Value	Description
		Example: `syslog.log.level=DEBUG`
syslog.port	1514	Sets the port for Syslog server message logging. Example: `syslog.port=1514`
syslog.suffix	local	Specifies a suffix that is appended at the end of the Syslog server log entry that Symphony generates. Example: `syslog.suffix=local`
Password Policy		
auth.password.policy.specialCharacters	!@#$%^&*()-_=+,.:;<>	
auth.password.policy.minCharacters	9	
auth.password.policy.maxCharacters	255	
auth.password.policy.minLowercaseCharacters	1	
auth.password.policy.minUppercaseCharacters	1	
auth.password.policy.minNumericCharacters	1	
auth.password.policy.minSpecialCharacters	1	
auth.password.policy.helpMessage	Password must contain at least 9 characters including 1 lowercase, 1 uppercase, 1 number and 1 special (!@#$%^&*()-_=+,.:;<>).	Text is not enclosed in quotation marks.
Data Export Properties		
zoomdata.export.data.max.cols	1000 columns	Use this property to define the maximum number of columns that can be exported for two-dimensional visuals (such as a pivot table). Symphony enforces this limit for visual data, but does not enforce it for raw data. The distributed default for this setting is 1000 columns. Valid values can range from 0 through 2147483647 columns.
zoomdata.export.data.max.rows	100000 rows	Use this property to define the maximum number of rows that can be exported for visuals. Symphony enforces this limit for visual data. However, for raw data, Symphony produces an error if the number of rows requested for export exceeds this setting. The distributed default for this setting is 100000 rows. Valid values can

Property	Default Value	Description
		range from 0 through 2147483647 rows.
zoomdata.export.visualdata.max.rows	100000 rows	Use this property to define the maximum number of rows that can be exported for visuals. Symphony enforces this limit for visual data. However, for raw data, Symphony produces an error if the number of rows requested for export exceeds this setting. The distributed default for this setting is 100000 rows. Valid values can range from 0 through 2147483647 rows.
Screenshot Microservice Client & Dashboard Scheduling Properties		
screenshot.service.name	screenshot-service	
screenshot.service.url	`http://localhost:8083/`	
screenshot.service.http.client.connect.timeout.milliseconds	10000 milliseconds	Specifies the number of milliseconds that can elapse before Symphony stops trying to connect to the screenshot microservice client.
screenshot.service.http.client.read.timeout.milliseconds	60000 milliseconds	Specifies the number of milliseconds that can elapse before Symphony stops trying to read from the screenshot microservice client. **Note:** If you increase the time set by the `dashboard.scheduling.screenshot.timeout` property, make sure that you increase the value of this property as well. The total time set by `screenshot.service.http.client.read.timeout.milliseconds` should always be greater than or equal to the time set by the `dashboard.scheduling.screenshot.timeout` property. Bear in mind that this property is specified in milliseconds, but the `dashboard.scheduling.screenshot.timeout` property is specified in seconds.
screenshot.service.http.client.write.timeout.milliseconds	60000 milliseconds	Specifies the number of milliseconds that can elapse before Symphony stops trying to write to the screenshot microservice client.
dashboard.scheduling.screenshot.png.height	720 pixels	Identifies the height (in pixels) of the screenshot PNG file that will be sent.
dashboard.scheduling.screenshot.png.width	1280 pixels	Identifies the width (in pixels) of the screenshot PNG file that will be sent.
dashboard.scheduling.screenshot.timeout	60 seconds	Specifies the timeout (in seconds) to take a screenshot for a dashboard email report.

Property	Default Value	Description
		Note: The time specified by this property must be less than or equal to the time set by the `screenshot.service.http.client.read.timeout.milliseconds` property. If you increase the value of this property, make sure that you increase the value of the `screenshot.service.http.client.read.timeout.milliseconds property` accordingly. Bear in mind that this property is specified in seconds, but the `screenshot.service.http.client.read.timeout.milliseconds` property is specified in milliseconds.
mail.from		Specifies the email address identifying where the email comes from.
mail.login		Specifies the email login to use to access the mail server.
mail.password		Specifies the password associated with the email login identified in the `mail.login` property.
In addition, JavaMail API properties (Symphony supports both IMAP and SMTP protocols) should be added to the `zoomdata.properties` file to identify the mail server and other mail properties required to use that server to send the scheduled dashboard (for example, `mail.smtp.auth`, `mail.smtp.host`, `mail.smtp.port`, `mail.imap.host`, and `mail.imap.port`). Complete descriptions of IMAP and SMTP protocol JavaMail properties can be found at these links: ■ IMAP: https://javaee.github.io/javamail/docs/api/com/sun/mail/imap/package-summary.html ■ SMTP: https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html		
Field Settings		
zoomdata.detect.type.attribute.max.length	200 characters	Use this property to set the maximum character length of attribute fields. If this limit is exceeded, the field will be recognized as a Text field.

zoomdata.jvm Options

This applies to: *Visual Data Discovery*

The `zoomdata.jvm` file contains JVM options (e.g. memory configuration) and Java system properties (e.g. timezone, temp directory, etc.) related to Symphony. The table below describes the options you can adjust.

Situations where the `zoomdata.jvm` file needs to be edited include:

- Configure Memory Settings
- Connect To A Kerberized CDH Cluster

For information on editing configuration files, see Edit A Data Discovery Configuration File.

Option	Default Value	Description
DEBUG_ENABLED	0/false	Toggle switch to enable or disable the Java debug capability. To enable, enter '1' or 'true'. Example: `DEBUG_ENABLED=false`
DEBUG_PORT	9393	The default port for the Java debug capability. Example: `DEBUG_PORT=9393`
JAVA_OPTS	-Xss256k -Xms2048m -Xmx8192m	Java-related options for JVM. Refer to Oracle's article on Java HotSpot VM Options for information.
KERBEROS_CONFIG	/etc/krb5.conf	Default location for the Kerberos configuration details. However, the path to the file may be different in your environment. Refer to Oracle's article on File Formats for information.
KERBEROS_PRINCIPAL	hdfs@HADOOP.COM	Kerberos principal name.
KERBEROS_KEYTAB	/etc/zoomdata/zoomdata.keytab	Kerberos keytab location.
PROXY_HOST	user-defined	For cloud-based connectors (including Google Analytics and Salesforce) being used in a proxy configuration, this property specifies the server host to be returned for calls, and identifies the proxy host server that will provide internet access.
PROXY_PORT	user-defined	For cloud-based connectors (including Google Analytics and Salesforce) being used in a proxy configuration, this property specifies the server port to be returned for calls, and identifies the proxy port server that will provide internet access.

Query Engine Properties

This applies to: *Visual Data Discovery*

Most of the query engine components can be edited in the `query-engine.properties` file. You can manage the query engine using the following properties. For information on editing configuration files, see Edit A Data Discovery Configuration File.

Property	Default Value	Description
Service Logging		
access.log.file.size	10	Access log file size (in MB).
qe.error.log.file.size	5	Error log file size (in MB).
log.file.size	20	General log file size (in MB).
websocket.log.file.size	10	Websocket log file size (in MB).
syslog.host	127.0.0.1	Sets the host IP address for Syslog server message logging.
syslog.log.level	OFF	Sets the syslog log level for messages logged to the Syslog server. The following options are available for this property: TRACE, DEBUG, INFO, WARN, ERROR. and OFF.
trace.requests	false	Enables detailed tracing of HTTP request.
tracing.sampler.probability	1	The distributed tracing request rate percentage. Valid values are between 0 to 1.0, where 0 disables tracing and 1.0 indicates tracing 100% of requests.
General Microservice Configuration for Jetty Web Server		
server.jetty.max-threads	200	Number of threads to serve the HTTP & WebSocket clients.
qe.server.ws.idle.timeout	86400000	Idle time (in milliseconds) that allows the WebSocket to be still valid.
qe.server.ws.input.buffer.size	16384	Buffer size (in bytes) for the WebSocket message.
qe.server.ws.max.message.size	1048576	Max size (in bytes) of a message that could be sent over the query engine WebSocket.
server.port	5580	REST/WebSocket microservice port.
server.ssl.enabled	false	Defines whether the REST/WS API should use SSL.
server.ssl.key-store		Path to the file with the keystore.
Graceful Shutdown		
application.graceful.shutdown.enabled	true	Indicates whether or not graceful shutdown processing should occur. Valid values are `true` (perform graceful shutdown processing) or `false` (do not perform graceful shutdown processing).
application.graceful.shutdown.event-	5	Specifies how long (in seconds) a query engine instance will wait to allow clients to receive the

Property	Default Value	Description
propagation-timeout-sec		information that it is out of service.
application.graceful.shutdown.force-kill-timeout-sec	30	The maximum number of seconds that a query engine instance will wait for the number of its active tasks to reach zero. When this time has elapsed, all remaining active WebSockets serving in-flight queries are closed and then the query engine instance will stop.
Topology Configuration		
calculations.detect.array.fields	true	Enables and disables the query engine validation of multivalue fields in a derived field. By disabling (set the value to `false`) this functionality, any custom metrics that you may have created in earlier versions of Symphony that are aggregations of multivalue fields will produce valid values.
qe.max.allowed.time.groups	10000	The maximum amount of time in which groups can be generated by the **Include Blanks** function. For information on even time intervals, see Even Time Intervals .
qe.max.rows.during.execution	5000000	The maximum number of rows that can be processed during a single query. The value specified must be at least slightly greater than the value for the `qe.zengine.edc.rows.limit` property. If you increase the value of `qe.zengine.edc.rows.limit`, consider increasing the value of this property. When a query exceeds the limit set by this property, a `Resource limit is reached during query execution` message appears.
qe.zengine.edc.rows.limit	1000000	The maximum number of records that can be fused from a single data source.
sharpening.enabled	true	Enables or disables Data Sharpening. Set this property to `true` to enable Data Sharpening; set it to `false` to disable it.
sharpening.samples.max.count	100	The maximum number of Data Sharpening requests that can be generated by the sharpening process.
sharpening.samples.skip.first	1	Defines how many first samples are skipped before visualizing the result of sharpening.
zoomdata.validation.histogram.max.size	1000	Histogram max bucket count.

screenshot-service.properties Properties

This applies to: *Visual Data Discovery*

The following table lists properties you might adjust for the Symphony Screenshot microservice.

For information on editing configuration files, see Edit A Data Discovery Configuration File.

Property Name	Default Value	Description
pool.queue.size	10	Sets the queue size for Screenshot microservice requests. This property and the pool.thread.size property specify the upper limits for Screenshot microservice processing. When the number of screenshot requests exceeds the limits set by these two properties, you will receive HTTP 429 "Too Many Requests" errors. You can exceed these limits in a number of ways, including: ■ starting up Symphony with lots of dashboards and visuals ■ deleting all your screenshots at once ■ rapidly creating a lot of large dashboards You can increase the values of this property and the pool.thread.size property when you encounter too many failed screenshots. However, do so with caution.
pool.thread.size	20	Sets the thread count for Screenshot microservice requests. This property and the pool.queue.size property specify the upper limits for Screenshot microservice processing. When the number of screenshot requests exceeds the limits set by these two properties, you will receive HTTP 429 "Too Many Requests" errors. You can exceed these limits in a number of ways, including: ■ starting up Symphony with lots of dashboards and visuals ■ deleting all your screenshots at once ■ rapidly creating a lot of large dashboards If your use of Symphony includes scheduling dashboard reports, update this setting so it is

Property Name	Default Value	Description
		greater than the number of concurrent reports. You can also increase the values of this property and the pool.queue.size property when you encounter too many failed screenshots. However, do so with caution.
syslog.host	127.0.0.1	Sets the host IP address for Syslog server message logging. Example: `syslog.host=127.0.0.1`
syslog.log.level	OFF	Sets the syslog log level for messages logged to the Syslog server. The following options are available for this property: TRACE, DEBUG, INFO, WARN, and ERROR. Example: `syslog.log.level=DEBUG`
syslog.port	1514	Sets the port for Syslog server message logging. Example: `syslog.port=1514`
syslog.suffix	local	Specifies a suffix that is appended at the end of the Syslog server log entry that Symphony generates. Example: `syslog.suffix=local`

Connector Properties and Property Files

This applies to: *Visual Data Discovery*

Symphony's architecture enables the deployment of Symphony's data connectors as standalone components running in their own process space. Each connector has its own dedicated connector server and a corresponding property files. Some properties are common to all connectors and some properties are unique to a specific connector. The properties for each connector are documented in the property files.

The kinds of configuration properties found in these files include:

- logging properties
- actuator properties (Spring Boot management endpoint configuration)
- data source connection pool properties
- data source-specific properties
- Kerberos configuration properties (for some connectors)
- service discovery properties
- other connector-specific properties.

Note: insightsoftware discourages changing properties in the `/opt/zoomdata/conf` directory (Linux) or `<install-path>/conf` (Windows). Copy the files you want to change to the `/etc/zoomdata` directory (Linux) or `<install-path>/conf-modify` (Windows) and change them there. This will ensure that your changes are not overwritten when Symphony is next upgraded.

Quickly determine what changes you've made to a properties file using `diff` in Linux. For example:

```
diff /opt/zoomdata/conf/edc-<connector-name>.properties /etc/zoomdata/<edc-<connector-name>.properties
```

or

```
diff /opt/zoomdata/conf/zoomdata.properties /etc/zoomdata/zoomdata.properties
```

For Windows environments, use your preferred diff utility to compare the differences between your original and updated property files.

The connector property files have names in the following format: `edc-<connector>.properties`. The following table lists these property files and identifies the associated connector.

Connector	Property File Name
Amazon Redshift	`edc-redshift.properties`
Amazon S3	`edc-s3.properties`
Apache Drill	`edc-drill.properties`
Apache Phoenix 4.7	`edc-phoenix-4.7.properties`
Apache Phoenix Query Server 4.7	`edc-phoenix-4.7-queryserver.properties`
Apache Solr	`edc-apache-solr.properties`
BigQuery	`edc-bigquery.properties`
Cloudera Impala	`edc-impala.properties`
Cloudera Search	`edc-cloudera-search.properties`
Couchbase	`edc-couchbase.properties`
Dremio	`edc-dremio.properties`
Elasticsearch 7.0	`edc-elasticsearch-7.0.properties`
Elasticsearch 8.0	`edc-elasticsearch-8.0.properties`
HDFS	`edc-hdfs.properties`
Hive	`edc-hive.properties`
Jira	`edc-jira.properties`
MemSQL	`edc-memsql.properties`
Microsoft SQL Server	`edc-mssql.properties`
MongoDB	`edc-mongo.properties`
MySQL	`edc-mysql.properties`
Oracle	`edc-oracle.properties`
PostgreSQL	`edc-postgresql.properties`
Python	`edc-python.properties`
Real-Time Sales	`edc-rts.properties`
Salesforce	`edc-salesforce.properties`
SAP Hana	`edc-saphana.properties`

Connector	Property File Name
SAP S/4HANA	`edc-saphanacloud.properties`
SAP IQ	`edc-sapiq.properties`
Snowflake	`edc-snowflake.properties`
Spark SQL	`edc-sparksql.properties`
Teradata	`edc-teradata.properties`
TIBCO DV	`edc-tibcodv.properties`
Trino	`edc-trino.properties`
Vertica	`edc-vertica.properties`

For more information about editing property files, see Edit A Data Discovery Configuration File.

Configure Memory Settings

This applies to: *Visual Data Discovery*

Unified service memory configuration recommendations can help you avoid out of memory issues, unpredictable service stability, and memory allocation. We strongly recommend you use the new default for Java: Xms=Xmx. These updated default memory allocations are designed configure your services more effectively for a production load after installation or upgrade.

Note: Symphony requires slightly higher memory expectations than previous releases. Upgrading will not overwrite your memory configuration if you have already overwritten the default configuration settings.

If you're upgrading your environment:

- Review your data source connector usage; stop unused connectors.
- Review your memory settings based on the chart below.

	7.x / 23.1 Xms/Xmx	23.2 and later Xms/Xmx
Admin Server	512M/1024M	Deprecated
Config Server	512M/1024M	Deprecated
Data Writer	128M/256M	512M/512M
EDC *	128M/512M	512M/512M
EDC hdfs/s3	512M/3584M	1500M/1500M
Query Engine	1G/6G	4G/4G
Screenshot Service	128M/1G	750M/750M
Zoomdata Web	1G/2G	4G/4G

Change memory allocation settings for Symphony microservices

1. From your terminal, SSH to your Symphony server.
2. To modify the memory settings for Symphony microservices, you need to edit or create the corresponding `.jvm` files in `/etc/zoomdata/`. Perform the following steps:

```
vi/etc/zoomdata/<component_name>.jvm
```

For complete information on editing configuration files, see Edit A Data Discovery Configuration File.

3. Add or update the following line(s) in the corresponding `.jvm` files.

   Important: Do not allocate more than 85% of your total system memory to all microservices.

   The following example configures the Symphony server to use 20 GB of RAM in the `zoomdata.jvm` configuration file. You can adjust the number to fit your system's needs. Replace the **20** with the necessary memory allocation for your operating environment.

   ```
   -Xms20g
   -Xmx20g
   ```

4. Save and exit the `.jvm` file.

5. Restart the microservice for which you have modified the settings:

   For CentOS and Ubuntu 18, 20, or 22:

   ```
   sudo systemctl restart <service-name>
   ```

Wait a few minutes for the microservice to restart completely, then open a new browser window to log back into Symphony.

Manage the Symphony Query Engine

This applies to: *Visual Data Discovery*

The query engine is a stand-alone microservice within the Symphony environment that processes visual queries. If you install or upgrade Symphony using the supplied installation script, the query engine microservice is started automatically. If you install or upgrade Symphony manually, you must manually enable and start the query engine microservice.

Configure the Query Engine

In your environment, the query engine is called `zoomdata-query-engine`. To configure and manage the microservices that the query engine executes and runs, you need to make changes to the `query-engine.properties` file. It contains properties that are specific to how the query engine operates within your environment.

The query engine also has a `query-engine.env` file and a `query-engine.jvm` file. You can edit these files to configure the following:

- To define the environment variables that are visible for the microservice, edit the `query-engine.env` file.
- To configure the JVM options that are used to start up Symphony microservices, edit the `query-engine.jvm` file.

The default memory configurations give the query engine is 4 GB of heap memory. During microservice startup, the query engine consumes at least 4 GB of memory (plus some off heap memory). You can alter this using the following parameters in the query-engine.jvm file, i.e. by increasing to 6Gb.

```
-Xms6g
-Xmx6g
```

For information on editing configuration files, see Edit A Data Discovery Configuration File. For information about query engine properties, see Query Engine Properties.

Pass Sensitive Data Using Linux Environment Variables

This applies to: *Visual Data Discovery*

You can pass sensitive property values (such as database passwords or user names) to Symphony using Linux environment variables, rather than hard coding the values in Symphony property files. Using the SpringBoot Java application's ability to consume application properties as environment variables, you can pass sensitive data to Symphony without showing the data as plain text in the Symphony property files.

Note: The information provided here requires a strong knowledge of Linux internals.

You can do this using either of two methods:

- Method 1: Use Systemd Override Files (Less Secure)
- Method 2: Use Parameters Passed To Microservices (More Secure)

You can also encrypt sensitive property values. See Encrypt Configuration Properties.

Method 1: Use `systemd` Override Files (Less Secure)

You can use `systemd` with default unit files override. This is less secure than Method 2: Use Parameters Passed To Microservices (More Secure) because the passwords are still exposed in the `env-pass.conf` files.

Complete the following steps:

1. Stop the affected microservices. See Stop Symphony Microservices.

```
sudo systemctl stop <service>
```

2. Add a `systemd` override file, called `env-pass.conf`, for each affected microservice. For example, the `systemd` override file for the `zoomdata` microservice would be `/etc/systemd/system/zoomdata.service.d/env-pass.conf` and the override file for the `zoomdata-query-engine` microservice would be `/etc/systemd/system/zoomdata-query-engine.service.d/env-pass.conf`. A complete list of microservice names is provided in ComposerSymphony Data Discovery Microservice Name Reference.

3. Edit the `env-pass.conf` file for each microservice and add lines specifying environment variable values for every microservice configuration property containing sensitive information. For example, the following environment variables specify the data store password, the upload destination password, and the keyset destination password in the `env-pass.conf` file for the `zoomdata` microservice:

```
[Service]
Environment="SPRING_DATASOURCE_PASSWORD=<data store password>"
Environment="UPLOAD_DESTINATION_PARAMS_PASSWORD=<upload destination password>"
Environment="KEYSET_DESTINATION_PARAMS_PASSWORD=<keyset destination password>"
```

In the `env-pass.conf` file for the `zoomdata-query-engine` microservice, you might add the following environment variable to specify the query engine database password:

```
[Service]
Environment="SPRING_QE_DATASOURCE_PASSWORD=<query engine database password>"
```

The environment variable names are the same as the property names in the corresponding microservice property files, but in all capital letters and substituting underscores for the periods in the property names. Review the property files for valid property names. For example, the environment variable name for the `spring.datasource.password` property is `SPRING_DATASOURCE_PASSWORD`.

4. Apply the changes to `systemd`:

```
sudo systemctl daemon-reload
```

5. Start the affected microservices. See Start Symphony Microservices.

Method 2: Use Parameters Passed to Microservices (More Secure)

The most secure method is to pass sensitive parameters to Symphony microservices as environment variables when you start the microservices. After the microservice is started, the settings for the sensitive parameters are no longer visible.

Important: If you use this method, Linux service management will be unable to automatically start or restart the affected Symphony microservices when your system or server restarts. To resolve this, develop a wrapper script for `systemd` that will automatically export the required environment variables and restart the microservices.

Complete the following steps:

1. Stop the affected Symphony microservice. See Stop Symphony Microservices.

```
sudo systemctl stop <service>
```

2. Disable the affected microservice. See Disable Symphony Microservices.

```
sudo systemctl disable <service>
```

3. Start the affected microservice, passing the sensitive parameters in environment variables as command line arguments. Make sure you use appropriate escape quotes if your password includes quotes.

   The environment variable names are the same as the property names in the corresponding microservice property files, but in all capital letters and substituting underscores for the periods in the property names. Review the property files for valid property names. For example, the environment variable name for the `spring.datasource.password` property is `SPRING_DATASOURCE_PASSWORD`.

   In the following start command, the `zoomdata` microservice is started and the data store password, the upload destination password, and the keyset destination password are all passed as command line arguments:

```
sudo -u zoomdata /opt/zoomdata/bin/zoomdata start -v
 -E 'SPRING_DATASOURCE_PASSWORD=<data store password>'
 -E 'UPLOAD_DESTINATION_PARAMS_PASSWORD=<upload destination password>'
 -E 'KEYSET_DESTINATION_PARAMS_PASSWORD=<keyset destination password>'
```

   The following start command starts the `zoomdata-query-engine` microservice and passes the query engine database password as a command line argument.

```
/opt/zoomdata/bin/zoomdata-query-engine start -v -E 'SPRING_QE_DATASOURCE_PASSWORD=<password>'
```

   See Start Symphony Microservices.

Encrypt Configuration Properties

This applies to: *Visual Data Discovery*

You can encrypt sensitive property values in Symphony's property files, if needed. This can be accomplished using the Spring Cloud CLI.

- Prerequisites
- Encrypting A Property
- Sample Script To Encrypt A Property

See also Change The Encryption Mode.

You can also pass sensitive property values to Composer using Linux environment variables, rather than hard coding the values in Symphony property files. See Pass Sensitive Data Using Linux Environment Variables.

Prerequisites

Before you can encrypt property values, your Symphony environment must meet the following requirements.

- You must have the full-strength Java Cryptography Extension (JCE) installed in your Java virtual machine (it's not there by default). You can download the JCE Unlimited Strength Jurisdiction Policy Files from Oracle at the following link: https://www.oracle.com/java/technologies/javase-jce8-downloads.html.

Follow the installation instructions in the README.txt file (essentially replacing the two policy files in the JRE `lib/security` directory with the ones that you downloaded).

- Install or upgrade to Spring Cloud CLI version 2.0.0. This specific version is required. Review the installation and upgrade guidance here: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#cli. To install the Spring Cloud CLI, follow the instructions here: https://cloud.spring.io/spring-cloud-static/spring-cloud-cli/2.0.0.RELEASE/single/spring-cloud-cli.html.

Encrypting a Property

Encrypt an individual configuration property

1. Ensure your environment meets the prerequisites listed in Prerequisites.
2. Use the following Spring Cloud CLI 2.0.0 command to encrypt a property value with a specified encryption key:

```
spring encrypt <property-value> --key <encryption-key>
```

Important: Version 2.0.0 of the Spring Cloud CLI must be installed. Other versions are not supported.

For example, the following command encrypts the value of the PASSWORD variable using the encryption key specified by the ENCRYPT_KEY variable.

```
spring encrypt $PASSWORD --key $ENCRYPT_KEY
```

The output of this command is the encrypted property. For example:

```
711448026e2c6a977b2be1b22f13649cc938366397fbd345113d2a50e27c348f)
```

3. Add the following to the properties file in which the encrypted property value will be stored:

i. The encrypted property you obtained in Step 2, prepended with `{cipher}`. The `{cipher}` prefix allows Spring Cloud to recognize encrypted properties. For example:

```
encrypted.property={cipher}711448026e2c6a977b2be1b22f13649cc938366397fbd345113d2a50e27c348f
```

ii. A new property (once per property file) that identifies the encryption key used. For example:

```
ENCRYPT.KEY=<encryption-key>
```

4. Save the properties file and restart its associated Symphony microservice. See Restart Symphony Microservices.

Alternatively, you could modify and use the sample script, provided next, to encrypt a property value in a properties file.

Sample Script to Encrypt a Property

You can modify and use the following sample script to encrypt a property value in a properties file.

```
#!/bin/bash

#define the property file location, property key, and encryption key
FILE=/etc/zoomdata/zoomdata.properties
PROPERTY_KEY=spring.datasource.password
ENCRYPT_KEY=zoomdata

#read the property value
PROPERTY_VALUE=$(< "$FILE" grep -w "$PROPERTY_KEY"|cut -d '=' -f2)

#encrypt the value
#NOTE: this step will fail with 'Unable to initialize due to invalid secret key'
#if the JCE prerequisite is not met
ENCRYPTED_VALUE="{cipher}"$(spring encrypt $PROPERTY_VALUE --key $ENCRYPT_KEY)

#update the file
sed -i "s/$PROPERTY_KEY=$PROPERTY_VALUE/$PROPERTY_KEY=$ENCRYPTED_VALUE/g" $FILE

#add the encryption key if not already present in the file
grep -q -F "encrypt.key=$ENCRYPT_KEY" $FILE || echo "encrypt.key=$ENCRYPT_KEY" >> $FILE
```

Change the Encryption Mode

This applies to: *Visual Data Discovery*

You can change the encryption mode used by Symphony (for example to change from AES to AES/CBC/PKCS5Padding encryption). The encryption mode you select is used to encrypt connection parameters, secure user attributes, and Trusted Access tokens.

Important: We recommend that you change the encryption mode used by Symphony with the assistance of Symphony Technical Support.

If you are upgrading to a newer version of Symphony and you also want to change your encryption mode, perform the upgrade first and then complete the steps described here.

Note: You must have system administration privileges to change the encryption mode.

You must have the full-strength Java Cryptography Extension (JCE) installed in your Java virtual machine (it's not there by default). You can download the JCE Unlimited Strength Jurisdiction Policy Files from Oracle at the following link: https://www.oracle.com/java/technologies/javase-jce8-downloads.html.

See also Encrypt Configuration Properties.

Change the encryption mode

1. Start the Symphony microservice. This will populate the Symphony database using the original encryption (for example AES). See Start Symphony Microservices.
2. Stop the Symphony microservice. See Stop Symphony Microservices.
3. Back up the Symphony database. See Back Up The Metadata Store.
4. Modify the following encryption properties in the `zoomdata.properties` file: `security.encryption.algorithm` and `security.encryption.key.algorithm`. For example:

   ```
   security.encryption.algorithm=AES/CBC/PKCS5Padding
   security.encryption.key.algorithm=AES
   ```

   See Zoomdata.properties Properties.
5. Start the Symphony microservice. Symphony will start using the new properties and the new encryption method. See Start Symphony Microservices.

Create a Symmetric Key to Encrypt Data Source Passwords

This applies to: *Visual Data Discovery*

Symphony provides a suite of prebuilt connectors that connect the Symphony Server directly to your data source. If a data store requires a connection password to access the data, the credential information is saved in Symphony's storage repository - PostgreSQL. Symphony uses symmetric encryption to store the credential information so that it can access the data store, as needed, while providing a level of security for the saved information.

Symphony administrators can generate their own KeyStore using a symmetric key algorithm. This capability provides an additional level of security in the connection to and access of the data sources.

A symmetric key can be generated using Oracle's keytool program, which is a key and certificate management tool. This tool manages a keystore (database) of cryptographic keys, X.509 certificate chains, and trusted certificates. Refer to Oracle documentation for additional details about this keytool program.

Use the latest Java SDK to install the keytool program (as older versions of the SDK may require different installation steps).

Note: Remember that this user-generated keystore should be provided to Symphony after a new installation, prior to any connections being stored in Symphony. If a new user-generated key is provided after some connections are already stored, the passwords for these connections have to be resupplied to Symphony after the new key is provided.

Generate a Keystore with a Symmetric Key

1. Install the keytool program. Use the latest Java SDK to install the keytool program.
2. Enter the following command line to generate your symmetric key.

```
keytool -genseckey -alias <YourKeyAlias> -keyalg AES -keysize 256 -storetype jceks -keystore <YourKeyStoreName>.jks
```

3. Create a keystore password and press **Enter** to continue.
4. Create a key password and press **Enter** to continue.
5. Store the keystore file in a location where the Symphony Server can access. For example:

```
/etc/zoomdata/<YourKeyStoreName>.jks
```

Next, you need to edit the `zoomdata.properties` file to add in the parameters needed for Symphony to integrate your symmetric key. If you have already logged into Symphony, be sure to log out first and close the browser.

6. Edit (or create) the Symphony configuration file (`zoomdata.properties`):

```
vi /etc/zoomdata/zoomdata.properties
```

> **Note:** If the configuration file does not exist, this command creates it.

7. Incorporate instructions for accessing your newly generated keystore file into the **.properties** file as provided below:

```
keystore.location=file:/etc/zoomdata/<YourKeyStoreName>.jks
keystore.password=<YourKeyStorePassword>
keystore.key.alias=<YourKeyAlias>
keystore.key.password=<YourKeyPassword>
```

8. Restart Symphony Server. This ensures that the new keystore file is enabled and active within Symphony.

   For the appropriate Linux commands, see Restart Symphony Microservices.

The symmetric key should now be active in Symphony. If you see any error messages after the restart, submit a request for assistance.

Set Up Unified Logging Using Fluentd

This applies to: *Visual Data Discovery*

In Symphony, you can use Fluentd as a logging layer to which you can direct the logs for various components of Symphony. This allows you to customize the log output to meet the needs of your environment.

Symphony leverages Fluentd's unified logging layer to collect logs via a central API. Fluentd can be configured to aggregate logs to various data sources or outputs. For example, if you are directing all log files from your `zoomdata-websocket.log` and `zoomdata-errors.log` to a Fluentd server, you can add one of Fluentd's plug-ins to write the log files to Elasticsearch to analyze web client errors for your environment.

Unified logging does not replace Symphony's default logging architecture. It only augments that experience with an option for those wanting additional logging control.

By default, unified logging is disabled and needs to be enabled for each microservice you want captured in Fluentd.

If written to a data store compatible with Symphony's connectors, Symphony can then be used to analyze and visualize your logs. The selected data store must be supported by Symphony's connectors if you plan to view the log files in Symphony dashboards. A list of supported data stores is provided in Data Connector Reference. The selected data store should be available and configured in conjunction with Fluentd to receive logs.

At a high level, the steps to set up Fluentd are as follows:

1. Set up a database to store the Fluentd log files. Typically, customers store log files in a log capture service such as Elasticsearch or in a database server such as PostgreSQL.

2. Configure Fluentd logging for your Symphony installation. See Configure Unified Logging Using Fluentd. This includes configuring the Fluentd `td-agent.conf` file to identify the Symphony microservice log data you want logged.

3. Enable unified logging and configure the host and port information for your Fluentd server for each microservice log file from which you are extracting data for Fluentd logging. See Enable Unified Logging In Symphony Using Fluentd.

Configure Unified Logging Using Fluentd

This applies to: *Visual Data Discovery*

For information and steps on installing Fluentd, refer to Fluentd's installation documentation. If Fluentd is not installed on the same server as the Symphony, you will need to obtain the host and port numbers for the Fluentd server.

This section provides describes how

to configure Fluentd to log Symphony log data.

Configure Fluentd for logging Symphony log data:

1. Run the following command on the server where Symphony is installed to install `td-agent` on that server.

   ```
   curl -L https://toolbelt.treasuredata.com/sh/install-redhat-td-agent3.sh
   ```

2. Browse to the `/etc` folder on the Symphony server to verify that `td-agent` is installed in the `/etc/td-agent` folder.

3. Install the Fluentd plugin for the data store you are using to store your Fluentd logs. Refer to your Fluentd documentation for more information.

   For example, if you are using an Elasticsearch database to store Fluentd logs. run the following command to install the Fluentd Elasticsearch plugin on the Symphony server.

   ```
   sudo td-agent-gem install fluent-plugin-elasticsearch
   ```

4. Modify the `td-agent.conf` file in the `/etc/td-agent` folder using the following template.

   The following is an example of `td-agent.conf` using an Elasticsearch database for Fluentd logs.

   ```
   <source>
      @type forward
      port <fluentd_port>
      bind 0.0.0.0
   </source>
   <match *.**>
      @type copy
      <store>
         @type elasticsearch
   ```

```
      host <elasticsearch_host>
      port <elasticsearch_port>
      user <elasticsearch-user>
      password <elasticsearch-password>
      logstash_format true
      logstash_prefix composer-unified-log
      logstash_dateformat %Y%m%d
      include_tag_key true
      index_name composer-unified-log
      type_name composer-unified-log
      tag_key @log_name
      flush_interval 1s
   </store>
   <store>
      @type stdout
   </store>
</match>
```

5. Restart `td-agent`.

```
systemctl restart td-agent
```

6. Complete the steps described inEnable Unified Logging In Symphony Using Fluentd to enable Fluentd logging by Symphony.

Enable Unified Logging in Symphony Using Fluentd

This applies to: *Visual Data Discovery*

Identify the Symphony microservices for which you want activity logged to Fluentd. A complete list of the Symphony microservices is given in ComposerSymphony Data Discovery Microservice Name Reference, but not all microservices support unified logging. You can enable unified logging for these microservices:

- Web microservices (`zoomdata.properties`)
- Query engine (`query-engine.properties`)
- Data source connectors (see Connector Properties And Property Files for a list of property files)
- Data Writer framework (`data-writer-postgresql.properties`)

Enable unified logging using Fluentd for a Symphony microservice

1. On your Symphony server, edit the `.properties` file for the microservice. For example, `etc/zoomdata/query-engine.properties`.
2. Define the following parameters in the `.properties` file:

Property Syntax	Description
`logging.unified.level = <log-level>`	Specify one of the following log levels for `<log-level>`: ERROR, WARN, INFO, DEBUG, TRACE, or OFF. If set to OFF, Fluentd unified logging is disabled.
`logging.unified.tag = <service-tag>`	Specify the unique tag for the Symphony microservice, used in grouping logs. This tag must be unique throughout Symphony. Supply the Symphony microservice name for `<service-tag>`. For example, `logging.unified.tag = query-engine`. This is important because the tag identifies which microservice the log message applies to. Valid values are `query-engine`, `zoomdata-server`, `data-writer-postgresql`, and `edc-<connector-name>` (where `<connector-name>` is one of the names listed in Connector Properties And Property Files).
`logging.unified.label = <service-label>`	Specify a more verbose label for the Symphony microservice.
`logging.unified.host = <yourFluentdServerIPaddress>`	Specify the IP address of the Fluentd logging server.

Property Syntax	Description
`logging.unified.port = <yourFluentdServerPort>`	Specify the port of the Fluentd logging server.

3. Save your changes and exit the `.properties` file.
4. Restart the Symphony server.

On the Fluentd server, you can then direct your logs to a data source of your choice. For information and steps, see Fluentd's documentation on Output.

Translate the Symphony Interface

This applies to: *Visual Data Discovery*

You can translate the Symphony interface into other languages. Symphony supports standard i18n languages. See https://perldoc.perl.org/I18N::LangTags::List#LIST-OF-LANGUAGES.

Note: To translate the interface, you must have a specific advanced setting switched on. Please contact technical support for information about this setting.

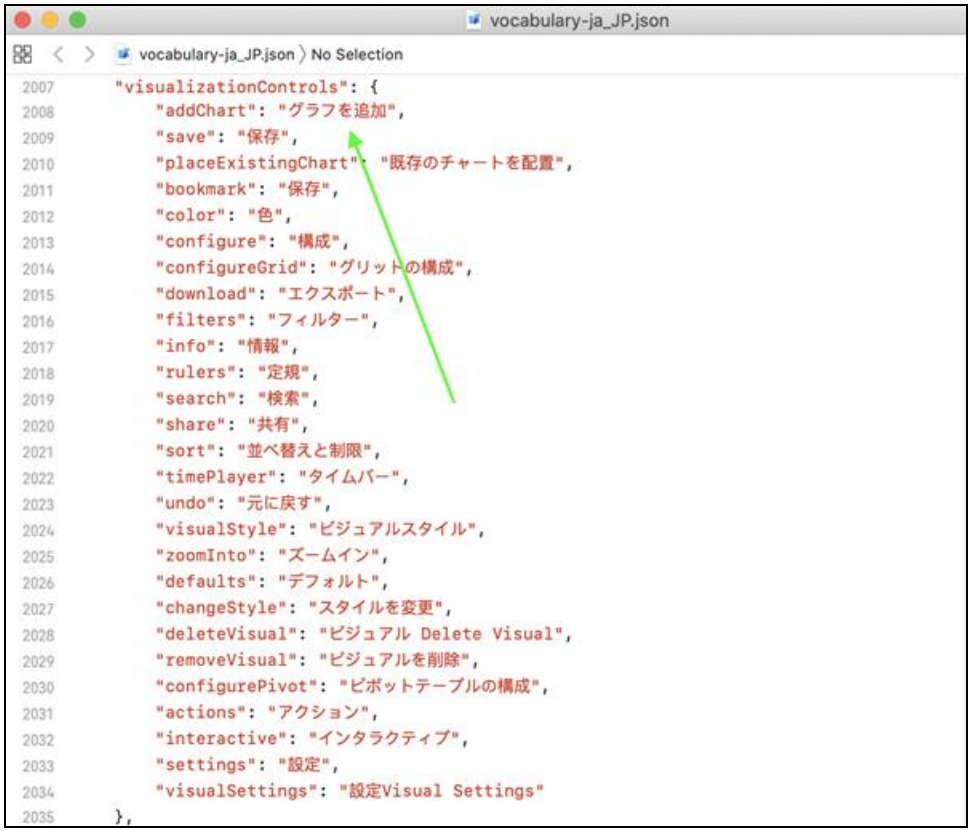
Translation files in the format `vocabulary-<xx>_<YY>.json` are used in the translation. The English translation file is provided and is named `vocabulary-en_US.json`.

- Linux Environments: Located in the `/opt/zoomdata/client/i18n` folder.
- Windows Environments: Located in the `<install-path>/client/i18n` folder.

Translate the Symphony interface:

1. Locate and copy the provided English translation file `vocabulary-en_US.json` in the appropriate `/client/i18n` folder.
2. Rename the copy of the file. Be sure to name it in the format `vocabulary-<xx>_<YY>.json`, where `<xx>` is a two-character code representing the language you want to use and `<YY>` is the dialect. See https://perldoc.perl.org/I18N::LangTags::List#LIST-OF-LANGUAGES.
3. Translate the JSON values to the right of the JSON keys (after the colon) in the file. All values should be specified in quotes. The JSON keys should not be translated because the keys are used by the Symphony code.

   Here is an example of part of a translated translation file:



This file is CSS-friendly. For example, `<b>` and `</b>` around a JSON value or part of a value indicate that the value should be bold.

Finally, placeholders (variables) are used throughout the file and should not be changed or removed, although they can be relocated within a JSON value, as needed for your translation. Placeholders appear surrounded by `%` symbols (for example, `%name%`) in the translation file. Symphony code inserts a value for these placeholders, based on where the JSON key is used in the product. For example, a dashboard name might be inserted for the placeholder `%name%` for one JSON key, but a visual name might be inserted for a different JSON key that also uses the `%name%` placeholder.

4. Save your translated file.

5. Contact your insightsoftware technical support representative for additional steps.

Note: If you import an exported source that has an associated translation file, you must re-upload the translation for that source.

Server-Level Variables

This applies to: *Visual Data Discovery*

Server-level variables can be viewed by Symphony administrators or members of the Supervisors group in the Server-Level Variables work area.

Important: Server-level variables are set during installation and must not be changed without understanding how the change affects or compromises your environment. If you must change a variable here, contact Technical Support. Do not add or delete server-level variables.

Select the **Advanced** menu option to access the Server-Level Variables work area.

Server-level variables are defined as key-value pairs. You can enable or disable the listed variables below if needed. Select **Save** to save and apply any changes you make in this work area.

Caution: Changing toggles or editing content other than as instructed in this work area may prevent your users from using various components of Symphony.

Key (Server-Level Variable)	Value	Description
allow-dashboard-sharing-within-tenant	false (default)	Enable or disable dashboard sharing options for sharing dashboards with groups and everyone within your environment. See Share a Dashboard with Users.
scheduled-report-file-drop	false (default)	Enable or disable users' ability to deliver a scheduled report to an SFTP location. When set to `true`, users can select an SFTP file location you have defined to accept the scheduled report. When set to `false`, users do not see an SFTP option for scheduled reports. Define the settings for your environment in `zoomdata.properties`. See Scheduled Dashboard Report PropertiesScheduled Data Discovery Dashboard Report Properties.
symphony-ai-sql-flow	blank (default)	Enable or disable users' ability to generate SQL statements. To enable, enter the appropriate Chatflow ID. This ID is shown as part of the URL when you have the chatflow open for editing in the Logi AI module. **Note:** Enter the appropriate Chatflow ID for SaaS or on-prem use.

Key (Server-Level Variable)	Value	Description
		Important: This feature is considered to be released in beta for your testing purposes. Workflows and features may change before a production-ready version is released.

Configure Symphony Logs

This applies to: *Visual Data Discovery*

The microservices used in Symphony write logs to their corresponding service's log files by default. You can configure Symphony to write these logs to the console only, or to write logs both to the console and the service's log files. Structured logging is also supported. See Structured Logging.

Every service config file includes two properties you can use to define where the logs are written:

- `log.console.level` - edit to define the types of logs to write to the console
- `log.file.level` - edit to define the types of logs to write to the files

Log Properties in Config Files

If you install using the bootstrap script, logging to files is enabled by default. You can reroute the logging to the console as needed to customize your installation.

> **Note:** If the value of log properties is set to `OFF`, Symphony access logs are not written. For all other cases (`ALL`, `ERROR`, `WARN`, `DEBUG`, and `INFO`) access logs are written to the selected destination: file, console or both.

- Bootstrap installation: The `*.jvm` file specifies `log.console.level=OFF` and `log.file.level=ALL` to support compatibility with earlier versions of Symphony.

Values for log.file.level and log.console.level

The following table lists possible values for `log.file.level` and `log.console.level`.

Value	Description
`ALL`	All available log information.
`ERROR`	Application error messages that may affect processes.
`WARN`	Unexpected application issues that may not affect processes.
`INFO`	Expected activities.
`DEBUG`	Triggers capture of `WARN`, `INFO`, and `ERROR`information.
`OFF`	Disables logging.

Enable Logging for a Service to the Console

Enable logging to the console and append the desired corresponding value of the properties to the appropriate service configuration file.

- Use `log.console.level=ALL` to route duplicates of all logs to the console.
- Use `log.console.level=ERROR` to route only duplicates of errors to the console.
- Alternatively, use `log.console.level=ALL` and `log.console.level=ERROR` to have all logs in files and errors duplicated to the console.

Adjust the values to meet your needs, then restart the service after you've edited the config.

Note: Symphony services have two config files, `*properties` and `*.jvm`. If you add log configuration properties to both files, the priority of `*.jvm` is higher than `*. properties`.

Consul Logging

By default, Consul writes logs to the console, but you can duplicate logs to files if needed.

Configure Consul Logs to Duplicate to Files:

1. Edit the Consul config file, `consul.json`. For Linux installations, this is located in `/opt/zoomdata/conf/consul.json`.
2. Add the following properties:

```
"log_file": "/opt/zoomdata/logs/zoomdata-consul.log",
"log_rotate_bytes": 104857600,
"log_rotate_max_files" : 5
```

Adjust the path to your log file and other log configuration properties as needed for your environment. In Linux environments, the log is located in `/opt/zoomdata/logs`, and in `<install-path>/logs` for Windows environments. See Consul documentation for more information.

Structured Logging

Symphony supports structured logging. You can enable for installations done via bootstrap, and is enabled by default for Kubernetes environments.

Components Support for Structured Logging

- zoomdata-web
- query engine
- admin server
- screenshot service
- data writer
- data connectors

Enable Structured Logging

In bootstrap environments, set the property `log.structured.enabled` to `true` in the config files of your microservices to output logs in JSON format to the `STDOUT` destination. Logs output to the `FILE` logs destination are sent in plain text. See Configuration Property Files.

In Kubernetes environments, structured logging is enabled by default. To disable, use one of the following approaches:

- Set the environment variable `LOG_STRUCTURED_ENABLED` to `false` to disable logging for specific services.
- Set `structuredLogsEnabled` in `values.yaml` to `false` to disable logging for all services.
- For Consul, change the `log_json` value to `false` in the `consul.server.extraConfig` section of `values.yaml`.

Enable or Disable a Security Service

This applies to: *Visual Data Discovery*

The Symphony supervisor can enable or disable a security authentication service.

Enable a Security Service

Enable a security authentication service

1. Log in as a system admin.
2. Select **Tools > Security** from the main menu.
3. The Security page appears. It consists of four tabs: **Security Services**, **SAML Settings**, **LDAP Settings**, and **Kerberos Settings**.
4. On the Security Services tab, turn on the appropriate switch (slide it to the right) for the authentication service you want to enable.

5. Select **Save**.

   After the settings have been saved, the status of the authentication service changes to **Will start on next restart**.
6. Close the Symphony UI.
7. Access the Linux prompt and log into your Symphony server (via Secure Shell or SSH).
8. Restart the Symphony server service. See Restart Symphony Microservices.

Wait a few minutes for the Symphony service to fully restart, then open a new browser window and log back in as the supervisor. Verify that your changes took effect. After restarting Symphony, the authentication service status changes to **Started**.

> **Note:** If you receive an error message stating that the connection to Symphony could not be made, you may need to give it a little more time for the service to fully restart. If the problem persists, contact Technical Support.

Disable a Security Service

Disable a security authentication service

1. Log in as a system admin.
2. Select **Tools > Security** from the main menu.
3. The Security page appears. It consists of four tabs: **Security Services**, **SAML Settings**, **LDAP Settings**, and **Kerberos Settings**.
4. On the Security Services tab, turn off the appropriate switch (slide it to the left) for the authentication service you want to disable.

5. Select **Save**.

   After the settings have been saved, the status of the authentication service changes to **Will stop on next restart**. The tab that allows you to configure the settings for the service will still be available and the service will keep running until the Symphony service is restarted.

6. Close the Symphony UI.

7. Access the Linux prompt and log into your Symphony server (via Secure Shell or SSH).

8. Restart the Symphony server service. See Restart Symphony Microservices.

   Wait a few minutes for the Symphony service to fully restart, then open a new browser window and log back in as the supervisor. Verify that your changes took effect. After restarting Symphony, the authentication service status changes to **Stopped** and access to the corresponding service tabs are disabled.

   > **Note:** If you receive an error message stating that the connection to Symphony could not be made, you may need to give it a little more time for the service to fully restart. If the problem persists, contact Technical Support.

Use Materialized Views (Experimental)

This applies to: *Visual Data Discovery*

Note: Materialized view functionality is disabled by default. To enable, contact technical support for assistance. The Materialized Views API is deprecated and will be removed in a future release.

Materialized views allow you to use pre-aggregated query results to speed up query processing in certain scenarios, especially when processing a query with heavily aggregated data. This boosts your visual rendering time.

Important: This is an experimental feature.

Materialized views only work if the metrics and groups in the materialized view definitions match those used by your visuals. For example, no benefit from using materialized views will occur if you define a materialized view for the Planned Sales metric in a data source, but none of the visuals in your environment use the Planned Sales metric. Consequently, it is a good idea to understand what data in your data source is used (or will be used) in visuals before you define a materialized view.

Note: A Count metric is required in all materialized view definitions.

Support for materialized views is performed using the REST API endpoint `/api/materialized-views`, described in Materialized Views API (Experimental). You can also specify materialized view settings for a data source configuration using the UI. See the following topics:

- List Materialized View Definitions
- Add A Materialized View Definition
- Edit A Materialized View Definition
- Delete A Materialized View Definition
- Enable And Disable Materialized View Definitions

List Materialized View Definitions

This applies to: *Visual Data Discovery*

Note: Materialized view functionality is disabled by default. To enable, contact technical support for assistance. The Materialized Views API is deprecated and will be removed in a future release.

Important: This is an experimental feature.

List the materialized view definitions for a data source configuration

1. Make sure you are logged in as a user with the **Administer Sources** privilege or that you have **read** and **write** permission for the data source.
2. Select the **Sources** option from the main menu. The Sources page appears.
3. Locate the data source configuration on the Sources page and select the More menu () icon in the **Actions** column to view the **More** menu.
4. Select **Materialized Views** from the menu.

   The Materialized View dialog appears, listing the materialized views defined for the data source configuration.

Add a Materialized View Definition

This applies to: *Visual Data Discovery*

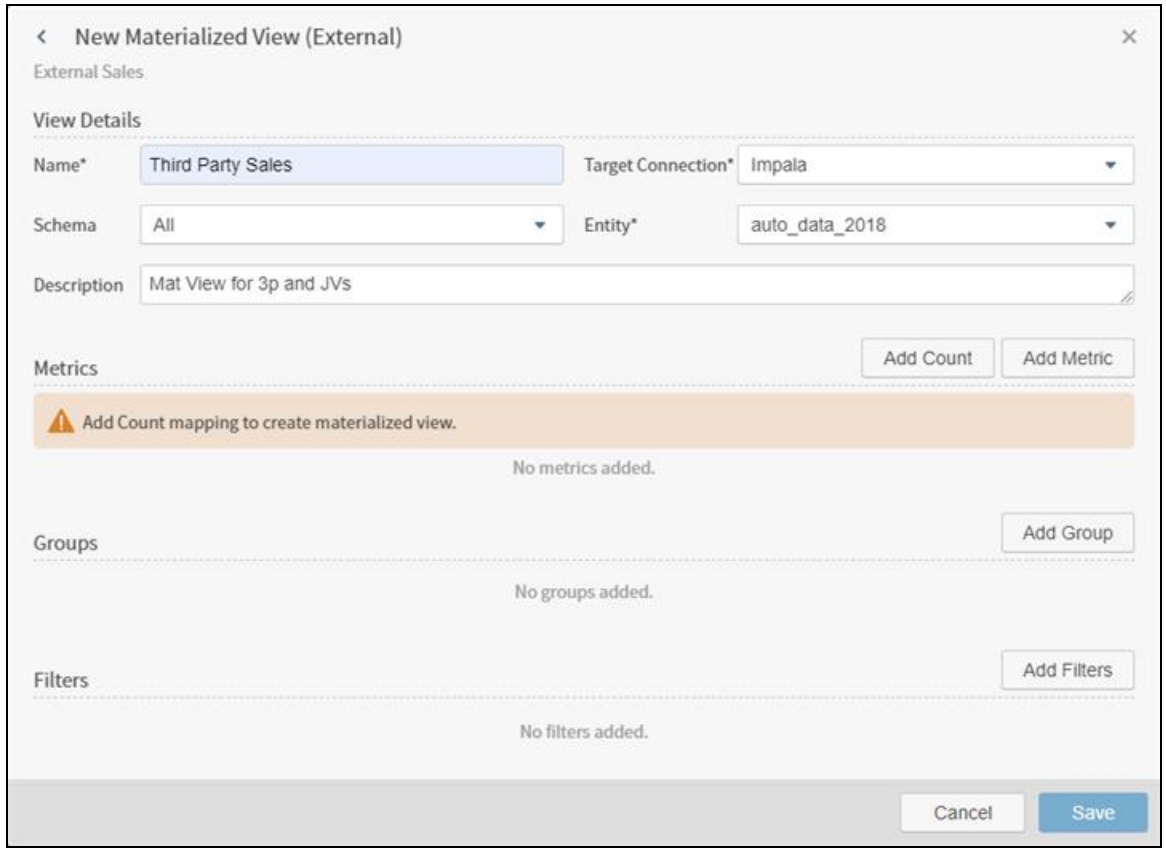
Note: Materialized view functionality is disabled by default. To enable, contact technical support for assistance. The Materialized Views API is deprecated and will be removed in a future release.

Important: This is an experimental feature.

Important: Pre-aggregated data is not managed by Symphony, so it should be maintained manually and kept up to date by the owner of the data.

Add a materialized view definition to a data source using the UI

1. Make sure you are logged in as a user with the **Administer Sources** privilege or that you have **read** and **write** permission for the data source. If you create in Symphony, you have **read**, **write**, and **delete** permissions for that source by default.
2. List the materialized views for the data source to which you want to add a materialized view. See List Materialized View Definitions.
3. Select **Add View**. The **New Materialized View (External)** dialog appears. This dialog is used to identify where the aggregated result from a query is stored so it can be quickly recalled in visuals.



4. Specify a name for the materialized view in the **Name** field. This is required.

5. Select an external target connection in the **Target Connection** field. This is required. The connections listed for this field match the list of connections defined for the Symphony instance. The target connection is not required to be the same as the connection of the data source; it can be any other connection configured in Symphony.

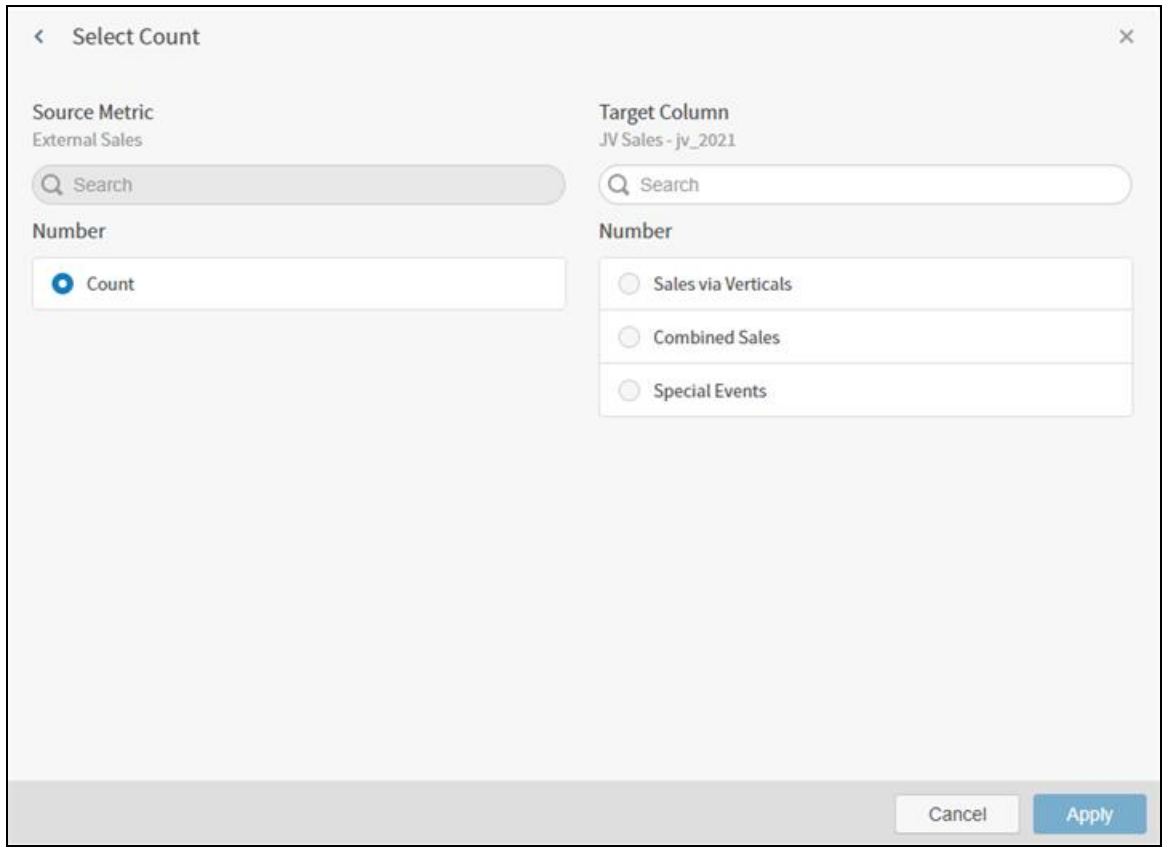
   Note: If you do not have read permissions for the target data store, an error message appears.

6. Use the **Schema** and **Entity** fields to identify the target entity that will store the aggregated data. The target entity is required, while the schema is not. However, selecting a schema filters the list of entities so you can more quickly find the entity you want.

   As soon as an entity is specified, the **Add Count**, **Add Metric**, **Add Group**, and **Add Filters** buttons become available on the dialog.

7. Optionally supply a description in the **Description** field.

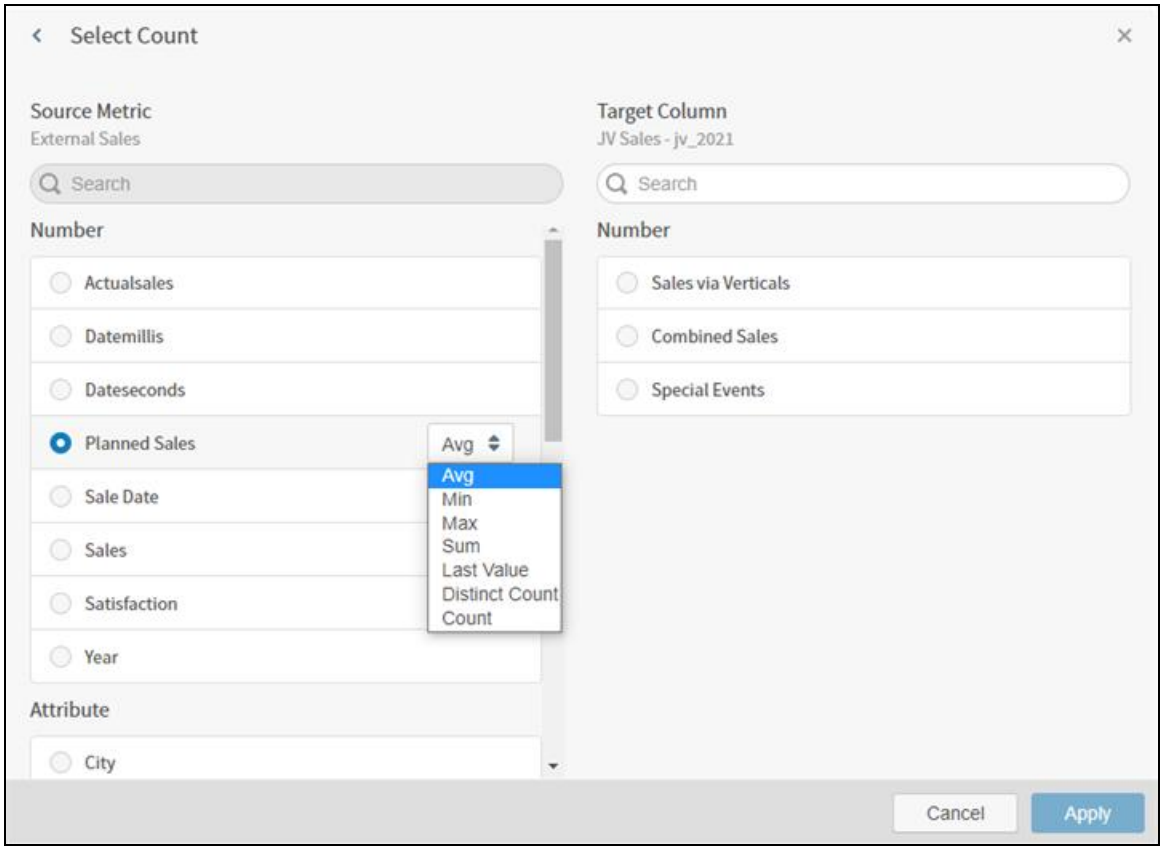
8. Select **Add Count** to open the Select Count work area. Select a source field in the Source Metric list and a Target Column to list which source field should be used, then select **Apply** to apply your changes.



9. Select **Add Metrics** to add metrics to your materialized view definition. The Count metric must be added because it is required. Other metrics are optional, but must match the metrics used in your visuals.

> **Note:** Materialized views only work if you specify the metrics and groups used by your visuals in the materialized view definition. For example, if you use Sales (SUM) as your metric and State as your group in a visual, be sure to add these metrics and groups to your materialized view definition. If they don't match, Symphony will not use the materialized view to boost your visual rendering time.

The Select Metric dialog appears.



Select a source field in the Source Metric list and a target column in the Target Column list to which the source field should be mapped. Aggregated data for the source field will be taken from the target column to provide the results of the query.

When you select a source field (other than Volume), you can also select a metric function to be used to aggregate the field data. Use the drop-down list to select the metric function. See Metric Aggregation Functions.

Use the search boxes at the top of the Source Metric and Target Column to quickly locate a source metric or target column in their lists.

Select **Apply** to apply the mapping to the materialized view definition.

Repeat this step for as many metrics as needed for the definition.

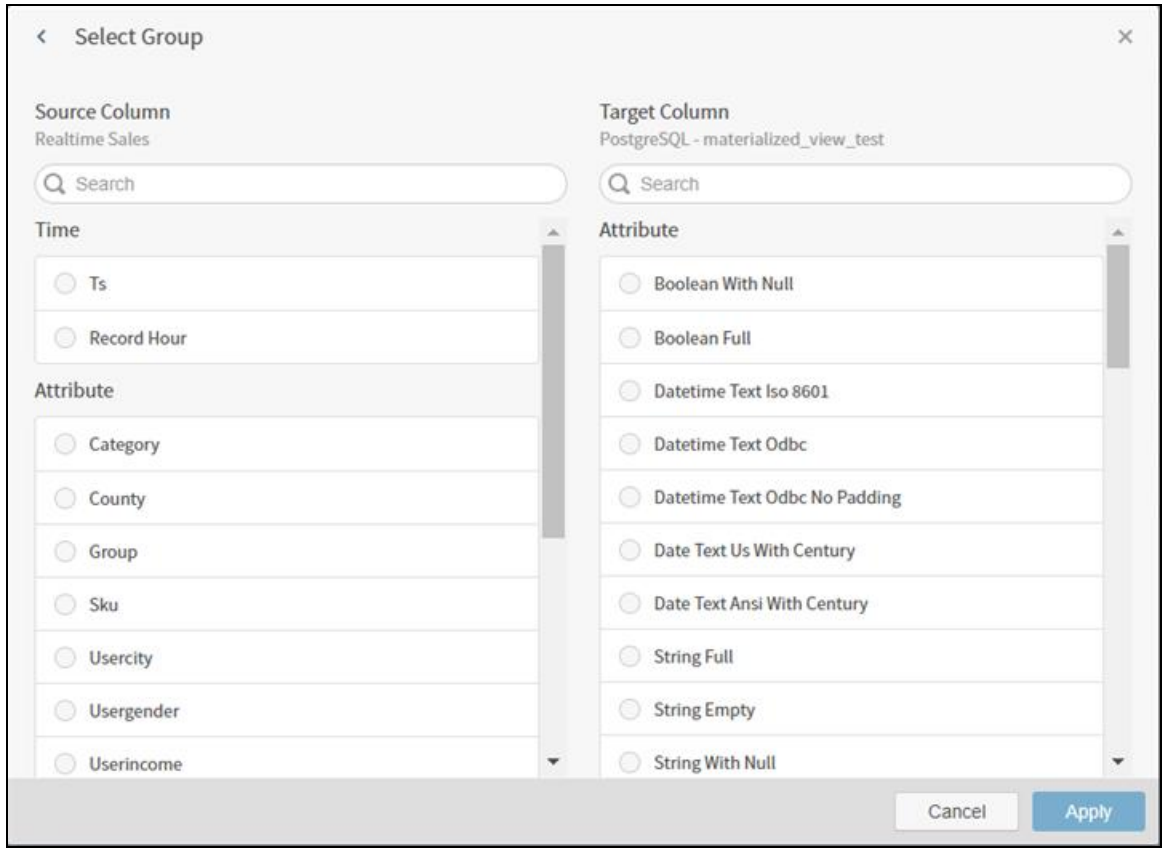
> **Note:** If you have two visuals that differ only in their metrics, and you have an externally stored table containing pre-aggregated data for both visuals, you can specify all of the metrics for both visuals in a single materialized view. When the visuals are rendered, both visuals will be matched by the same materialized view.

10. Select **Add Groups** to add groups to your materialized view definition. The groups must match the groups used in your visuals.

> **Important:** Be sure that the target entity contains correct data for this materialized view. If there are more pre-aggregated group columns in the target entity than configured groups in the materialized view, the data for the query will be taken as-is from the target entity. This might result in incorrect data shown on the visual, (for example, non-unique group values and incorrect metrics).

> **Note:** Materialized views only work if you specify the metrics and groups used by your visuals in the materialized view definition. For example, if you use Sales (SUM) as your metric and State as your group in a visual, be sure to add these metrics and groups to your materialized view definition. If they don't match, Symphony will not use the materialized view to boost your visual rendering time.

The Select Group dialog appears.



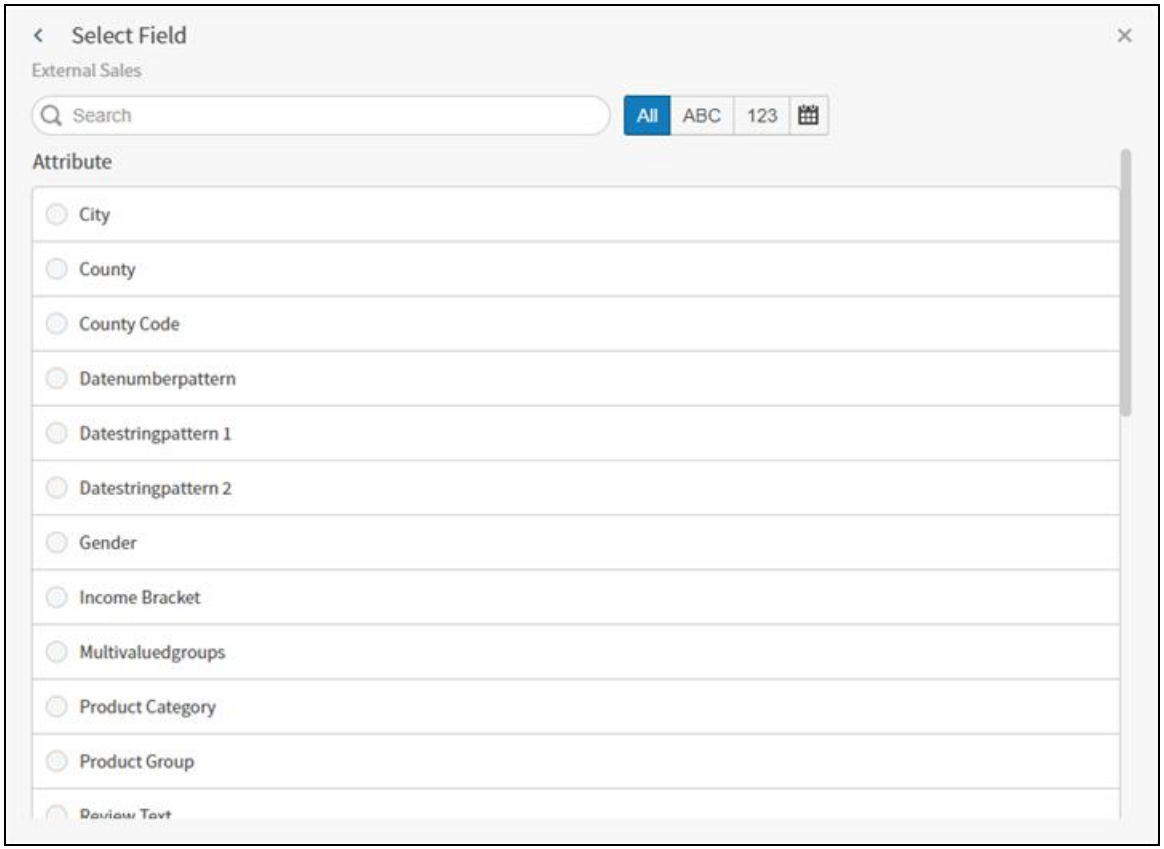
Select a source field in the Source Column list and a target column in the Target Column list to which the source field should be mapped. Aggregated data for the source field will be taken from the target column to provide the results of the query.

Use the search boxes at the top of the Source Column and Target Column to quickly locate a source field or target column in their lists.

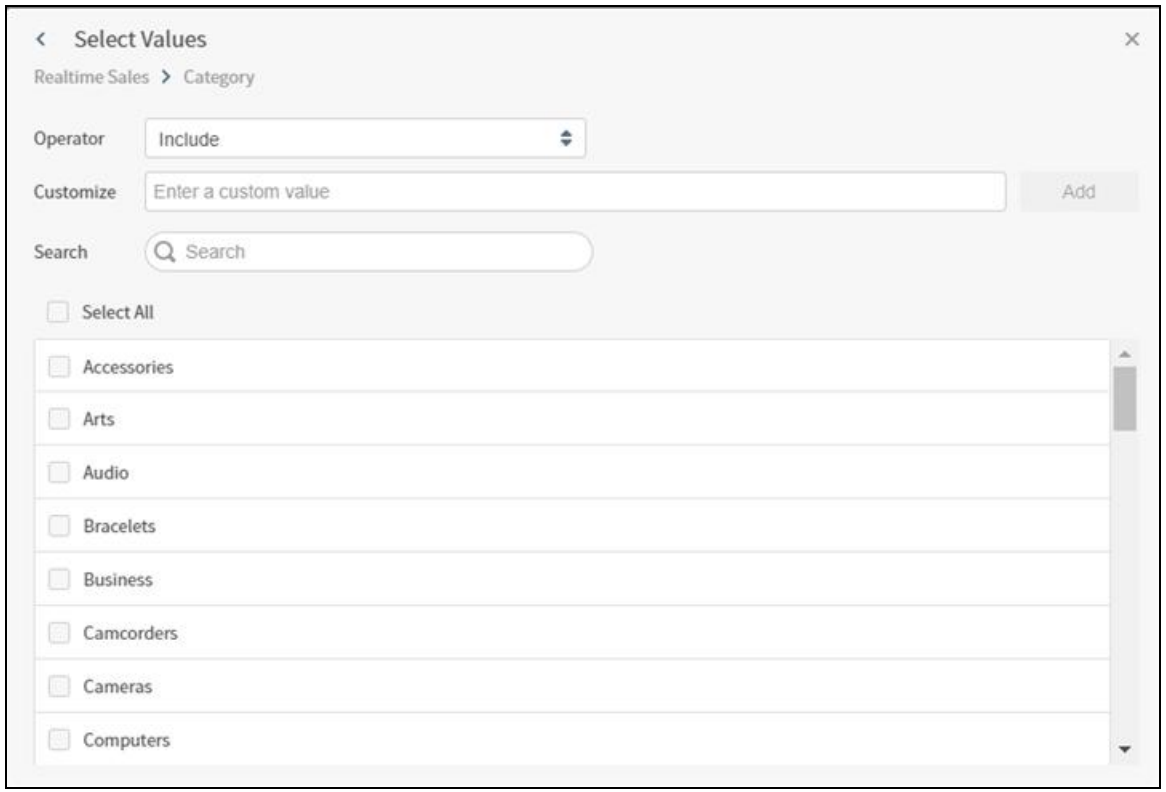
Select **Apply** to apply the mapping to the materialized view definition. Repeat this step for as many groups as needed for the definition.

11. Select **Add Filters** to add filters for the data that is stored in the target entity. Only requests that match the specified filters in the materialized view definition will be processed using the data from this materialized view.

    The Select Field dialog appears.


Select Field
External Sales
Search
All
ABC
123
Attribute
City
County
County Code
Datenumberpattern
Datestringpattern 1
Datestringpattern 2
Gender
Income Bracket
Multivaluedgroups
Product Category
Product Group

i. Select a source field for the filter. Use the search box at the top of the list to quickly locate a source field in the list. The Select Values dialog appears.



ii. Select an operator for the filter in the Operator drop-down menu.

iii. Optionally enter a custom value in the Customize field. The field name you supply is added and selected in the value list on the dialog.

iv. Select one or more values in the value list. Use the search box at the top of the list to quickly locate a value in the list.

12. Select **Save** to save the materialized view definition.

Edit a Materialized View Definition

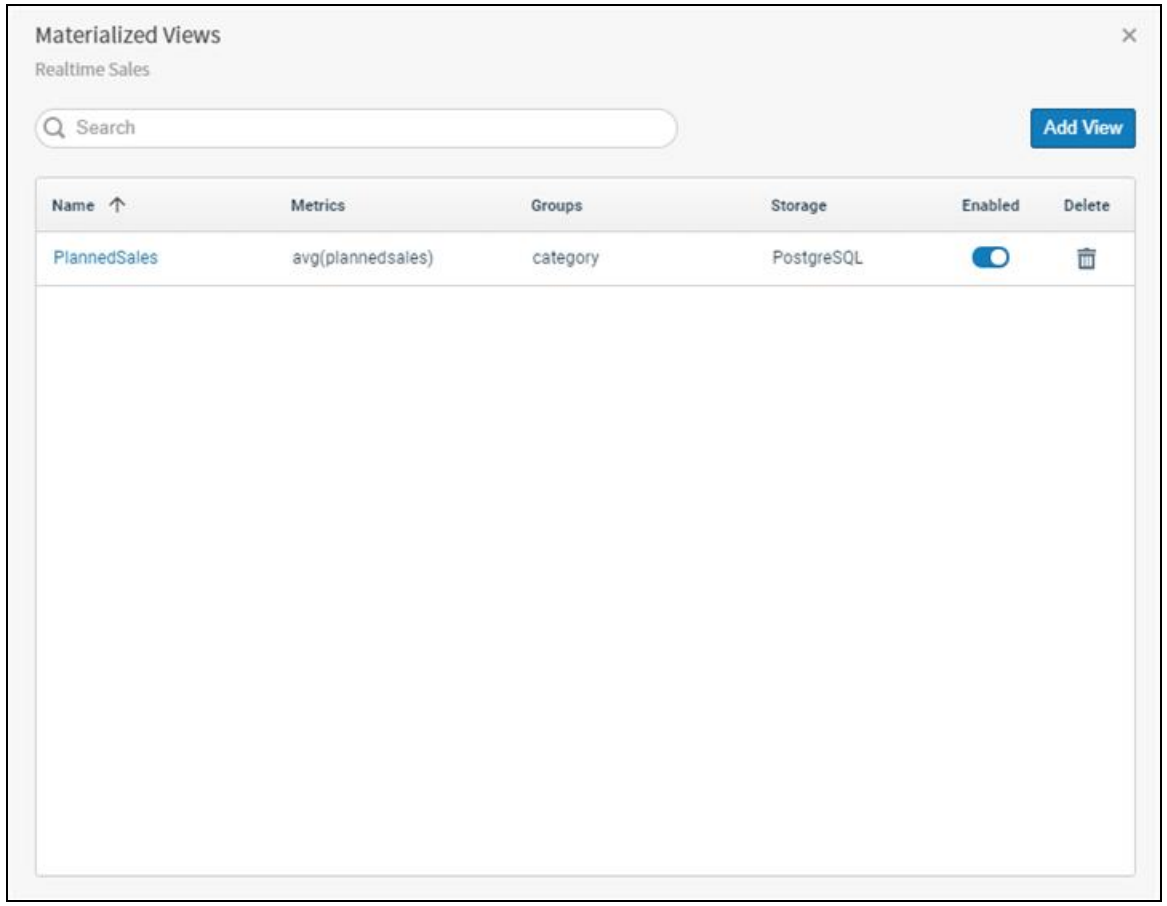
This applies to: *Visual Data Discovery*

Note: Materialized view functionality is disabled by default. To enable, contact technical support for assistance. The Materialized Views API is deprecated and will be removed in a future release.

Important: This is an experimental feature.

Edit a materialized view definition using the UI

1. Make sure you are logged in as a user with the **Administer Sources** privilege or that you have **read** and **write** permission for the data source.
2. List the materialized views for the data source. See List Materialized View Definitions.



3. Select the materialized view name in the list of materialized view definitions. The definition opens in a new dialog.

4. Modify any field in the definition. You can change the definition name, description and target settings. You can also add and remove metrics, groups, and filters from the definition. Remove metrics, group, and filter specifications by selecting ⊖ in the **Delete** column of the Metrics, Groups, and Filters tables.

   > **Note:** Materialized views only work if you specify the metrics and groups used by your visuals in the materialized view definition. For example, if you use Sales (SUM) as your metric and State as your group in a visual, be sure to add these metrics and groups to your materialized view definition. If they don't match, Symphony will not use the materialized view to boost your visual rendering time.

   See Add A Materialized View Definition for explanations of each field in the definition.

5. Select **Save** to save the materialized view definition.

Delete a Materialized View Definition

This applies to: *Visual Data Discovery*

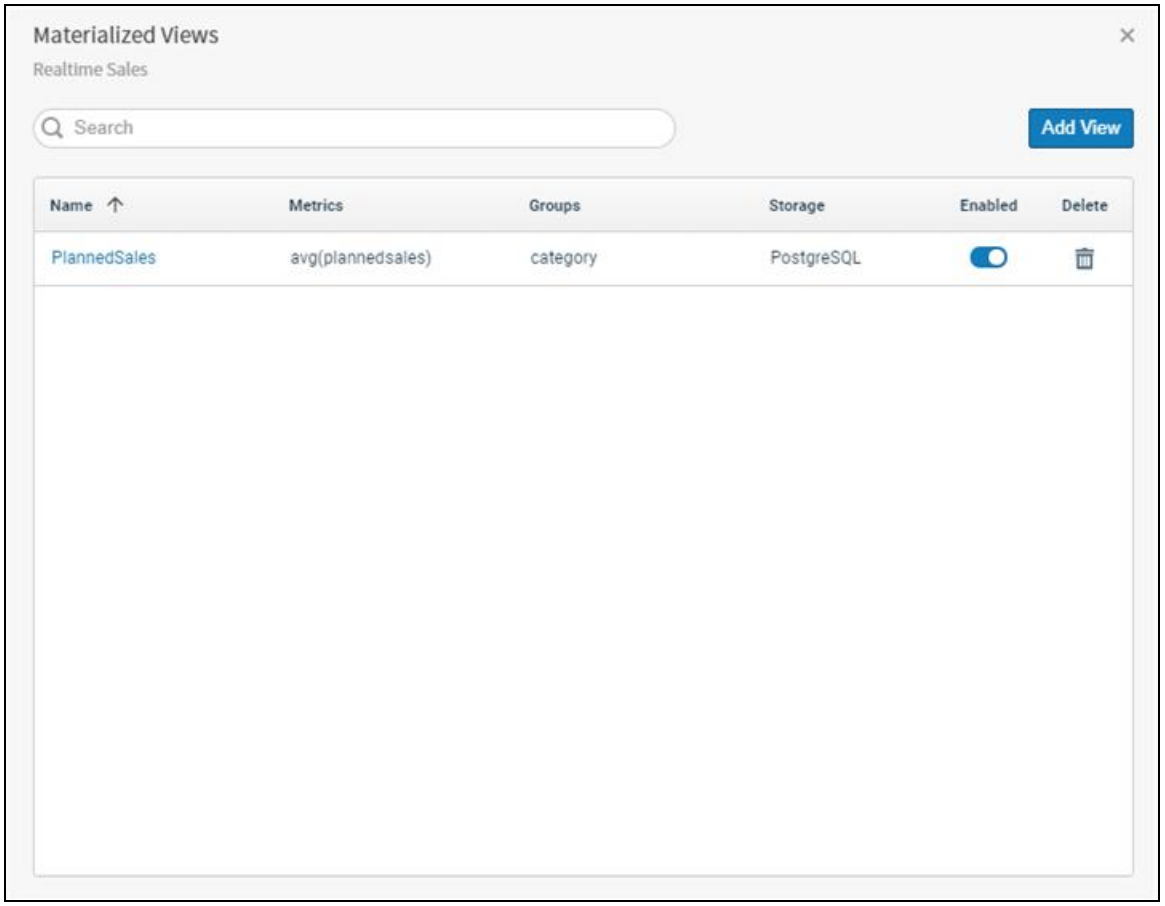
Note: Materialized view functionality is disabled by default. To enable, contact technical support for assistance. The Materialized Views API is deprecated and will be removed in a future release.

Important: This is an experimental feature.

Deleting a materialized view does not affect the underlying pre-aggregated data. When that data is not used any more, it should be deleted manually from you external storage by the data owner.

Delete a materialized view definition using the UI

1. Make sure you are logged in as a user with the **Administer Sources** privilege or that you have **read**, **write**, and permission for the data source.
2. List the materialized views for the data source. See List Materialized View Definitions.



3. Locate the materialized view definition you want to delete and select in the Delete column for the definition.

   A warning dialog appears, prompting you to confirm the deletion.

4. Select **Delete** on the warning dialog.

   The definition is deleted.

Enable and Disable Materialized View Definitions

This applies to: *Visual Data Discovery*

Note: Materialized view functionality is disabled by default. To enable, contact technical support for assistance. The Materialized Views API is deprecated and will be removed in a future release.

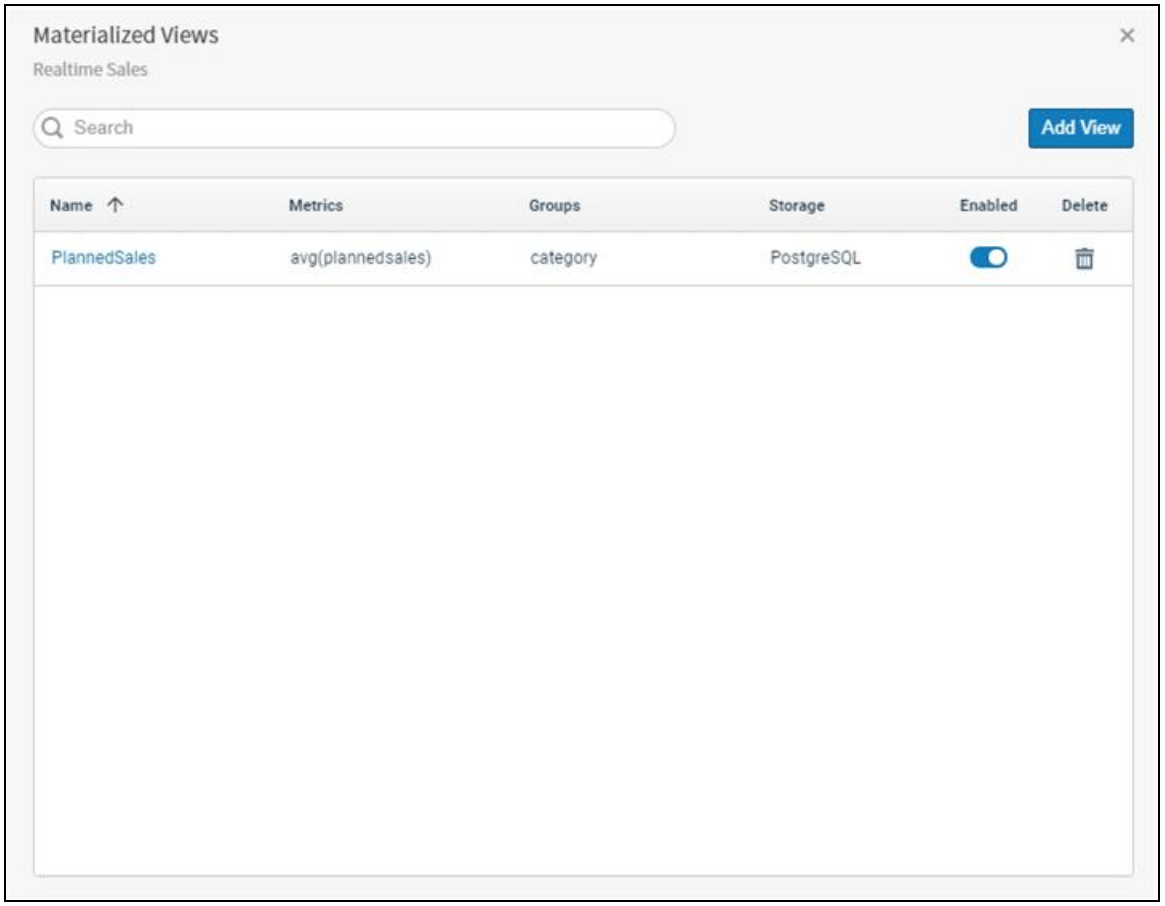
You can enable and disable individual materialized view definitions. When disabled, the stored query data for the materialized view is not used in visuals for the data source. Disabling a materialized view definition allows you to remove the effect of the definition without deleting it.

By default, individual materialized view definitions are enabled when they are defined.

Important: This is an experimental feature.

Disable a materialized view definition using the UI

1. Make sure you are logged in as a user with the **Administer Sources** or the **Create New Data Sources** privilege.
2. List the materialized views for the data source. See List Materialized View Definitions.

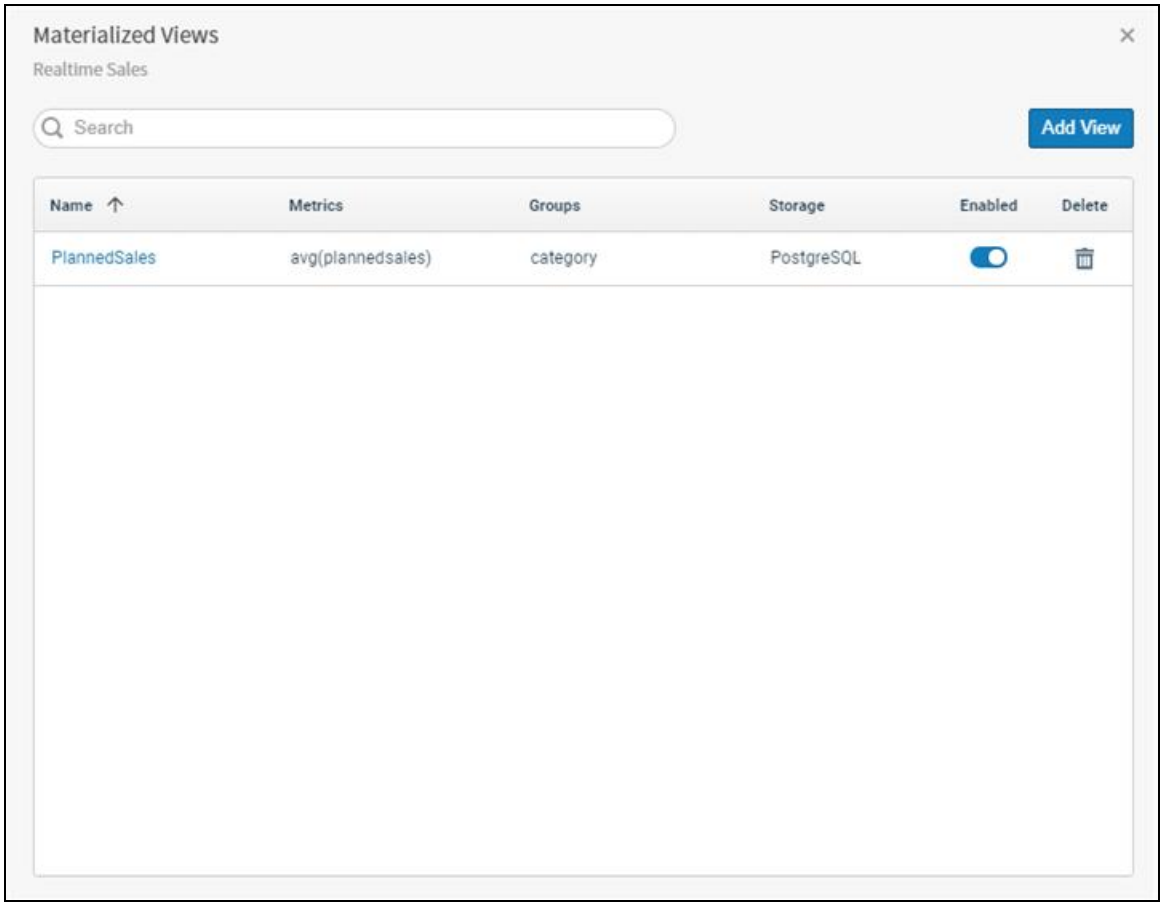


3. Locate the materialized view in the list of materialized view definitions and slide the switch in the **Enabled** column for the definition to the left (off).

   The definition is disabled.

Enable a materialized view definition using the UI

1. Make sure you are logged in as a user with the **Administer Sources** or the **Create New Data Sources** privilege.
2. List the materialized views for the data source. See List Materialized View Definitions.



3. Locate the materialized view in the list of materialized view definitions and slide the switch in the **Enabled** column for the definition to the right (on).

   The definition is enabled.