TOC

Composer 24

Install, Upgrade, and Remove Composer 24

System Requirements

Composer can be downloaded and installed in different Linux and Windows distributions. You can also deploy Composer in the cloud. To try Composer for free, check out our Free Trials page.

The client application is accessed using web browsers that support WebSocket technology. See caniuse.com to check whether your web browser supports WebSocket technology. The following requirements are needed in order to successfully deploy and access the Composer Server in your operating environment.

- Operating System And Software Requirements
- Hardware Requirements
- Metadata Repository
- Network Requirements

See also Installation Prerequisites.

Operating System and Software Requirements

For all deployments, a 64-bit OS is required. The Composer server also uses Java and PostgreSQL. Ensure that the appropriate version of these tools are used in the deployment of Composer in your operating environment.

The following table shows the operating systems supported, along with the corresponding minimum supported version.

Distribution	Supported Version
CentOS	9
Red Hat	7 & 8
Ubuntu	18, 20, 22
Amazon Linux	2
Windows Server	2012 R2, 2016, 2019 and higher

Note: CentOS Stream 9 is supported for new instances of Composer 24.3 and higher. Contact Support for upgrade assistance from earlier versions.

Support services required versions:

Service	Required Version
PostgreSQL	12
Java	11 for Composer version 23.1, 17 for Composer 23.2 and later

Hardware Requirements

The following server specifications are recommended:

- 64 GB RAM (minimum is 16 GB)
- 500 GB disk space
- 16 cores

See also Server Size Guidelines.

Metadata Repository

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

- Composer v24.2 and earlier: PostgreSQL 12
- Composer v24.3 and later: PostgreSQL 16

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

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

Network Requirements

Ports

The following table lists the ports you must consider when installing or upgrading Composer.

Composer Feature	Microservice	Default Port	Required?	Notes
Composer server	`zoomdata`	8080	✓	
Composer server	`zoomdata`	8443, 443	✗	Required only if you use HTTPS
Consul	`zoomdata-consul`	8500	✓	
Query Engine	`zoomdata-query-engine`	5580	✓	
Appropriate Composer connectors	`zoomdata-edc-<connector>`	varies	✓	Only the ports for the Composer connectors to the data stores you will use are required. For a complete list, see Data Connector Reference.
Data Writer	`zoomdata-data-writer`	8081	✗	Required only if you use flat file or Upload API sources.
Screenshot Service	`zoomdata-screenshot-service`	8083	✗	Required only if you schedule dashboard reports.
PostgreSQL metadata repository	`zoomdata-postgres`	5432	✓	

Time Synchronization

Composer recommends installing the Network Time Protocol daemon (NTPD) to allow for time synchronization of networked servers to Coordinated Universal Time (UTC), if not already available in your network. See Using the Network Time Protocol to Synchronize Time for guidance.

Clean Installation and Upgrade Differences

An upgrade of Composer means that you are upgrading from an earlier version of Composer to a newer version of Composer. This includes uninstalling the previous version and installing a later version. To learn more about upgrades, see Upgrade Composer.

A clean installation of Composer means that you are installing a new version of Composer from scratch. The Composer metadata is empty and contains no information. This either means that the PostgreSQL is a fresh install as well or that the **zoomdata** database is empty.

A supported upgrade of Composer allows users to continue working in an existing Composer environment while a clean installation of Composer is completely new. For clean installation instructions, see Install Composer.

The main difference between an upgrade and clean installation of Composer is whether or not the PostgreSQL is from an existing Composer installation or is new and empty.

Install Composer

This section provides instructions for performing a clean installation of Composer in your operating environment and is applicable to for both RPM (CentOS, REHL) and Ubuntu environments.

For information about the difference between a clean installation of Composer and an upgrade to the latest GA release, see Clean Installation And Upgrade Differences.

Upgrade and Migration Considerations

- Windows Server 2012R2 is not compatible with both Java17 binaries and the latest releases of Composer (23.2 and later). We recommend you use Windows 2016 or later to run Composer 23.2 and later.
- In general, you can upgrade directly to the latest version of Composer from a prior version.
- If you are upgrading to a newer version of Composer and you also want to change your encryption mode, perform the upgrade first and then complete the steps described in Change The Encryption Mode.

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

Important: If you are upgrading to a newer version of Composer and have created an attribute named `User.timeZone`, this may be overwritten on upgrade. See Upgrade Workflow for more information about preparing your environment for the upgrade process.

In general, the installation process is automated and you only need to run an installation script. The installation script is implemented using Bootstrap. This script accesses a dedicated Composer repository and automatically downloads all the necessary components to install your Composer microservice.

You can install Composer without using the automated installation script. This lets you install and enable each Composer microservices manually in your target server. If you need an alternative option, read Alternative Installation Options.

Note: To install Composer in other environments, see Install Composer In A Windows Environment. To install an orchestrated Composer solution, see Running Composer In Kubernetes.

After the installation has completed, you need to activate the Composer microservices, download and configure a JDBC driver if you are using specific data sources (see Post-Installation Options for a list), and open a browser window and enter the specific IP address to access the Composer client.

Review and complete (as appropriate for your installation) the following installation information:

Running Composer in Kubernetes

Use Kubernetes to mange your Composer configuration management tasks. This scalable solution allows you deploy your Composer environment securely and scale all microservices as needed.

Important: Kubernetes is supported for new installations only for Composer version 23.1 and later.

See the following topics:

Install Composer in a Windows Environment

This section provides instructions for performing a clean installation of Composer in your Windows Server environment.

For information about the difference between a clean installation of Composer and an upgrade to the latest GA release, see Clean Installation And Upgrade Differences.

Upgrade and Migration Considerations

- Windows Server 2012R2 is not compatible with both Java17 binaries and the latest releases of Composer (23.2 and later). We recommend you use Windows 2016 or later to run Composer 23.2 and later.
- In general, you can upgrade directly to the latest version of Composer from a prior version.
- If you are upgrading to a newer version of Composer and you also want to change your encryption mode, perform the upgrade first and then complete the steps described in Change The Encryption Mode.

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

Important: If you are upgrading to a newer version of Composer and have created an attribute named `User.timeZone`, this may be overwritten on upgrade. See Upgrade Workflow for more information about preparing your environment for the upgrade process.

In general, the installation process is automated and you only need to run an installation script. The installation script is implemented using Bootstrap. This script accesses a dedicated Composer repository and automatically downloads all the necessary components to install your Composer microservice.

By default, the bootstrap script installs these core components, connectors, and required dependencies:

- Zoomdata web application
- Query Engine
- MSSQL
- Mongo DB
- Elastic 7

- Solr
- Cloudera Search
- Consul
- PostgreSQL 12
- Corretto JDK17
- Chocolatey

To install other components, adjust bootstrap switches as needed. See Windows Bootstrap Reference, ComposerSymphony Data Discovery Microservice Name Reference and Data Connector Reference.

After the installation has completed, you need to activate the Composer microservices, download and configure a JDBC driver if you are using specific data sources (see Post-Installation Options for a list), and open a browser window and enter the specific IP address to access the Composer client.

Review and complete (as appropriate for your installation) the following installation information:

- Installation Prerequisites
- Installation Steps
- Post-Installation Options
- Access ComposerSymphony

Windows Bootstrap Reference

The Windows bootstrap installation script installs, by default, these components for Composer:

Composer core components:

- Zoomdata web application
- Query Engine
- Consul

Mandatory dependencies:

- Corretto JDK17
- PostgreSQL 12 or 16

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

Other components:

- MSSQL
- Mongo DB
- Elastic 7
- Solr
- Cloudera Search
- Chocolatey

Add or remove other components or adjust optional components using bootstrap switches. For further help with the bootstrap script, run `Get-Help ./bootstrap-composer.ps1` for syntax help or `Get-Help ./bootstrap-composer.ps1 -Full` for extended parameter help.

Important: When you upgrade Composer, all services are stopped before the upgrade is installed, and can't be accessed by users until the upgrade is complete and services restarted.

Installation Parameters

Parameter	Description	Notes
`-InstallDestinationPath`	The path to install Composer files.	
`-ComposerVersion`	The version of Composer to install.	`latest` Install the most recent version of Composer available. `latest-lts` Install the most recent Long Term Support version of Composer available. `X.x` Install the most recent version of Composer available for that trunk line (for example, `23.4`. `X.x.x` Install the exact patch version of Composer specified (for example, `23.4.1`.
`-ComposerRepository`	The repository to connect to for installation manifests and binary packages for Composer.	Official Composer repository: `<https://composer-repo.logianalytics.com`.
`-ConnectorsList`	A comma separated list of connectors to install.	By default, MSSQL, PostgreSQL, MongoDB, Elasticsearch 7.0, Apache Solr,and Cloudera Search are installed if you make no changes to the initial script. If you include connectors with this parameter, only those connectors are installed. Include the default connectors to install those as well. If you run the script with new connectors added using this parameter, the new connectors included are added to the existing installed connectors. See Data Connector Reference.
`-AdditionalComponentsList`	A comma separated list of additional services to install.	Include one or more of the following services. By default, all are included if you make no changes to the initial script.

Parameter	Description	Notes
		■ data-writer-mssql ■ data-writer-postgresql ■ screenshot-service ■ zoomdata-admin-server ■ zoomdata-config-server If you install the screenshot-service, Google Chrome and the Chrome driver are installed as well.
`-PostgreSQLHost`	The hostname or IP adddress of your external PostgreSQL server (configured for Composer metadata storage).	By default, `localhost`, which installs a new instance of PostgreSQL. ■ PostgreSQL 12 for version 24.2 and earlier ■ PostgreSQL 16 for version 24.3 and later
`-PostgreSQLTCP`	The TCP port of your external PostgreSQL server (configured for Composer metadata storage).	
`-PostgreSQLUser`	The PostgreSQL user name configured as owner of your Composer database.	
`-PostgreSQLPass`	The PostgreSQL password for the user configured as owner of your Composer database.	
`-ServicesAction`	Manage Windows services for Composer. You can perform this without performing other bootstrap activities.	Use on one or more services as needed. Actions include: ■ install ■ start

Parameter	Description	Notes
		■ stop ■ restart ■ status ■ uninstall
`-SkipDependenciesInstall`	Set to skip installing Composer dependencies.	■ Chocolatey ■ PostgreSQL 12 or PostgreSQL 16 ■ Google Chrome ■ Google Chrome Driver
`-SkipComposerInstall`	Set to skip installing Composer services.	Download, extract, and configure actions are the only actions performed.
`-SkipComposerConfigure`	Set to skip configuring Composer services.	Download, extract, and install actions are the only actions performed.
`-SkipComposerStart`	Set to perform all installation actions, but do not start Composer services.	
`-SkipMetadataDumpOnUpgrade`	Set to disable automatic metadata dumping during upgrade or installation of Composer services.	
`-DumpComposerMetadata`	This is a separate action to perform a metadata dump. You must stop all services first to ensure consistency.	After stopping all services, you can perform this without performing other bootstrap activities.
`-DeinstallComposer`	Completely remove all Composer components and dependencies from this instance.	
`-PreDownloadedPackagesPath`	Set to install Composer binaries from the defined path instead of downloading from the Composer repository.	The folder you point to must include all core components, required connectors, and additional services.
`-BootstrapLogFile`	Set to define an alternative path to record the	By default, the log file is put in the same folder you

Parameter	Description	Notes
	bootstrap logs.	use to launch the bootstrap script.

Note: Manual install of Composer is not currently supported for Windows environments.

See Post-Installation Options for more information and links to instructions.

Installation Examples

Install the most recent rapid release version of Composer and PostgreSQL as a dependency. Includes Impala and MSSQL EDC connectors and the MSSQL-based data writer service to `c:\logi-composer`. All components are started after installation.

```
./bootstrap-composer.ps1 -ComposerVersion latest -ConnectorsList impala,mssql -AdditionalComponentsList data-writer-mssql
```

Install only core components (Consul, Composer Server, Query Engine) of the most recent released Composer version in the 7.10 trunk. Default EDC connectors are installed (MSSQL, PostgreSQL, Mongo DB, Elastic 7, Solr, Cloudera Search).

```
./bootstrap-composer.ps1 -ComposerVersion 23.4
```

Install the most recent published long term support version of Composer. Do not start services.

```
./bootstrap-composer.ps1 -ComposerVersion latest-lts -SkipComposerStart
```

Installation Prerequisites

The installation script works in the following environments:

- CentOS 9
- Ubuntu 18, 20, and Ubuntu 22.
- RHEL 7 and 8
- Windows Server versions 2012 R2, 2016, and 2019

Note: CentOS Stream 9 is supported for new instances of Composer 24.3 and higher. Contact Support for upgrade assistance from earlier versions.

Upgrade and Migration Considerations

- Windows Server 2012R2 is not compatible with both Java17 binaries and the latest releases of Composer (23.2 and later). We recommend you use Windows 2016 or later to run Composer 23.2 and later.
- In general, you can upgrade directly to the latest version of Composer from a prior version.
- If you are upgrading to a newer version of Composer and you also want to change your encryption mode, perform the upgrade first and then complete the steps described in Change The Encryption Mode.

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

Important: If you are upgrading to a newer version of Composer and have created an attribute named `User.timeZone`, this may be overwritten on upgrade. See Upgrade Workflow for more information about preparing your environment for the upgrade process.

Upgrading with Kubernetes

Important: Composer does not support upgrading to use Kubernetes in this release.

Java Considerations

Java 11.0.5 is required to run Composer 23.1 and Java 17 is required to run Composer 23.2.

An option to install OpenJDK is included in the installation and upgrade scripts provided by Composer. If you skip this option or if you install or upgrade the product manually, make sure that Java 11.0.5 is installed for Composer 23.1 and Java 17 for Composer 23.2. If you do not, Composer will not start after the installation.

Target Server Prerequisites

The target server for the Composer software should meet the following prerequisites:

- The server must be connected to the Internet.
- If this is a new (fresh) installation, the server must not have PostgreSQL already installed. (Not required for upgrades.)
- If this is a new (fresh) installation, the server must not contain any `zoomdata` folders or property files from previous versions. If a previous version of Zoomdata or Composer was installed on this server, ensure that all property files have been deleted before running the installer script.(Not required for upgrades.)
- The user installing Composer must be able to use the `sudo` command on the server on Linux platforms or `Administrator` privileges for the server on Windows platforms.

If you do not have an internet connection on the server on which Composer is being installed, download the Composer installation package and load it onto the target server. After this is done, you can manually install Composer. See Install Composer Manually.

If the server on which Composer is to be installed does not meet all the prerequisites, see Alternative Installation Options.

In addition, Composer benefits from having time synchronization in your network. Specifically, Composer leverages the Network Time Protocol daemon (NTPD), which performs time synchronization of networked servers to Coordinated Universal Time (UTC). If needed, read Use The Network Time Protocol To Synchronize Time for instructions to set this up.

After you have made any needed adjustments to your network configurations, you can continue the installation process.

Server Size Guidelines

To maximize the operational efficiency of Composer., you need to take into account several factors to appropriately size the Composer. server in your operating environment:

- Concurrent user count. This is the most important parameter that increases the load on the system. Concurrent users represent only a portion of the total number of available users in an organization who may access the Composer. server at the same time. A single-node deployment can handle up to 100 concurrent users with acceptable performance. If you need more than 100 users, you should consider using several Composer. nodes.

- Entity (users, data sources, visuals, dashboard, etc) count. The number of entities in the system doesn't impact the performance of the system if you have up to 1000 instances of each entity type. If you want to support more than 1000 instances of each entity type you must allocate more resources than shown in the table below.

- Data cardinality. The amount of data processed doesn't affect the performance of Composer., because Composer. pushes the work for the queries down to your data stores and thus will rely on the performance of your data stores. However, if you have large data set and want to group the data by a high-cardinality field, you should increase the resource allocation for the query engine.

- High availability (HA). If your Composer. environment is an HA environment, sizing requirements may double if each Composer. microservice is running on multiple instances.

See also System Requirements for information about Composer system requirements.

The table below reviews the recommended sizing guidelines for a single-node Composer. server. Keep in mind that the specifications provided are suggested starting points and reflect only **the minimum estimates**. Sizing for performance is impacted by many other factors outside of the ones mentioned here, such as where Composer. resides, the types of data sources you are using, interactions between applications and data sources, and a variety of other factors.

The following sizing guidelines reflect testing of a single-node Composer. server with up to 1000 concurrent users, 400 data sources, 500 dashboards, and 6000 visuals.

Concurrent Users	Hardware Requirements	Time Opening Dashboard (12 Visuals)	Composer Web	Query Engine	Connector	PostgreSQL Metadata Store
Up to 5	Memory: 8 GB CPU: 8 core Disk: 100 GB	17 seconds	Memory: 2 GB	Memory: 1 GB	Memory: 512 MB	Memory: 512 MB

Concurrent Users	Hardware Requirements	Time Opening Dashboard (12 Visuals)	Composer Web	Query Engine	Connector	PostgreSQL Metadata Store
Up to 20	Memory: 8 GB CPU: 8 core Disk: 100 GB	19 seconds	Memory: 2 GB	Memory: 1 GB	Memory: 512 MB	Memory: 512 MB
Up to 100	Memory: 16 GB CPU: 16 core Disk: 100 GB	26 seconds	Memory: 4 GB Configuration Settings: `spring.datasource.hikari.maximum-pool-size=50`	Memory: 3 GB	Memory: 1 GB	Memory: 1 GB

Up to 1000	Memory: 16 GB
	CPU: 16 core
	Disk: 100 GB
		4 minutes	Memory: 5 GB
			Configuration Settings:
			`spring.datasource.hikari.maximum-pool-size=150`
			`server.jetty.max-threads=1000`
			`query.engine.service.maxConcurrentRequests=1000`
				Memory: 4 GB
				Configuration Settings:
				`server.jetty.max-threads=1000`
					Memory: 1 GB	Memory: 1 GB

Use the Network Time Protocol to Synchronize Time

The Network Time Protocol daemon (NTPD) is a service that performs time synchronization of networked servers to Coordinated Universal Time (UTC). Using NTP helps mitigate the effects of network latency by synchronizing your network with accurate time servers. In addition, certain Composer functionalities benefit from having NTP in your network, including:

- The connection between Composer server and the data sources (so that monitoring of data source performance and possible network latency issues can be done).
- Authentication protocols (for example, Kerberos), which require precise time correspondence on all instances to work properly.
- Scaled out deployments so that all nodes can have synchronized time.
- Single Sign-On (via SAML), to avoid potential failure by the identity provider to authenticate SAML users.

Ideally, NTP should be installed prior to installing the Composer server. The steps below help you install the NTP service in your network. However, be sure to work with your network administrator to use the most appropriate time protocol service for your network environment.

Install NTP on RPM-Based Distributions

To install NTP on CentOS or RHEL, perform the following steps:

1. Run the following command:

```
sudo yum install -y ntp
```

2. Check that the service is up and running:

```
sudo service ntpd status
```

Install NTP on Ubuntu

To install NTP on Ubuntu, perform the following steps:

1. Run the following commands:

```
sudo apt-get update
sudo apt-get install -y ntp
```

2. Check that the service is up and running:

```
sudo service ntp status
```

Post-Installation Steps

If you install the NTP service after Composer has already been installed in your network, you should restart Composer service after NTP has been successfully installed:

```
sudo service zoomdata restart
```

Related Topics:

- Install Composer
- Install Composer Manually

Configure the Maximum Number of Open Processes and Files

Configuring the maximum number of open processes and open files that can run in your operating environment keeps Composer processes from hitting or exceeding the resource limits that may be imposed by your operating system.

Instructions are provided here for the following operating environments:

CentOS Instructions

If you are installing via RPM using CentOS or Red Hat, the recommended settings differ slightly. Because `systemd` is responsible for starting the various Composer microservices, you need to amend the files and microservices that are launched so the `limits.conf` file is not ignored.

Composer recommends that you create an override directory specifically for the microservices you want to override.

1. Create a new `systemd` directory:

   ```
   mkdir /etc/systemd/system/<servicename>.service.d/
   ```

   Replace <servicename> with the microservice you need to override.

2. The file name for the microservice you want to override needs to be in `.conf`, so use the following command:

   ```
   touch /etc/systemd/system/<servicename>.service.d/<servicename>.conf
   ```

For more information about managing microservices for Red Hat, see the Red Hat documentation.

If Composer stops operating and you receive the error message `too many files, cannot operate`, you may need to increase the system limits in your CentOS or Ubuntu 18, 20, or 22 environments. Make the following changes:

1. In the `service.d` folder for each microservice, open the `limits.conf` file. If the configuration file does not exist, this command creates it.

   ```
   vi /etc/systemd/system/<servicename>.service.d/limits.conf
   ```

2. Add the following to the file:

```
[Service]
LimitNOFILE=10000
```

3. Restart the init service:

```
systemctl daemon-reload
```

4. Restart the `zoomdata` microservice:

```
systemctl restart zoomdata.service
```

5. Verify your changes were applied:

```
ps -ef | grep zoomdata-web | grep -v grep | awk '{system("cat /proc/"$2"/limits")}' | grep -i "Max open files"
```

6. In the output, you should see the following, indicating that your changes were applied:

```
Max open files            10000                10000                files
```

Return to the installation topic you are using to continue the installation process:

- Install Composer
- Install Composer Manually

Manually Install OpenJDK

Composer v23.2 and later runs on Java 17. When you perform a straightforward upgrade to this release, Java is updated automatically.

Composer v23.1 and earlier continue to run on Java 11.

Manually install OpenJDK

1. Stop all Composer components (if any are running) before you install OpenJDK. See Stop ComposerSymphony Microservices.
2. Run the following command on the installation machine.

   For JDK 17:

   ```
   sudo mkdir -p /opt/zoomdata/jre && curl -sL https://corretto.aws/downloads/latest/amazon-corretto-17-x64-linux-jdk.tar.gz | sudo tar zx -C /opt/zoomdata/jre --strip-components=1
   ```

   For JDK 11:

   ```
   sudo mkdir -p /opt/zoomdata/jre && curl -sL https://corretto.aws/downloads/latest/amazon-corretto-11-x64-linux-jdk.tar.gz | sudo tar zx -C /opt/zoomdata/jre --strip-components=1
   ```

3. Restart all Composer components. See Start ComposerSymphony Microservices.

Note: In Windows environments, Composer installs a compatible JDK as part of the bootstrap installation. To install a system wide Java distribution, see https://www.java.com/en/download/help/windows_manual_download.html.

For information on how Composer validates the version of Java used in an environment, see How Composer Validates An Environment's Java Version.

How Composer Validates an Environment's Java Version

Composer determines which Java is being used in your environment while the microservices are starting. It evaluates your environment in the following sequence. The first Java executable in this sequence that meets minimum requirements is used and validation stops.

1. The directory in the $JAVA_HOME environment variable path is evaluated to verify that it points to JRE/JDK and its `$JAVA_HOME/bin/java` path contains a valid Java executable that meets the minimum requirements.

2. The `/etc/environment` file is parsed for its JAVA_HOME variable identifying the Java installation directory. This Java installation directory is then evaluated for a valid Java executable that meets the minimum requirements.

3. The `<install-path>/jre` folder is evaluated for a valid Java executable that meets the minimum requirements.

4. The directories listed in the $PATH environment variable are evaluated for a valid Java executable that meets the minimum requirements.

If a Java executable that meets the minimum requirements is not found in this process, an exception occurs and Composer is not started. To install a valid version of OpenJDK, see Manually Install OpenJDK.

Installation Steps

To begin the installation process, you must receive the installation instructions from Composer Technical Support. This email contains the installation script that you will run on the server where the Composer environment will reside. If you have not received installation instructions, open a ticket with Composer Technical Support.

Note: To install Composer in other environments, see Install Composer In A Windows Environment. To install an orchestrated Composer solution, see Running Composer In Kubernetes.

Steps

After you have received installation instructions from Technical Support, complete the following steps.

- Step 1: Run The Installation Script
- Step 2: Configure The Firewall
- Step 3: Identify The IP Address
- Step 4: Access Composer From A Web Browser

After you have installed Composer, review the post-installation options in Post-Installation Options. For information about accessing Composer after it is installed, see Access ComposerSymphony.

Step 1: Run the Installation Script

The email or PDF you receive from Composer Technical Support describes the command used to obtain the Composer installation bootstrap procedure and the command used to run the bootstrap procedure. Run these commands, in order, on the target server for Composer to start the automated installation process.

Linux environments: The following Composer components are downloaded to your target server:

- Database for metadata store (using PostgreSQ)
- The Composer Server
- Query Engine

- Data Writer microservice
- Connector microservices

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

- Composer v24.2 and earlier: PostgreSQL 12
- Composer v24.3 and later: PostgreSQL 16

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

Windows Environments

By default, the bootstrap script installs these core components, connectors, and required dependencies:

- Zoomdata web application
- Query Engine
- MSSQL
- Mongo DB
- Elastic 7
- Solr
- Cloudera Search
- Consul
- PostgreSQL

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

- Corretto JDK17
- Chocolatey

To install other components, adjust bootstrap switches as needed. See Windows Bootstrap Reference, ComposerSymphony Data Discovery Microservice Name Reference and Data Connector Reference.

When the installation script has completed, complete the remaining steps in this section.

Step 2: Configure the Firewall

See Configure The Firewall.

Step 3: Identify the IP Address

See Identify The Composer IP Address.

Step 4: Access Composer from a Web Browser

After the installation script has completed, it will take a few minutes for Composer to complete its setup of the metadata store. Please wait a few minutes before accessing Composer from your web browser. When you are ready to access Composer, read Access ComposerSymphony.

> **Note:** We recommend that you log into Composer for the first time using admin credentials. This allows you to review all the account-level features available. As an admin, you can access functions that let you connect your data sources to Composer. You can also create and activate user accounts, including an admin user account that will allow you to access the admin functions. See Supplied Users And User Groups.

If you receive a message indicating that Composer is not yet accessible, the Composer setup may not yet be complete. Wait a few more minutes before trying again or opening a Support ticket. If you continue to have issues accessing Composer from your browser, open a ticket with Composer Technical Support.

Configure the Firewall

After you have successfully installed the Composer components onto your server, you need to configure the firewall. Configure "iptables" to accept port 8443 and to forward incoming HTTPS requests on port 443 to the Composer server port 8443. Note that the command lines may differ slightly depending on the Linux environment. Select the appropriate Linux environment below.

- CentOS Commands
- Ubuntu 18, 20, Or 22 Commands
- CentOS Commands

These commands set up the firewall rules to the default eth0 network interface. If you want to apply them to another network interface, replace it in the commands below. If you want to apply the rules to all the interfaces, remove '-i eth0' from the command line.

CentOS Commands

```
sudo yum install iptables-services
sudo systemctl enable iptables
sudo systemctl start iptables
sudo iptables -I INPUT 1 -i eth0 -p tcp --dport 8443 -j ACCEPT
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j DNAT --to-destination :8443
sudo iptables -I INPUT 1 -i eth0 -p tcp --dport 80 -j ACCEPT
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT --to-destination :8080
sudo /usr/libexec/iptables/iptables.init save
```

Ubuntu 18, 20, or 22 Commands

```
sudo apt-get install iptables-persistent
sudo iptables -I INPUT 1 -i eth0 -p tcp --dport 8443 -j ACCEPT
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j DNAT --to-destination :8443
sudo iptables -I INPUT 1 -i eth0 -p tcp --dport 80 -j ACCEPT
sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT --to-destination :8080
sudo netfilter-persistent save
sudo netfilter-persistent reload
```

When prompted for input for the question of `iptables-persistent`, enter `yes`.

Windows Defender Firewall

See this best practices article: Best practices for configuring Windows Defender - Windows Security.

Continue Install

Return to the installation topic your are using to continue the installation process:

- Install Composer
- Install Composer Manually
- Upgrade Composer

Identify the Composer IP Address

After Composer is installed and the firewall has been configured, you need to identify the Composer IP address so you can access Composer on your web browser. Record the IP address of the host where Composer is installed.

To obtain the Composer IP address, enter the following command in a terminal window on the Composer server:

```
[zoomdata@localhost ~]$ hostname -I
```

Make note of this IP address. You will need it when you access Composer from a web browser.

Add a JDBC Driver

For certain data sources, the JDBC drivers needed are no longer included in the installation package of Composer. You need to provide your own JDBC driver for the following data sources:

- Amazon Redshift
- Dremio
- Jira
- MemSQL
- MySQL
- Oracle
- SAP Hana
- SAP S/4HANA
- SAP IQ
- Salesforce
- Snowflake
- Teradata
- Vertica

The following Composer connectors are distributed with a JDBC driver, but you can download and install newer versions using the information in this topic: Snowflake.

This approach allows you the flexibility to add a specific JDBC driver that meets your licensing, support policies or operational needs. As a result, in order to connect to and visualize data from Composer, you first need to download and install a JDBC driver.

Caveats

If the JDBC driver for the Composer connector is not configured, the connector server will not start and the connector cannot be enabled within Composer. See Manage Connectors And Connector Servers.

Install a JDBC Driver

To use any of the connectors listed above, perform the following steps to install the required JDBC driver after successful installation of Composer microservices:

1. Download the required driver from the vendor's site to the corresponding Composer instance. Place the required driver in the following folder:

i. Linux platforms: `/opt/zoomdata/lib/edc-<connector_name>/`.

ii. Windows platforms: `<install_path>\lib\edc-<connector_name>/`. For example, place MySQL libraries for the default Composer path install in this folder: `c:\logi-composer\lib\edc-mysql\` if Composer is installed in `c:\logi-composer\`.

Important: The truststore path passed as part of the JDBC url must only contain forward (/) slashes in Windows environments.

If the folder does not exist, you need to create it in the location mentioned above. See the following table for resources for vendor's JDBC drivers. Make sure that the Composer administrator has read-level access rights to the JDBC driver (JAR) file.

Connector	Link to JDBC Driver	License Type	Supported Version
Amazon Redshift	http://docs.aws.amazon.com/redshift/latest/mgmt/configure-jdbc-connection.html#download-jdbc-driver	Commercial	1.2.16.1027
Dremio	https://www.dremio.com/drivers/	LGPL	4.1
MemSQL	https://dev.mysql.com/downloads/connector/j/	LGPL	8.0.13
MySQL	https://dev.mysql.com/downloads/connector/j/	LGPL	8.0.13
Oracle	https://www.oracle.com/database/technologies/appdev/jdbc-ucp-183-downloads.html	Commercial	18.3.0.0
SAP Hana	https://developers.sap.com/trials-downloads.html	Commercial	2.0
SAP IQ	https://www.sap.com/index.html	Commercial	16
Snowflake	https://repo1.maven.org/maven2/net/snowflake/snowflake-jdbc/	LGPL	
Teradata	https://downloads.teradata.com/download/connectivity/jdbc-driver	Commercial	15.00.00.30
Vertica	https://www.vertica.com/client-drivers/	Commercial	All

2. Use the following command to access and open the property file:

i. Linux platforms: `vi /etc/zoomdata/edc-<connector_name>.properties`. If you are not logged in as a root user, enter `sudo vi /etc/zoomdata/edc-<connector_name>.properties` to create the desired file. If the properties file does not exist, this command creates it.

ii. Windows platforms, using a text editor that can edit Windows property files: `<install-path>\conf-modified\edc-<connector_name>.properties`.

Replace `<connector_name>` with the name of the connector you are configuring:

Connector	Connector Property File Name
Amazon Redshift	edc-redshift.properties
MemSQL	edc-memsql.properties
Microsoft SQL Server	edc-mssql.properties
MySQL	edc-mysql.properties
Oracle	edc-oracle.properties
SAP Hana	edc-saphana.properties
SAP S/4HANA	
SAP IQ	edc-sapiq.properties
Snowflake	edc-snowflake.properties
Teradata	edc-teradata.properties
Vertica	edc-vertica.properties

3. In the edc-<connector_name>.properties file, add the following property:

```
datasource.driver-config.jar-path=<JDBC_driver_filepath>
```

If you need to add multiple paths, use a comma-separated list:

```
datasource.driver-config.jar-path=<JDBC_driver_filepath1>,<JDBC_driver_filepath2>
```

4. For MemSQL and MySQL connectors, add the following property to the property files:

```
datasource.driver-config.class-name=com.mysql.cj.jdbc.Driver
```

5. Save your changes to the properties file.

6. Restart the corresponding connector by running the appropriate command:

i. For CentOS and Ubuntu 18, 20, or 22: `systemctl restart zoomdata-edc-<connector_name>`

ii. For Windows: `PS C:\> Restart-Service -Name zoomdata-edc-<connector_name>`

7. Log in as the supervisor, access the Connectors page, and verify that the connector is enabled so it appears in the data source list. After the JDBC driver has been configured and the connector has been enabled, users with the correct access privileges can use the connector to connect to the data store in a data source configuration.

Alternative Installation Options

The following alternative installation options are available. Select an option for step-by-step instructions to set up your Composer instance:

- Install Composer Manually (instead of using the installation script). This entails (1) downloading and placing the Composer installation package in a dedicated directory on your target server, (2) installing the Composer components and (3) registering and activating the Composer components.
- Install Composer from a tarball package. Contact Technical Support for assistance. You will receive a tarball package that you unzip in a dedicated directory. You can then install each component manually.

Note: To install Composer in other environments, see Install Composer In A Windows Environment. To install an orchestrated Composer solution, see Running Composer In Kubernetes.

Install Composer Manually

If running the installer script is not a viable option for you, you can install Composer manually.

Note: Manual install of Composer is not currently supported for Windows environments.

To install Composer manually, complete the following steps:

Step 1. Review the Prerequisites

Refer to System Requirements and Server Size Guidelines for information on the recommended settings for deploying your software on-premises.

The target server for your software should meet the following conditions:

- The server does not have PostgreSQL already installed
- The server does not contain any Composer property files, meaning if a previous version of Composer was installed in this server, ensure that all property files have been deleted.
- The user installing Composer is able to use the `sudo` command in the server

CentOS Requirements

Note: CentOS Stream 9 is supported for new instances of Composer 24.3 and higher. Contact Support for upgrade assistance from earlier versions.

Time Synchronization Requirements

Composer benefits from time synchronization in your network. Specifically, it leverages Network Time Protocol (NTP), which performs time synchronization of networked servers to Coordinated Universal Time (UTC). If needed, read Use The Network Time Protocol To Synchronize Time for instructions on setting this up.

Java Requirements

You must have Java 11.0.5 installed for Composer 23.1 and Java 17 installed for Composer 23.2 and later to use Composer. Without it, your software will not start.

After you have made any needed adjustments to your network configurations, return to this topic to continue the installation process. See also How Composer Validates An Environment's Java Version.

Step 2. Set Up Composer's Metadata Store

Composer uses a standard PostgreSQL database instance to store its metadata. Composer strongly recommends using this instance as it is configured with the appropriate settings.

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

To use your own or alternative PostgreSQL instance, contact Composer Technical Support for further guidance.

Read Install And Set Up Composer's Metadata Store. Complete the setup instructions and then return to this topic to continue the manual installation of Composer on your server.

Step 3. Configure a Dedicated Directory

Before you can download the Composer installation packages onto your server, you need to create the Composer directory where the installation and property files are stored. After the directory is created, you need to create property files in that directory.

Create the Composer Directory

Create the following directory to store all Composer-related files:

```
sudo install -o root -g root -m 0755 -d /etc/zoomdata
```

Create the Default Composer Properties File

Create the default Composer properties file that contains the available variables and parameters related to Composer operation:

```
sudo touch /etc/zoomdata/zoomdata.properties
sudo chmod 0644 /etc/zoomdata/zoomdata.properties
sudo vi /etc/zoomdata/zoomdata.properties
```

Create the Query Engine Properties File

Create the Composer query engine properties file that contains the available variables and parameters related to query engine operation:

```
sudo touch /etc/zoomdata/query-engine.properties
sudo chmod 0644 /etc/zoomdata/query-engine.properties
sudo vi /etc/zoomdata/query-engine.properties
```

Add the Default Metadata Parameters to the Appropriate Composer Properties File

1. Add the following metadata store-related parameters in the newly-created `zoomdata.properties` file. Essentially, you are storing the username and password details for the metadata store in this property file.

```
spring.datasource.url=jdbc:postgresql://<ip of host>:<port>/zoomdata
spring.datasource.username=<db_username>
spring.datasource.password=<db_password>
keyset.destination.params.jdbc_url=jdbc:postgresql://<ip of host>:<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 of host>:<port>/zoomdata-upload
upload.destination.params.user_name=<db_username>
upload.destination.params.password=<db_password>
upload.destination.schema=public
upload.batch-size=1000
```

2. Add the following `zoomdata-qe` database metadata store-related parameters in the newly-created `query-engine.properties` file.

```
spring.qe.datasource.jdbcUrl=jdbc:postgresql://<ip of host>:<port>/zoomdata-qe
spring.qe.datasource.username=<db_username>
spring.qe.datasource.password=<db_password>
```

In each case, remember to save the files before exiting the editor.

Step 4. Download Dependencies and the Composer Installation Packages

Composer requires the following external dependencies for a successful installation:

- EPEL (for CentOS environments)
- Socat
- OpenSSL

If you have not already received the Composer installation package, contact Composer Technical Support to request it. In the request, be sure to include the Linux operating system version you are using (for example, CentOS). You can select the Support button on this page to create your request.

Note: If your server does not have internet access, you will need an internet-enabled computer to download the packages and then move them to the intended server.

Step 5. Obtain Download Instructions and Installation Packages

Contact your Composer technical support representative and obtain download instructions for the Composer installation packages. Follow the instructions and download the installation packages, remembering to place them in the Composer directory on the target server. The following Composer components are included

in your installation packages:

- The Composer server
- Connector microservices
- Query Engine

Step 6. Install the Composer Server

Use the following command to install Composer in a CentOS environment:

```
sudo yum install zoomdata* -y
```

Use the following command to install Composer in an Ubuntu environment:

```
sudo dpkg -i zoomdata*
```

Step 7: Set Composer Microservices to Start Whenever the Server Boots

Microservices need to be set to automatically start whenever the server is started or rebooted.

In a CentOS or an Ubuntu environment, run the following command:

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

Optionally, you can manually set up each microservice by running the following commands in CentOS and Ubuntu:

```
sudo systemctl enable zoomdata-edc-<connector-name>
sudo systemctl enable zoomdata-screenshot-service
sudo systemctl enable zoomdata-consul
sudo systemctl enable zoomdata
sudo systemctl enable zoomdata-query-engine
```

Step 8: Start the Microservices

The Composer microservices must be enabled.

In a CentOS or an Ubuntu environment, run the following command:

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

Optionally, you can manually enable each microservice by running the following commands in CentOS and Ubuntu 18, 20, or 22:

```
sudo systemctl start zoomdata-edc-<connector-name>
sudo systemctl start zoomdata-edc-rts
sudo systemctl start zoomdata-screenshot-service
sudo systemctl start zoomdata-query-engine
sudo systemctl start zoomdata
```

Step 9. Configure the Firewall

Read Configure The Firewall for complete information.

Step 10. Identify the Composer IP Address

Read Identify The Composer IP Address for complete information.

Step 11. Access Composer

Read Access ComposerSymphony for more information.

Step 12. Complete Post-Installation Steps

Complete any post-installation actions needed for your environment. These include, but are not limited to:

1. Enabling the Real-Time Sales demo data source
2. Setting up the Screenshot microservice.

See Post-Installation Options for more information and links to instructions.

Set Up and Use the Data Gateway Service

Adding a data connector gateway to your Composer environment allows you to connect securely to data outside of your environment. Use a gateway client to authenticate your connection and make the data available to users.

To establish communication between your data and Composer, enable the data gateway service, SSL environment, then generate gateway clientes to retrieve your data from your external data bases, either on-premise or in the cloud.

After you've set up the data connector, users with appropriate privileges can access and use the data in source, visuals, and dashboards.

Enable the Data Gateway Service

First, enable the data gateway service in your environment. You must be the default Admin user or a member of the Admin or Supervisors group.

Important: If you have not enabled SSL, do so before completing these steps. See Enable Secure Sockets Layer.

- Composer installed in a Linux or Windows environment:
  - Enable the data gateway in the zoomdata web service by including `data-gateway.client-api.enabled=true` in the properties file.
  - Add properties, if needed, to control resource consumption in `dataGatewayService` in the property file.
  - Start the service.
  - Composer listens for the service on port 80.
- Composer installed via Kubernetes:
  - Enable `dataGatewayService` (set to `true`) in the values.yaml for Composer. See Enable the Data Gateway.
  - Two services are added and exposed through the default ingress once you enable them:
    - `data-gateway-service-external` - looks externally to accept websocket connections
    - `data-gateway-service-internal` - looks internally to represent connecters inside the cluster
  - Start the service.

Once enabled, use the API to create data gateway clients, one for each connection you want to establish, using the `/api/data-gateway/clients` endpoint. Provide a name and description as needed. ComposerSymphony returns the client `id` and client `secret`, used to authenticate to ComposerSymphony.

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

Use the Data Gateway Service

After enabling the data gateway service and creating one or more gateway clients, use credentials and the client id and client secret for each database to establish the connections from the database to ComposerSymphony.

Manage Connectors

At some point, the connector shows up on the manage connector servers page. After that you can make a connection.

Important: Once established, data gateway connectors are available in all tenants to users with appropriate privileges who can create connectors. Limit access in a multi-tenant environment as needed.

See Establish a Data Gateway Connection.

Enable Secure Sockets Layer

If you have not enabled SSL, enable it now.

Set SSL to `true` through your preferred software, such as Spring Boot. Pass the key store and the key store password. Once complete, you can connect to the gateway service through a secured Websocket URL connection. Specify this setting as a parameter in the data gateway agent.

Establish a Data Gateway Connection

Once you or your system administrator has enabled the data gateway and added a connector, you can make a connection your users can select when creating a new data source.

Users will see the new connector as an option during source creation, and can simply select it as they would any other source. See Define a SourceDefine a Visual Data Discovery Source.

Note: You are not prompted for credentials when you create a connection: they were provided as part of defining the connector.

Install and Set Up Composer's Metadata Store

Before the Composer components can be installed in the target server, you must install and set up the metadata store that is used in the Composer environment. This must occur regardless of the technique used to install the components (using the installation script or an alternative installation method).

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

- Composer v24.2 and earlier: PostgreSQL 12
- Composer v24.3 and later: PostgreSQL 16

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

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

Important: Before you attempt to upgrade to Composer, be sure to back up your metadata store. See Back Up The Metadata Store.

If you are upgrading Composer, consider the following possible PostgreSQL migration issues:

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

- No check for available disk space is performed during the upgrade. So the PostgreSQL automatic upgrade might fail if there is not enough available disk space to accommodate it. The Bootstrap installation procedure cannot predict the required amount of free disk space it will need.
- If your installation uses an external PostgreSQL instance for Composer's metadata store, you ***must*** disable the automatic PostgreSQL upgrade in the Bootstrap procedure before you upgrade Composer.

If either of these migration issues is true for your installation, you can disable the automatic PostgreSQL upgrade using the Bootstrap `ZOOMDATA_POSTGRES_DISABLE_UPGRADE` option. This option must be exported before running the Bootstrap procedure (for example by running `export ZOOMDATA_POSTGRES_DISABLE_UPGRADE=TRUE`). After running the Bootstrap procedure with the PostgreSQL upgrade disabled, you must manually upgrade your PostgreSQL metadata store to the appropriate version before you can use Composer.

Configuring PostgreSQL as the Composer involves defining authentication and connection parameters, restarting PostgreSQL, establishing login credentials for the Composer user, and creating metadata space for the Composer server and microservices. Complete the following steps to install and set up the PostgreSQL metadata store:

1. Set Up The PostgreSQL Metadata Store
2. Change Metadata Store Authentication To MD5 -- not necessary in Ubuntu environments
3. Create The Metadata Store User & Stores
4. Configure The Metadata Store For SSL

In addition, insightsoftware highly recommends that you optimize the PostgreSQL database instance after every upgrade. See Vacuum Composer's Metadata Store.

If you have performance problems with your PostgreSQL metadata store, examine the database settings related to automatic vacuuming, automatic analyzing, and the write-ahead log (WAL). See Optimize Composer's Metadata Store Performance.

Optionally, if the target server on which Composer will be installed does not have Internet access, read Obtain The Installation Package Without Internet Access.

Set Up the PostgreSQL Metadata Store

The instructions to set up PostgreSQL as Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

1. 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
```

2. 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
```

3. After installation, initialize the PostgreSQL database:

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

4. Start and enable the PostgreSQLmicroservice:

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

5. Confirm that the service started without errors:

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

   If necessary, start it:

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

6. 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
```

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

8. 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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

1. 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
```

2. 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`.

3. 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.

4. 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
```

5. 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".
```

Create the Metadata Store User & Stores

A Composer user must be established for the Postgres metadata store.

To create the Composer user for the Postgres metadata store, complete the following steps:

1. For all Linux operating systems, create the Composer user in PostgreSQL. Run the following command:

   ```
   sudo -u postgres -H psql -c "CREATE USER <db_username> WITH PASSWORD '<db_password>'"
   ```

   Substitute the PostgreSQL user name and password for `<db_username>` and `<db_password>`.

2. Create the stores that will hold the Composer metadata, upload data, keyset, and query engine data. Run the following series of commands, substituting the user name for `<db_username>`:

   ```
   sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata\" WITH OWNER <db_username>"
   sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata-upload\" WITH OWNER <db_username>"
   sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata-keyset\" WITH OWNER <db_username>"
   sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata-qe\" WITH OWNER <db_username>"
   ```

Change Metadata Store Authentication to MD5

If you installed Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

Change authentication for your metadata store to MD5

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

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

2. 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
```

3. Restart PostgreSQL. In CentOS environments, run:

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

Configure the Metadata Store for SSL

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.

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

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

2. Create a `.postgresql` subdirectory.

```
sudo mkdir .postgresql
```

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

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

Vacuum Composer's Metadata Store

insightsoftware highly recommends that you vacuum the Composer PostgreSQL metadata store after every upgrade. Vacuuming the metadata store optimizes it by maximizing database performance and minimizing the disk space it uses. The following simple procedure recollects metadata store statistics and "vacuums" the unused or dead rows in the database.

Note: This procedure blocks writing to the database. Consequently, work with the database is impossible until the procedure completes.

This procedure only works for a PostgreSQL database.

insightsoftware tested this procedure after an upgrade to Composer 5.9. on a machine with 2.6 GHz 6-Core Intel Core i7, 16 GB 2667 MHz DDR4, and SSD storage. The tested database contained 300 accounts, 4800 users, 60,000 sources, 9000 dashboards with 30 visuals each. The procedure took 5-10 minutes.

If you have performance problems with your PostgreSQL metadata store, examine the database settings related to automatic vacuuming, automatic analyzing, and the write-ahead log (WAL). See Optimize Composer's Metadata Store Performance.

Vacuum your PostgreSQL metadata store

1. Upgrade your version of Composer (if you have not already done so). This automatically upgrades the PostgreSQL metadata store.
2. Stop Composer and any process connecting to its PostgreSQL metadata store. See Stop ComposerSymphony Microservices.
3. Back up the PostgreSQL metadata store. See Back Up The Metadata Store.
4. Connect to the PostgreSQL database.
5. Run the following command in the console for the PostgreSQL database:

   ```
   VACUUM (FULL, ANALYZE);
   ```

   For more information about VACUUM, see *https://www.postgresql.org/docs/12/sql-vacuum.html*.
6. After vacuuming completes, start Composer. See Start ComposerSymphony Microservices.

Optimize Composer's Metadata Store Performance

If you have performance problems with your PostgreSQL metadata store, examine the database settings related to automatic vacuuming, automatic analyzing, and the write-ahead log (WAL).

Automatic Vacuuming (VACUUM)

In PostgreSQL, whenever rows in a table are deleted, the existing row (or tuple) is marked as "dead," but it is not physically removed. During an update, PostgreSQL marks the existing tuple as dead and inserts a new tuple. So a PostgreSQL UPDATE operation is a combination of a delete and an insert operation (DELETE + INSERT).

Dead tuples consume unnecessary storage and eventually, your PostgreSQL database is bloated with them. The VACUUM procedure reclaims the storage occupied by dead tuples. Bear in mind that the reclaimed storage space is never given back to the resident operating system. Instead it is just defragmented within the same database page, and the storage is available for reuse by future data inserts in the same table.

Bloating seriously affects PostgreSQL query performance. PostgreSQL tables and indexes are stored as an array of fixed-size pages (usually 8 KB in size). When a query request for rows is processed, the PostgreSQL instance loads these pages into the memory and the dead rows cause expensive disk I/O during data loading.

To check for dead tuples and the latest vacuum run, use the following query:

```
SELECT
  schemaname,
  relname,
  n_live_tup,
  n_dead_tup,
  last_autovacuum,
  last_vacuum
FROM pg_stat_user_tables
ORDER BY n_dead_tup
/ (n_live_tup
current_setting('autovacuum_vacuum_scale_factor')::float8
+ current_setting('autovacuum_vacuum_threshold')::float8)
DESC;
```

To check your automatic vacuuming settings, use the following query:

```
select
from pg_settings
```

```
where
name like '%autovacuum%';
```

For more information about adjusting automatic VACUUM settings, see:

- https://www.postgresql.org/docs/current/runtime-config-autovacuum.html
- https://habr.com/en/company/postgrespro/blog/486104/
- https://dzone.com/articles/tuning-postgresql-autovacuum-to-prevent-table-bloa

For information about vacuuming your Composer PostgreSQL metadata store, see Vacuum Composer's Metadata Store.

Automatic Analyzing (ANALYZE)

The PostgreSQL query planner relies on statistical information about the contents of tables to generate good plans for queries. These statistics are gathered using the ANALYZE command, which can be invoked by itself or as an optional step in VACUUM. It is important to have reasonably accurate statistics, or poor planning choices might degrade database performance. The PostgreSQL autovacuum daemon, if enabled, automatically issues ANALYZE commands whenever the content of a table has changed sufficiently. The daemon schedules ANALYZE strictly as a function of the number of rows inserted or updated; it has no knowledge of whether that will lead to meaningful statistical changes.

As with vacuuming for space recovery, frequent updates of statistics are more useful for heavily updated tables. But even for a heavily updated table, there might be no need for statistics updates if the statistical distribution of the data has not changing much. A simple rule of thumb is to consider how much the minimum and maximum values of the columns in the table change. For example, a time stamp column that contains the time of row update will have a constantly-increasing maximum value as rows are added and updated; such a column will probably need more frequent statistic updates than a column containing URLs for pages accessed on a website. The URL column might receive changes just as often, but the statistical distribution of its values probably changes relatively slowly.

For more information about the use of automatic ANALYZE processing, see:

- https://www.postgresql.org/docs/12/sql-analyze.html
- https://habr.com/en/company/postgrespro/blog/486104/

Write-Ahead Log (WAL)

PostgreSQL databases rely on a write-ahead log (WAL). All databases changes and transactions are written to the WAL first, and then to the data files. This provides durability, because if the database crashes, it can use the WAL to recover. It can read the changes from the WAL and reapply them to the data files. While this may double the number of writes, it may actually improve performance. Users only have to wait for the WAL (to be flushed to disk), while the data files are

modified only in memory and then flushed later in the background. PostgreSQL uses checkpoints in its sequence of transactions to identify points at which the data files and the heap have been updated fully with all the data written before the checkpoint. Checkpoints are points in the transaction stream before which the WAL is no longer needed for recovery, reducing disk space requirements and recovery time.

For more information about the WAL, see:

- https://habr.com/en/company/postgrespro/blog/494464/
- https://www.postgresql.org/docs/current/runtime-config-wal.html
- https://postgreshelp.com/postgresql-checkpoint/
- https://www.postgresql.org/docs/9.5/wal-configuration.html

Obtain the Installation Package Without Internet Access

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

If the target server for your Composer installation does not have Internet access, you can download the PostgreSQL repo package from another source and then transfer the files to the target server. Take the following steps to obtain the PostgreSQL repo package:

1. Access the PostgreSQL website, navigate to the appropriate version of PostgreSQL section, and locate the correct repo package for your target server.

   ```
   https://yum.postgresql.org/repopackages.php#pg12
   ```

2. Copy the link address of the PostgreSQL repo package. For example, right-click on the link and select the **Copy Link Address** option.

3. Paste the copied link address into the following command line and execute it:

   ```
   yum install -y <copied_link_URL>
   ```

   This command needs to be run from a server containing the same OS version as the target server for the Composer installation.

4. Run the following command line to download the dependencies for your selected PostgreSQL repo package:

   ```
   yum install --enablerepo=pgdg12 -y postgresql12-server --downloadonly --downloaddir=/<your_target_directory>
   ```

5. Transfer the PostgreSQL repo package to the target server.

6. Run the PostgreSQL repo package from the target server.

Post-Installation Options

After the Composer environment has been installed in your server, the following configuration options are available to you:

To	Read
Add an SSL Certificate. By default, Composer enables an *http* port. To enable *https*, you must add an SSL certificate to the Composer server.	Add an SSL Certificate
Disable SSL.	Disable the SSL Certificate in Composer
Use SQL-based connectors. Some SQL connectors require a JDBC driver to be configured before you can connect to your data source. You can download the driver from the vendor's site. Be aware that you need to download and configure JDBC drivers for the following Composer connectors as soon as you complete the Composer installation: ■ MemSQL ■ MySQL ■ Oracle ■ Amazon Redshift ■ Teradata	Add A JDBC Driver
Configure Composer memory settings.	Configure Memory Settings
Use Composer's sample data generator.	Manage the Real Time Sales Demo Source

Helm Chart for Composer

The Helm chart provided in the Composer Helm Chart repository is a lightweight way to configure and run Composer in Kubernetes. It includes sub-charts for use with 3rd-party components such as PostgreSQL, Consul, OpenTelemetery Collector, and more.

This Helm chart is publicly available in the Composer Helm Chart repository, and includes, by default, a 14 day trial Composer license. Optional components are disabled by default.

This topic covers:

Prerequisites

The following prerequisites are required for a successful deployment using Kubernetes and the Helm chart:

1. A Kubernetes cluster (versions 1.23-1.27)

i. Install Kubernetes or have access to a cluster.

ii. A local copy of `kubectl` configured to work with your cluster.

2. Helm (3.8-3.11). See Installing Helm.
3. Access to Docker Hub.

This Helm chart was developed with portability in mind. It relies only on Vanilla Kubernetes features and doesn't use any (cloud) vendor-specific features. Testing is performed in an Amazon EKS cluster and Minikube: you should need to make little to no modifications while working with other flavors of Kubernetes such as Azure AKS or Google GKE.

Required Resources

Composer is built using the microservices architecture. Since the list of required microservices (e.g. connectors) depends on your usage scenarios, it's difficult to estimate the exact resource requirements. However, for a deployment to be useful it must include an instance of Composer Web, a Query Engine instance, a Connector instance, and a Consul service for discovery. The usage of autoscaling will obviously impact the resource requirements.

Resource estimates for some common setups:

Minimal, one replica setup	Standard one replica setup (default)	Advanced, tree replica setup
Mandatory services: Web Server, Query Engine, Consul. Optional services: none. Connectors: one arbitrary connector. PostgreSQL metadata store location: external. Footprint: ■ CPU: 4.5 vCPU ■ RAM: 11.1 Gi ■ Storage: 11 Gi	Mandatory services: Web Server, Query Engine, Consul. Optional services: Data Writer. Connectors: PostgreSQL. PostgreSQL metadata store location: internal. Footprint: ■ CPU: 5.3 vCPU ■ RAM: 12.1 Gi ■ Storage: 19 Gi	Mandatory services: Web Server, Query Engine, Consul. Optional: Data Writer, Screenshot Service. Connectors: two arbitrary connectors. PostgreSQL metadata store location: external. Footprint: ■ CPU: 17.7 vCPU ■ RAM: 41.6 Gi ■ Storage: 31 Gi

For more exact estimates, you'll need to first determine the set of required components, then sum up their resource limits specified in the chart.

Working With Helm

This section explains the generic usage of Helm and Helm chart lifecycle applied to the Composer Helm Chart. For a generic introduction to Helm, see the Helm Quickstart Guide.

Obtain the Chart

Add the chart repository:

```
helm repo add composer https://composer-repo.logianalytics.com/helm-charts/stable
```

List the charts available:

- `helm repo update composer`
- `helm search repo composer`

Note: These instructions are valid for Composer 23.1 and later, chart version 1.5.2 and later.

Install with the Default Config

Run the following command to install the chart with the default config:

```
helm install <release-name> composer/composer
```

by running the first command the chart was released. Remember the release name, you will need it to interact with it later. Take a look at the output, it suggests the next steps like retrieving the address for ingress.

Important: insightsoftware recommends you do not use the command `helm install --replace`.

To keep track of a release's state, or to re-read the next steps, you can use the helm status:

```
helm status <release-name>
```

Default Config Caveats

Our default Ingress resource and Ingress Controller's configuration might not work for you. Follow this guide to re-configure our Ingress.

Customize the Chart Before Installing

Run the following command to list all the configurable parameters of the Composer chart and their default values:

```
helm show values composer/composer
```

Review the list of parameters to decide if anything needs to be changed. Use the guidance provided in Deciding on the Configuration to evaluate the parameters to help you decide if anything has to be changed.

Once you have identified the necessary changes, follow this generic guide on applying the changes to your Helm release.

Applying changes to your Helm release:

1. Create the `values.yaml` file and add YAML sections that you want to override to it. For example, in the following snippet, the PostgreSQL connector logging level is increased to `DEBUG`.

```
edc:
  postgresql:
```

```
    properties:
        logging.level.com.zoomdata: DEBUG
```

2. Install a new release with the override values.

```
helm upgrade <release-name> composer/composer -f values.yaml
```

Alternatively, upgrade an existing release.

Upgrade a Release and Recover on Failure

To apply a new configuration to an existing release use helm upgrade:

```
helm upgrade <release-name> composer/composer -f values.yaml
```

All affected pods will be automatically restarted.

Note: Some updates to the Helm chart values are incompatible with `helm upgrade`. If you have such an incompatibility, re-install the chart.

If something went wrong during the upgrade, review the revision history of your release:

```
helm history <release-name>
```

Identify the revision you want to rollback to, then run:

```
helm rollback <release-name> <revision-number>
```

Uninstall a Release

If a particular release is no longer required, use this command to uninstall:

```
helm uninstall <release-name> --keep-history
```

Using `--keep-history` is optional. If you use it, you can better audit the cluster's history, and even undelete a release using helm rollback.

Persistent Resources Considerations

Running `helm uninstall` on a release will remove most of the Kubernetes resources from the cluster. Some resources that constitute Composer metadata and re-usable configs are preserved. This allows you to create another Composer release with the same metadata like Sources, Visuals, Dashboards, etc. The preserved resources are PersistentVolumeClaims (PVC) and corresponding PersistenVolumes (PV) for:

1. Composer PostgreSQL metadata database folder. The PVC will have a name such as `data-<release-name>-postgresql-0` and the capacity of 8Gi. It will be `Bound` to a PV. These resources will be present only if the release was configured to create an internal PostgreSQL instance (the default setting). If you don't intend to reuse the metadata, you can delete this PVC and PV with `kubectl delete`. See PostgreSQL Metadata Store Configuration for information about reusing this metadata database in a new release.

2. The volume with connectors' drivers. The PVC will have the name `composer-shared-volume`. It will be `Bound` to the PV with the same name only if you configured Driver Drop-In for your connectors. If you did not configure this, it will be in the `Pending` state. If you don't intend to install a new release into the current namespace, you can delete the PVC. Delete the `composer-shared-volume` PV only if you don’t intend to install new Composer charts into this cluster.

3. The volume with the Consul service metadata. The PVC will have a name such as `data-<namespace>-<release-name>-consul-server-0` and the capacity of 10Gi. It will be `Bound` to a PV. Composer can re-create the service discovery information stored in Consul upon a new release, you can safely delete these resources.

Deep Chart Inspection and Customization

Helm pull provides a way to get all source files of the chart. This is useful when you want to perform a deep inspection of the chart. Use the following commands to download the chart from the repository and unpack it in a local directory:

```
helm repo update composer
helm pull composer/composer --untar
```

Once you have the source code of the chart, you’ll be able to modify it.

Important: Resort to this method only if there is no other way to achieve your goal via exposed configuration options.

Once you are done with the required modifications, you’ll be able to either repackage the chart into your custom chart or install it from the local directory:

```
helm install <release-name> . -f values.yaml
```

Note: If you require such a chart modification, please submit an enhancement request for the configurability gaps identified.

Debugging the Chart

Perform a dry run before installing a release:

```
helm install <release-name> composer/composer -f values.yaml --debug --dry-run
```

Adding the `--debug` option causes the server to render your templates. If rendering goes fine, then the resulting manifest file is returned. If not, an error with a stack trace is returned. Attach this error output when reaching out to support.

Helm also allows you to see what templates are installed on the server for a particular (successful) release:

```
helm get manifest <release-name>
```

Configuring the Chart

The Composer platform has lots of configuration options and some optional components. While the out-of-the-box configuration of the Helm chart provides some meaningful defaults, most likely you'll need to customize some important aspects like the list of required connectors or the need for horizontal autoscaling.

The Default Configuration

The default configuration installs a release with the following components:

Enabled Composer services:

- Mandatory: Web Server, Query Engine, Consul.
- Connectors: PostgreSQL.
- Optional: Data Writer.
- PostgreSQL metadata store location: internal.

Advanced capabilities:

- Default Ingress rule: `ENABLED`
- Tracing infrastructure: `DISABLED`
- Horizontal autoscaling: `DISABLED`
- Data Gateway: `DISABLED`

See the next section to learn how to customize it.

Deciding on the Configuration

This list covers the main decisions and recommended actions that you'll need to perform to determine your configuration:

1. Do you want to use an external, managed PostgreSQL instance as the Composer metadata store (the recommended setup)? Alternatively, you can let the chart install an internal PostgreSQL instance (the default).

i. Change the default PostgreSQL passwords if using an internal instance.

2. What is the list of data stores that you need to connect to? By default, only PostgreSQL connector is installed. Customize the list of connectors according to your needs.
3. Do you want to enable horizontal autoscaling?
4. Are you going to schedule Dashboard Reports that will deliver you dashboard screenshots periodically? If yes, enable the Screenshot Service component.
5. Are you going to upload flat files (e.g. CSV, JSON) for further analysis? If not, disable the Data Writer component to save cluster resources.
6. Do you have special requirements for the Ingress configuration? If yes, reconfigure our default Ingress.
7. Do you have a license that you want to apply to this deployment? If yes, inject the license during the chart installation.
8. How do you want to integrate your software into your observability infrastructure?

Once you decide on your target configuration, prepare corresponding override values for the Helm chart and put them into your `values.yaml`. Override values fall into two categories:

1. Composer application configuration properties injected into services running inside pods/containers:

i. Regular application properties

ii. Credentials

2. Configuration for Kubernetes resources governed by Helm.

The next section covers Composer application configuration.

Injecting Composer Configuration Properties

Application Properties

There are two categories of Composer application configuration properties that differ in their sensitivity, hence the way they are specified in the `values.yaml`:

- Regular application configuration properties that don't contain sensitive data and are not exposed as separate Helm chart parameters.
- Sensitive application configuration properties, such as database credentials, that are exposed as Helm chart parameters.

Regular Application Properties

Regular properties for a Composer service are specified as the `properties` map within the object representing this service in the `values.yaml` file. Each key-value pair in this map represents a property name and value.

For example, the following snippet shows a number of regular properties with names starting with `mail.` that specify mail server configuration for Zoomdata Web component:

```
zoomdataWeb:
  properties:
    mail.smtp.host: "email-smtp.us-east-1.amazonaws.com"
    mail.smtp.port: 465
    mail.smtp.auth: true
    mail.smtp.ssl.enable: true
    mail.smtp.ssl.protocols: "TLSv1.2"
    mail.from: "admin@example.com"
```

Regular properties go to ConfigMaps when Composer is installed in a Kubernetes cluster and are treated as if they are specified in regular properties files. The names of the properties should have the same names as in regular properties files (usually found under `/etc/zoomdata` or `/opt/zoomdata/conf` for Linux-based deployments).

Sensitive Application Properties

Each sensitive property for a Composer service is specified as a separate parameter in the object representing this service in the `values.yaml` l file. For example, parameters `zoomdataWeb.mailLogin` and `zoomdataWeb.mailPassword` below are sensitive properties that specify, correspondingly, login and password for the mail server configured for Zoomdata Web component:

```
zoomdataWeb:
  mailLogin: composer
  mailPassword: ChangeMe12345
```

It is not possible to override sensitive configuration properties with regular ones. For example, in the snippet below, regular properties `mail.login`and `mail.password` will be ignored by the Zoomdata Web component:

```
zoomdataWeb:
  mailLogin: composer
  mailPassword: ChangeMe12345
  properties:
    mail.login: ignored
    mail.password: ignored
```

Sensitive properties go to Secrets when Composer is installed in a Kubernetes cluster. They are usually injected into corresponding services through environment variables.

Enable the Data Gateway

Use a data gateway in your environment to connect to information securely outside of your environment. Use a gateway client to authenticate your connection and make the data available to users. To enable the data gateway, add these values to the `values.yaml`.

```
# Data Gateway Service configuration
dataGatewayService:
  # Is Data Gateway Service enabled
  enabled: true
```

List of Properties Available as Helm Parameters

Parameter in Values File	Description	Injected As	Corresponding Application Property
Zoomdata Web			

Parameter in Values File	Description	Injected As	Corresponding Application Property
zoomdataWeb.contextPath	Web context path	Env variable	server.servlet.context-path
zoomdataWeb.metadataDbUrl	Metadata database URL	Env variable	spring.datasource.url
zoomdataWeb.metadataDbUsername	Metadata database username	Secret	spring.datasource.username
zoomdataWeb.metadataDbPassword	Metadata database password	Secret	spring.datasource.password
zoomdataWeb.uploadDbUrl	Upload database URL	Env variable	upload.destination.params.jdbc_url
zoomdataWeb.uploadDbUsername	Upload database username	Secret	upload.destination.params.user_name
zoomdataWeb.uploadDbPassword	Upload database password	Secret	upload.destination.params.password
zoomdataWeb.keysetDbUrl	Keyset database URL	Env variable	keyset.destination.params.jdbc_url
zoomdataWeb.keysetDbUsername	Keyset database username	Secret	keyset.destination.params.user_name
zoomdataWeb.keysetDbPassword	Keyset database password	Secret	keyset.destination.params.password
zoomdataWeb.userAuditingDbUrl	User auditing database URL	Env variable	user-auditing.destination.params.jdbc_url
zoomdataWeb.userAuditingDbUsername	User auditing database username	Secret	user-auditing.destination.params.user_name
zoomdataWeb.userAuditingDbPassword	User auditing database password	Secret	user-auditing.destination.params.password
zoomdataWeb.mailLogin	Mail server login	Secret	mail.login
zoomdataWeb.mailPassword	Mail server password	Secret	mail.password
zoomdataWeb.adminPassword	Password for the built-in admin user. If not set, you'll be prompted to set it on the first login. Supported in v23.2 and later only. Setting this value for earlier versions will have no effect.	Secret	admin.password
zoomdataWeb.supervisorPassword	Password for the built-in spervisor user. Defaults to the value of	Secret	supervisor.password

Parameter in Values File	Description	Injected As	Corresponding Application Property
	`adminPassword`. Supported in v23.2 and later only. Setting this value for earlier versions will have no effect.		
Query Engine			
`queryEngine.dbEnabled`	Use Query Engine database for storing query results cache when `true`		
`queryEngine.dbUrl`	Query Engine database URL	Env variable	`spring.qe.datasource.jdbcUrl`
`queryEngine.dbUsername`	Query Engine database username	Secret	`spring.qe.datasource.username`
`queryEngine.dbPassword`	Query Engine database password	Secret	`spring.qe.datasource.password`

JVM Properties

There are two categories of properties available for all services:

- Properties to configure Composer services heap size.
- A catch-all property that allows passing arbitrary Java options to each service.

Heap Size Configuration

The following Helm chart parameters are used to control heap size for Composer Java services:

Property in Values File	Description	Scope	Java Option
`heapSizeMin`	Initial JVM heap size.	Service	`-Xms`
`heapSizeMax`	Maximum JVM heap size.	Service	`-Xmx`

The following snippet shows the default heap size configuration for connectors:

```
edc:
  common:
    heapSizeMin: "256M"
    heapSizeMax: "512M"
```

The following snippet shows how to override the default settings for the PostgreSQL connector:

```
edc:
  postgresql:
    heapSizeMin: "512M"
    heapSizeMax: "1024M"
```

Passing Arbitrary Java Options to Services

A catch-all property for passing arbitrary Java options is called `additionalJavaOpts` and is supported for each Composer service. For example, this is how to enable garbage collector logging for Query Engine:

```
queryEngine:
  additionalJavaOpts: "-verbose:gc"
```

Injecting Credentials

Some services might need additional credentials provided in separate files, like Java trust stores and Kerberos keytab files. To inject such credentials, you need to use additional Kerberos Secrets and Volumes.

For example, let's consider how to inject a Java trust store into the Elasticsearch 8 connector:

1. Create a trust store Secret:

   ```
   kubectl create secret generic <secret-name> --from-file=truststore.p12
   ```

2. Configure the Elasticsearch 8 connector to mount this Secret as a volume and access the trust store from its file system:

   ```
   edc:
     elasticsearch-8.0:
       enabled: true
   ```

```
    additionalJavaOpts: "-Djavax.net.ssl.trustStore=/opt/zoomdata/security/truststore.p12 -
Djavax.net.ssl.truststoreType=PKCS12 -Djavax.net.ssl.trustStorePassword=<truststore-password>"
    extraVolumeMounts:
      - name: truststore
        mountPath: /opt/zoomdata/security/truststore.p12
        subPath: truststore.p12
        readOnly: true
    extraVolumes:
      - name: truststore
        secret:
          secretName: <secret-name>
```

3. Install or upgrade the Helm chart.

PostgreSQL Metadata Store Configuration

Composer stores its metadata in a PostgreSQL database. The Helm chart lets you customize the location of the metadata store database. You have two options:

- Default: The chart installs an *internal* instance of PostgreSQL.
- Recommended: Point Composer to an *external*, managed instance of PostgreSQL.

Internal PostgreSQL Metadata Store

Configuring Credentials

The default chart config is supplied with the hardcoded database username and password to simplify the initial installation of the chart:

```
defaultDbUsername: "zoomdata"
defaultDbPassword: "ChangeMe12345"
```

Important: Change the username and password for your production release if you choose to use the internal PostgreSQL instance.

Reusing Existing Internal Metadata Database for New Release

The usage of an *internal* instance of PostgreSQL will create a PersistentVolumeClaim (PVC) and corresponding PervistentVolume (PV) with the PostgreSQL data dir folder. Uninstalling such a chart will keep these PVC and PV so that you can re-use the metadata for another release. The usual name of the PVC is a name such as `data-<release-name>-postgresql-0`. To reuse the metadata:

1. Add the following config to your `values.yaml` to point a new release to the existing PVC:

```
postgresql:
  primary:
    persistence:
      existingClaim=data-<release-name>-postgresql-0
```

2. Install or upgrade the Helm chart.

External PostgreSQL Metadata Store

To configure the chart to use an *external* PostgreSQL-compatible database system:

1. Disable the creation of the built-in PostgreSQL instance by adding this config to your `values.yaml`.

```
postgresql:
  enabled: false
```

2. Create the next databases in your database system.

```
zoomdata
zoomdata-upload
zoomdata-keyset
zoomdata-user-auditing
zoomdata-qe
```

   We recommend that you also create a separate PostgreSQL user for Composer databases. See Create The Metadata Store User & Stores.

3. Define the following Helm chart parameters in your `values.yaml` and set them according to your database setup.

```
zoomdataWeb:
  metadataDbUrl: ""
  metadataDbUsername: ""
  metadataDbPassword: ""
  uploadDbUrl: ""
  uploadDbUsername: ""
```

```
  uploadDbPassword: ""
  keysetDbUrl: ""
  keysetDbUsername: ""
  keysetDbPassword: ""
  userAuditingDbUrl: ""
  userAuditingDbUsername: ""
  userAuditingDbPassword: ""

queryEngine:
  dbUrl: ""
  dbUsername: ""
  dbPassword: ""
```

If you only define a single user, then you can skip individual `*DbUsername` and `*DbPassword` parameters and set only the following two defaults.

```
defaultDbUsername: ""
defaultDbPassword: ""
```

4. Install or upgrade the Helm chart.

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Apply Licenses

By default, Composer comes with a trial license valid for for 14 days. The first start up of Composer activates this license and generates a unique `installation id` for your deployment.

Once you are done with the trial, you'll need to request a new license key using the generated `installation id` and then apply the key to your deployment. There are multiple ways to apply a license key: via UI, using the licensing API, or by setting Helm override values. This article describes setting Helm override values.

- Replace An Existing License
- Apply An OEM License

Caveats of Different License Configuration Options

If you apply your license using Composer's user interface or the licensing API, you can only scale the Web Server service to more than three replicas, if allowed by the license's restrictions. The replica count for all other services is limited to three replicas.

If you apply your license using Helm values, you should use Helm to update the license; it won't be possible to update the license via UI or API. See Manage License Keys Using Configuration Properties Or Environment Variables.

Replace an Existing License

1. Obtain a new license key for your existing deployment.
2. Add the following properties to your `values.yaml`:

```
licenseKey: "your-license-key"
licenseInstallationId: "your-installation-id"
```

3. Upgrade your existing Helm release.

   The license is applied to all Composer services.

Apply an OEM License

Regular license keys are tied to the `installation id` that is unique for your deployment. The ability to have an arbitrary number of deployments can be provided by an OEM license.

An OEM license allows you to have an unattended installation because you won't need to request a license key for each unique `installation id`.

If you have an OEM license key, set only the `licenseKey` property in your `values.yaml`. Install a new Helm chart release or upgrade an existing one.

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Scaling Configuration

When deployed in Kubernetes, Composer supports both vertical (adding resources) and horizontal (adding pods) scaling.

This topic covers:

Horizontal Autoscaling

By default, the Helm chart comes with horizontal autoscaling disabled. Follow this guide to learn more about Horizontal Pod Autoscaling and how to enable it.

Manual Scaling

If you decide not to enable autoscaling you can still scale the resources/pods manually.

Manual Horizontal Scaling

To set the number of replicas for a Composer service such as `zoomdataWeb` do the following:

1. Add the desired number of replicas to the `values.yaml`:

```
zoomdataWeb:
  replicaCount: 2 # default is 1
```

2. Install a new helm chart release or upgrade an existing one.

Important: Don't use `kubectl` to change the number of replicas for Composer pods. These changes will be overwritten on the next usage of Helm.

Manual Vertical Scaling

Configure each pod's resource manually in the corresponding block of the `values.yaml` file, but it's always recommended to prefer horizontal over vertical scaling. Improper change of the resources may lead to application failure on startup or instability during the operation.

Since Composer consists of a number of Java-based microservices, apart from generic pod resource types (i.e. CPU and memory), Composer services also expose properties for configuring application JVM memory: `heapSizeMin` and `heapSizeMax`. For example, here are the resource-related defaults for the `zoomdataWeb`service:

```
zoomdataWeb:
  # Initial JVM heap size
  heapSizeMin: "4G"
  # Maximum JVM heap size
  heapSizeMax: "4G"
  resources:
    limits:
      memory: "6Gi"
    requests:
      cpu: "3.5"
      memory: "6Gi"
```

If you decide to customize these properties, please follow these general recommendations:

1. Memory request and limit should be the same.
2. Memory request should be at least 25% bigger than JVM's maximum heap size to account for non-heap memory consumed by Java applications.
3. CPU limit is empty.

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Horizontal Pod Autoscaling

Enable the Horizontal Pod Autoscaler in your environment with a new or existing instance of Prometheus and Prometheus Adapter and our custom metrics for Kubernetes to adjust the number of replicas used, scaling up or down depending on the workload.

This topic covers:

Overview - About HPA

The Horizontal Pod Autoscaler (HPA) is a Kubernetes feature that dynamically adjusts the number of replicas (pods) in a deployment or replica set based on the observed metrics. It helps maintain optimal resource utilization and ensures that the application can handle varying levels of workload efficiently. HPA continuously monitors the specified metrics and automatically scales the number of pods up or down to meet the desired performance targets.

The Composer HPA implementation works by utilizing custom metrics provided by Prometheus Adapter and Prometheus. Prometheus collects application metrics from each pod. Prometheus Adapter makes these metrics available to the Horizontal Pod Autoscaler, which uses them to make scaling decisions.

Composer's HPA Metrics

Three custom metrics are utilized in Composer environments:

- `cpuAverageUtilization` - CPU average utilization measures the average CPU utilization of the application over a specific period. It provides insights into how much CPU capacity the application requires to handle its workload. The default value is 80%.

- `memoryAverageUtilization` - JVM heap average utilization monitors the average utilization of Java Virtual Machine (JVM) heap memory. It measures the proportion of heap memory consumed by the application over a defined interval. HPA utilizes this metric to scale the application based on memory consumption, preventing memory-related issues and optimizing resource allocation. The default value is 90%.

- `threadsQueueAverageSize` - The number of threads in the queue to the application reflects the number of HTTP requests waiting in the application's processing queue. This metric provides visibility into the application's ability to handle incoming requests promptly. HPA automatically adjusts the number of replicas to ensure sufficient processing capacity and minimize request queuing. The default value is 400 requests in the queue.

These metrics work in conjunction to ensure optimal performance and resource utilization.

HPA Configuration

HPA is disabled by default. Once you've enabled it, it's preconfigured with reasonable defaults.

Enable HPA

Before you enable HPA, see Caveats of the Default Config.

1. Add the following to your `values.yaml`:

```
hpa:
  enabled: true
```

2. Install a new helm chart release or upgrade an existing one.

> **Note:** When you enable HPA, 8GB of additional storage for Prometheus is required if you install it using the Helm chart.

Global and Per-Service HPA Configuration

All Composer HPA configuration properties can be specified on the global and per-service levels. The service-level configuration has higher priority and overrides the global one.

For example, here are the steps to change the number of replicas on the global level:

1. Set the value of the minimum and/or maximum number of replicas in your `values.yaml`:

```
hpa:
  minReplicas: 2  # default is 1
  maxReplicas: 4  # default is 3
```

2. Install a new helm chart release or upgrade an existing one.

   This configuration will be applied to all Composer services.

Note: The default number of maximum replicas is 3. A higher value requires a corresponding license.

HPA can also be configured on the per-service level, for example:

```
hpa:
  enabled: true

zoomdataWeb:
  hpa:
    maxReplicas: 5

dataWriter:
  hpa:
    enabled: false
```

In the above example, HPA is enabled for all services except for the Data Writer. The maximum number of replicas for Zoomdata Web is increased to 5 and defaults to 3 for all other services.

Additional Properties

The full list of configuration properties can be found in the chart values under the hpa object:

```
helm show values composer/composer
```

While enabling HPA and customizing the minimum and maximum replicas is a common practice, it's important to exercise caution when modifying other properties. Many properties are preconfigured for optimal performance. Modifying these properties without a clear understanding of their implications may lead to unexpected issues or degraded performance.

Caveats of the Default Config

By default, when you enable HPA for Composer, Composer's helm chart will try and install an instance of Prometheus and Prometheus Adapter into the namespace of Composer's deployment. This is not a best practice, but rather a convenient getting-started setup for those who are new to Kubernetes and who have no other things running in their clusters.

The limitation is that only one service can implement Kubernetes Custom Metrics API. In our case it's Prometheus Adapter. By installing Prometheus Adapter from our helm chart it will come pre-configured with Composer-specific rules and it'll be difficult to re-use it in other Kubernetes applications. Also, you won't be able to install another instance of the Composer application or another Prometheus Adapter into the same cluster. The recommended approach is to have a cluster-wide Prometheus and Prometheus Adapter that can serve all the HPAs in the cluster.

Cluster-wide Prometheus and Prometheus Adapter

Installing an instance of Prometheus and Prometheus Adapter as part of the Composer's deployment is not a best practice. For a production deployment, we recommend that you deploy a cluster-wide instance of Prometheus and Prometheus Adapter and refer to them from different Composer deployments.

Note: If you already have Prometheus and the Prometheus Adapter installed and configured you'll only need to add Composer rules to your Prometheus Adapter configuration (step 3) and disable Prometheus and Prometheus Adapter installation in Composer's helm chart (steps 4 and 5).

Installing and configuring Prometheus and Prometheus Adapter:

1. Create a `monitoring` namespace in your Kubernetes cluster:

```
kubectl create namespace monitoring
```

2. Install Prometheus into the `monitoring` namespace. We don't require any special configurations for Prometheus, all our pods will be automatically discovered and scraped by Prometheus with the default configuration:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install prometheus prometheus-community/prometheus -n monitoring
```

3. Install the Prometheus Adapter into the `monitoring` namespace. The Prometheus Adapter requires two pieces of configuration:

- The URL and port of the Prometheus server. If they are installed in the same namespace (for Composer, the `monitoring`namespace), it'll be simply `http://prometheus-server:80`.

- The rules for converting Composerapplication metrics coming from Prometheus to custom Kubernetes metrics required by the HPA configuration. Copy the following snippet into a file called `prometheus-adapter.yml`:

```
prometheus-adapter:
  prometheus:
    url: http://prometheus-server
    port: 80
  rules:
    custom:
      - seriesQuery: '{__name__=~"jvm_memory_(used|max)_bytes",area="heap"}'
        resources:
          overrides:
            namespace: { resource: "namespace" }
            pod: { resource: "pod" }
        name:
          matches: '^jvm_memory_(used|max)_bytes$'
          as: "jvm_heap_usage_percent"
        metricsQuery: 'round(sum(jvm_memory_used_bytes{area="heap", <<.LabelMatchers>>}) by (<<.GroupBy>>) * 100 / sum(jvm_
memory_max_bytes{area="heap", <<.LabelMatchers>>}) by ((<<.GroupBy>>))'
      - seriesQuery: 'system_cpu_usage'
        resources:
        overrides:
          namespace: { resource: "namespace" }
          pod: { resource: "pod" }
        name:
          matches: 'system_cpu_usage'
          as: 'cpu_usage_percent'
        metricsQuery: 'round(sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>) * 100)'
      - seriesQuery: 'jetty_threads_jobs'
        resources:
          overrides:
            namespace: { resource: "namespace" }
            pod: { resource: "pod" }
        name:
          matches: 'jetty_threads_jobs'
          as: 'jetty_threads_jobs'
        metricsQuery: 'sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)'
```

Run the following commands to install the Prometheus Adapter into the `monitoring` namespace:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install prometheus-adapter prometheus-community/prometheus-adapter -f prometheus-adapter.yml -n monitoring
```

4. Update the Composer helm chart configuration file `values.yaml`:

```
# Make sure HPA is enabled
hpa:
  enabled: true

# Disable installation of Prometheus
prometheus:
  enabled: false

# Disable installation of Prometheus Adapter
prometheus-adapter:
  enabled: false
```

5. Install a new helm chart release or upgrade an existing one.

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Ingress Configuration

By default, the chart creates an Ingress resource with the name `default-ingress`. Also, it relies on the ingress-nginx sub-chart to install an Ingress Controller with the default name `nginx`. This might not work with your existing Ingress Controller. To resolve this issue, override the config for the ComposerIngress in your `values.yaml`:

```
ingress:
  enabled: true
  installController: false
  className: <your-ingress-controller-name>
  ingressName: default-ingress
```

This config disables installation of the `nginx` Ingress Controller and points Composer's Ingress to your controller. If such a config is still not versatile enough for your needs, disable both the Ingress and Ingress Controller and create your own Ingress.

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Screenshot Service Configuration

The Screenshot Service component is an optional component for , disabled by default. Enable if you are going to schedule Dashboard Reports.

To enable, add the following configuration to your `values.yaml`:

```
screenshotService:
  enabled: true
```

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Data Writer Configuration

The Data Writer component is an optional component that is enabled by default. It's required by the following Composerfeatures:

1. Uploading flat files (e.g. CSV, JSON) for further analysis.
2. Landing streaming data via Upload API.
3. Performing multipass and multisource filtering of your data with the help of Keysets.
4. Tracking end users' access to data via User Auditing.

If none of these features are required, add the following configuration to your `values.yaml` to disable it:

```
dataWriter:
  enabled: false
```

Then install a new helm chart release or upgrade an existing one.

Note: Find more information about Composer and Kubernetes here: Running Composer In Kubernetes.

Add an SSL Certificate

Composer supports SSL certificates so that a secure connection between the Composer server and the browser can be established. In particular, Composer supports two common formats for SSL certificates - JKS and PKCS12. For information on creating a keystore or Certificate Signing Request (CSR) for use with Composer, 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 Composer directory and the proper parameters be added to the `zoomdata.properties` configuration file.

Perform the following steps:

1. From your terminal, SSH to your Composer server.
2. Stop Composer microservices. See Stop ComposerSymphony 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. Composer supports the JKS and PKCS12 certificate formats.
6. Save and exit the `.properties` file.
7. Start Composer microservices. See Start ComposerSymphony Microservices.

After the Composer 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 Composer microservices. See Restart ComposerSymphony Microservices.

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

Disable the SSL Certificate in Composer

You can disable the SSL certificate in Composer 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 Composer Server.
2. Stop Composer microservices. See Stop ComposerSymphony 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 Composer services. See Start ComposerSymphony Microservices .

After the Composer 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://<Composer-IP-address>/composer
```

Otherwise, use the following URL format:

```
http://<Composer-IP-address>:8080/composer
```

Configure the Firewall (for CentOS)

Refer to the topic Configure The Firewall for setup instructions.

Set Up the Screenshot Microservice

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

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 Composer 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 Composer 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 Composer 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

Install the Screenshot microservice

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

Test the Screenshot microservice

1. Open a web browser to Composer. Obtain the ID for a dashboard to use for testing. Log in to Composer 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 Composer 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

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 ComposerSymphony Microservices. See also Icons Not Reverting To Defaults After Screenshot Microservice Disabled

Timeouts

Composer 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 Composer 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 Composer 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 Composer 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 Composer and Screenshot microservices. See Restart ComposerSymphony Microservices.

Self-signed Certificate

If Composer 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 Composer 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 ComposerSymphony 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

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 Composer server microservice after the change (see Restart ComposerSymphony 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 a Composer Distributed Environment

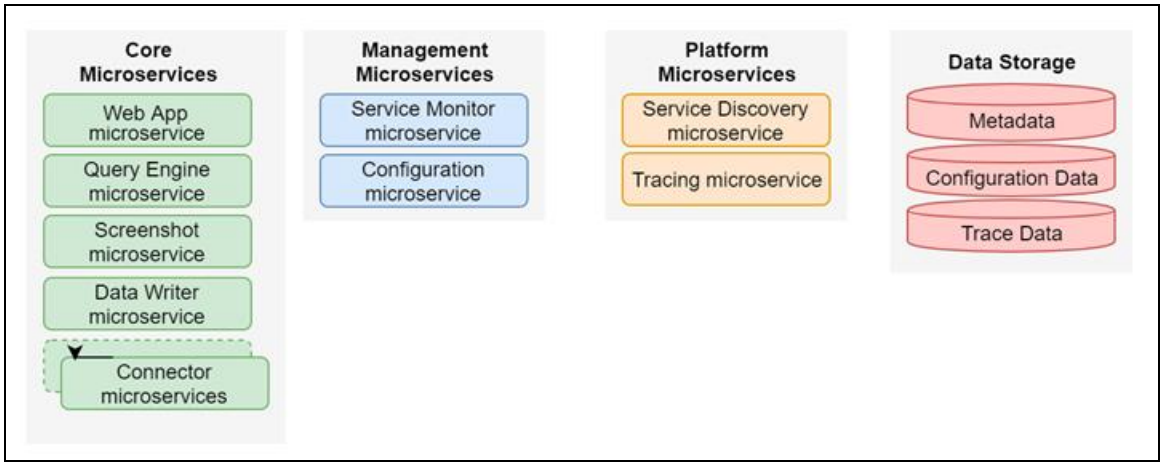
The Composer platform is architected as a set of loosely coupled Java microservices. They can be subdivided into the following categories when considering application availability:

- Core microservices - application components of Composer. These services provide key functionality in the application.
- Platform microservices - services that facilitate the operation and coordination between Composer's core microservices.
- Management microservices - services that provide views and controls for administration and troubleshooting of Composer's core microservices.

The communication protocols used by the microservices include WebSockets (for realtime bidirectional communication) and HTTP/HTTPS.

A metadata store is also needed to store Composer's metadata, application configuration data, and application trace data. Composer centralizes its metadata store in a relational database and includes an internal messaging queue.

The microservices and metadata store are depicted in the following diagram:



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

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

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

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

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

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

  For more information, see Configure A High Availability Environment.

Both load balancing and high availability require a centralized metadata store and a multi-node license. In a high availability environment, the metadata store can be clustered.

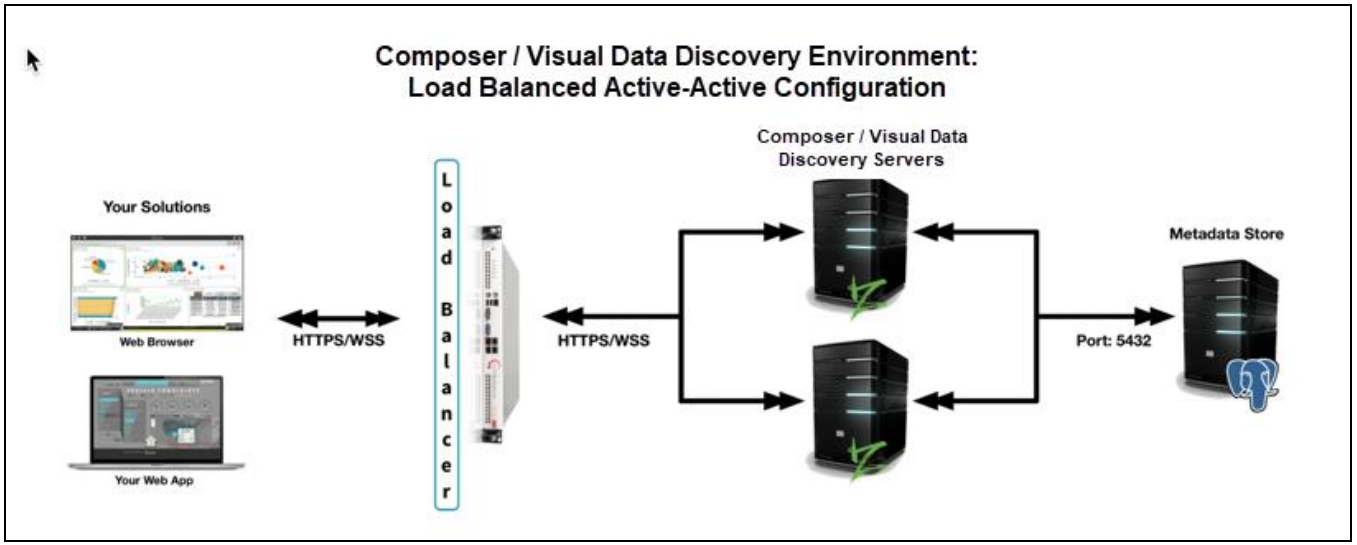
The configuration of load balancing and high availability are similar, but the high availability configuration is more complex in regards to its Consul configuration. To configure either, select one of the links above.

If you already have a distributed Composer environment in place, read Upgrade A Composer Distributed Environment to upgrade it.

Configure the Composer Server Behind a Load Balancer

Load balancing helps you to scale Composer 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.



Composer 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 Composer, when it runs standalone, uses its own dedicated PostgreSQL server as the metadata store (in other words, PostgreSQL was installed as part of the Composer installation process). If you are running in a high availability environment, you will need to use a high availability (clustered) PostgreSQL data store. See Configure A High Availability Environment.

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

The metadata store should be centrally installed, accessible by all Composer 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 Composer and are now trying to set up a distributed environment, there is a good chance that the metadata store was installed with your existing Composer instance or instances.

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

- If the metadata store has already been installed, identify its host name and port. Then follow the steps for upgrading a distributed environment. See Upgrade A Composer Distributed Environment.

- 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 Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

- 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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

- 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 Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

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

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

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 Composer On Your Distributed Servers

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

Deploy multiple Composer 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 Composer PostgreSQl metadata store

ii. `<postgres-db-username>` and `<postgres-db-password>` are the user name and password required to access the Composer 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 Composer 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 Composer instance (node) in your Composer 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 Composer 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 Composer 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 Composer, such as vocabulary files, manually copy them to each instance of Composer in your distributed environment.

Configure a High Availability Environment

A high availability environment includes multiple Composer nodes, each with its own set of microservices. This ensures that a microservice is available at all times, somewhere in the cluster.

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

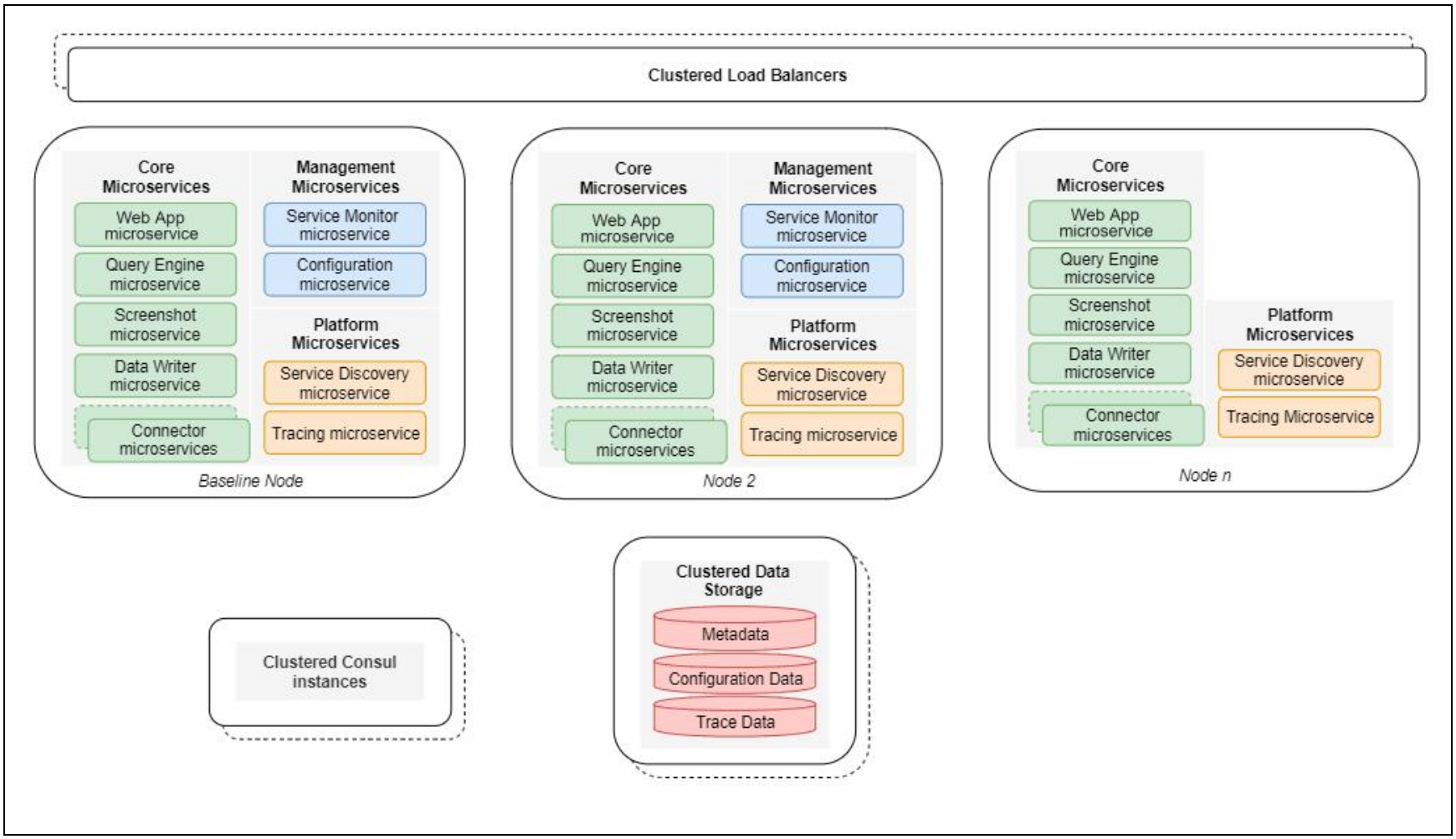
Different numbers of different types of microservices can be defined for the Composer nodes, although at least two of each must be installed. You can monitor all microservices and collect diagnostic trace information for them using the Service Monitor in conjunction with your distributed tracing services. 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.

In addition:

- At least one configuration server (`config-server`) must be installed and started in the high availability environment.
- Only one Service Monitor (`admin-server`) should be installed and started in the high availability environment. The Service Monitor is not required, but is helpful.
- The Consul's UI is available through port 8500 and is a lighter alternative to the Service Monitor. To ensure there is no single point of failure, Composer recommends that the Consul instance (`zoomdata-consul`) and the PostgreSQL metadata repository be configured as clusters external to the Composer nodes. For information, see https://www.consul.io/commands/join (`zoomdata-consul join <ip-address>`) and https://www.postgresql.org/docs/12/high-availability.html.
- Intra-service communication must be enabled by making any necessary networking (ports) and firewall changes.

Note: Composer's microservice architecture supports client-side load balancing for all microservices except Composer Web.

The following diagram depicts a classic high availability setup. (In this diagram, the Consul cluster is configured separately.)



Important: Configuring Composer 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 set up a high availability Composer environment. (To upgrade an existing high availability environment, see Upgrade A Composer Distributed Environment).

Step 1: Identify or Install the Postgres Metadata Store

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

The metadata store should be centrally installed, accessible by all Composer 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 Composer and are now trying to set up a distributed environment, there is a good chance that the metadata store was installed with your existing Composer instance or instances.

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

- If the metadata store has already been installed, identify its host name and port. Then follow the steps for upgrading a distributed environment. See Upgrade A Composer Distributed Environment.

- 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. Set up the Metadata Store - Necessary for all installations.

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

i. PostgreSQL Setup for CentOS Environments

ii. PostgreSQL Setup for Ubuntu Environments

PostgreSQL Setup for CentOS Environments

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

a. 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
```

b. 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
```

c. After installation, initialize the PostgreSQL database:

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

d. Start and enable the PostgreSQLmicroservice:

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

e. Confirm that the service started without errors:

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

If necessary, start it:

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

f. 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
```

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

h. 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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

a. 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
```

b. 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`.

c. 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.

d. 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
```

e. Test the PostgreSQL connection.

i. 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
```

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

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

iii. Start PostgreSQL using this command.

```
$ psql
```

iv. 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".
```

v. 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
```

vi. 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 Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

Change authentication for your metadata store to MD5

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

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

b. 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
```

c. 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.

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

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

b. Create a `.postgresql` subdirectory.

```
sudo mkdir .postgresql
```

c. 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** - Not necessary if installing a high availability environment in the cloud.

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

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

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

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

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

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

```
listen_address='*'
```

c. Edit the `pg_hba.conf file`:

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

d. 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
```

e. Save the `pg_hba.conf` file.

f. Restart the PostgreSQL service:

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

E. **Create user and Database** - Not necessary if installing a high availability environment in the cloud.

The PostgreSQL data store must be configured so it is available to all instances in a distributed environment. For more information about PostgreSQL high availability clustering, see https://www.postgresql.org/docs/12/high-availability.html.

Create users and databases in the PostgreSQL data store so they are available to all Composer instances

```
CREATE USER zoomdata WITH PASSWORD 'StrongZoomdataPassword';
CREATE DATABASE "zoomdata" WITH OWNER zoomdata;
CREATE DATABASE "zoomdata-upload" WITH OWNER zoomdata;
CREATE DATABASE "zoomdata-keyset" WITH OWNER zoomdata;
CREATE DATABASE "zoomdata-auth" WITH OWNER zoomdata;
CREATE DATABASE "zoomdata-qe" WITH OWNER zoomdata;
CREATE DATABASE "zoomdata-user-auditing" WITH OWNER zoomdata;
```

Step 2. Install Composer On Your Distributed Servers

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

Deploy multiple Composer 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 Composer PostgreSQl metadata store

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

2. In high availability (HA) environments only, run the following commands for each instance to set up the environment variables for the Consul configuration:

```
export ZOOMDATA_CONSUL_ADDRESS=<instance_ip>:8500
export ZOOMDATA_CONSUL_CLIENT=0.0.0.0
```

where `<instance_ip>` is the IP address of the Composer instance.

3. 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*.

4. 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 Composer version and release.

5. 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>
```

6. 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
```

7. Repeat these steps for every Composer instance (node) in your Composer cluster. For additional information on how many nodes to deploy in a high availability environment, see Determine How Many Nodes To Deploy.

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 Composer 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 Composer 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 Composer, such as vocabulary files, manually copy them to each instance of Composer in your distributed environment.

Step 5. Configure Consul Services for High Availability

You can install individual Consul instances locally for each Composer node and then configure the Consul instances as a cluster for the Composer high availability environment. Before you do, make sure you are familiar with the general clustering techniques used for Consul. See https://www.consul.io/docs/install/bootstrapping for more information.

The primary advantage of using a Consul cluster in a Composer high availability environment is that it requires fewer configuration changes to Composer components. Each component will look for the Consul on the localhost interface at port 127.0.0.1. The only configuration necessary is to the Consul cluster nodes themselves.

Another advantage is that each node of the Composer high availability cluster will use several discovered instances of the same component to balance the load and become more tolerant of any component failures.

Configure a Consul cluster for a Composer high availability environment

1. Install all the individual Consul instances on each Composer node. This happens automatically when you use the Composer bootstrap installation procedure.
2. Make sure that a firewall is opened in your environment for ports 8500, 8300, 8301, and 8302 on all hosts that will form the Consul cluster.
3. Edit the Consul custom configuration file `consul.json` on each Composer node.

   ```
   vi /etc/zoomdata/consul.json
   ```

   If you did not install the Consul instances using the Composer bootstrap installation procedure, its custom configuration file might have a different name and location.
4. Configure the Consul custom configuration file for each Consul instance so it includes these lines:

   ```
   {
   .........
   "node_name": "<node-name>"
   "bind_addr": "0.0.0.0",
   "bootstrap": false,
   "client_addr": "0.0.0.0",
   "retry_join": [
   "<host-ip-address-1>",
   "<host-ip-address-2>",
   "<host-ip-address-n>"
   ],
   "server": true
   .........
   }
   ```

   The `<node-name>` settings for each Consul node should be unique within the cluster. Each Consul instance in the cluster should have a different name.

   A bind address (`bind_addr`) and client address (`client_addr`) of `0.0.0.0` allow the Consul to listen over all network interfaces. The `bind_addr` setting can be limited to the host's IP address instead.

   The `bootstrap` setting should be set to `true` on one node in the cluster only. Set it to `false` on all other cluster nodes.

   For the `retry_join` option, list all the Composer host IP addresses in the cluster. At least one must be listed. If you are using cloud-hosted instances such as AWS or GCE, the `retry_join` option can be changed to something like this (assuming each cluster node is an AWS EC2 instance and has a `tag_key` called `Role` that is assigned to `zoomdata-cluster-node`):

```
.........
"retry_join": [
"provider=aws tag_key=Role tag_value=zoomdata-cluster-node"
],
.........
```

5. Restart each Consul instance and wait for several seconds for the cluster to form. Then validate the cluster by entering the following command:

   ```
   #/opt/zoomdata/bin/zoomdata-consul members
   ```

   The following shows sample output from this command:

Node	Address	Status	Type	Build	Protocol	DC	Segment
node-1	10.0.0.1:8301	alive	server	1.2.2	2	dc1	<all>
node-3	10.0.0.3:8301	alive	server	1.2.2	2	dc1	<all>
node-2	10.0.0.2:8301	alive	server	1.2.2	2	dc1	<all>

6. When the Consul cluster has formed correctly, restart all the Composer microservices for the instance. See Restart ComposerSymphony Microservices.

Step 6. Install and Configure the Configuration Microservice On Each Node

Complete the following steps.

1. **Set Up the Configuration Microservice Metadata Store or Repository**

   Before you can install and start Composer's configuration microservice, you must set up a PostgreSQL metastore or a GitHub repository to store Composer 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:

a. 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.

b. Add the following properties to the Composer`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>`.

c. 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:

a. Add the following properties to the Composer`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.

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

2. **Install, Configure, and Start the Configuration Microservice**

Install, configure, and start the Composer configuration microservice

a. 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.

b. Open the SSH client associated with your Composer instance.

c. Configure all installed and enabled Composer microservices to use the Composer 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
```

d. Each Composer 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 Composer 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 Composer 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.

e. Install, enable and start the Composer 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.

f. Restart all the other Composer microservices. Enter the following command:

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

See also Restart ComposerSymphony Microservices.

Step 7. Install and Configure the Service Monitor (Optional)

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

> **Note:** If you are installing Composer in a Windows environment, you can install the Service Monitor as part of running the initial bootstrap script. See Install Composer In A Windows Environment and Windows Bootstrap Reference.

Install, configure, and start the Service Monitor

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

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

   Use the following command to install the Composer 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 Composer 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 Composer administrator for `<Composer-admin-username>` and `<Composer-pswd>`. If the default Composer 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 Composer credentials as plain text in both the properties file and as tags in the Composer 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 Composer Service Monitor using `systemd` in a CentOS or Ubuntu environment:

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

See also Start ComposerSymphony Microservices.

Additional high availability information can be found in these topics:

- Determine How Many Nodes To Deploy
- Add Nodes To An Existing High Availability Installation
- Remove Nodes From A High Availability Environment
- Migrate Properties To The Configuration Server

Determine How Many Nodes to Deploy

To effectively implement high availability in your Composer environment, at least two nodes, each containing the following microservices, are required.

- Composer Web App microservice
- Query Engine microservice
- Data Writer microservice
- Appropriate connector microservices for your installation
- Configuration microservice

The Service Monitor and Screenshot microservices, as well as distributed tracing services, are not required on every node. Install these microservices on at least two nodes in your cluster to ensure that management of the nodes is retained should one node fail.

Finally, the Consul instance and the PostgreSQL data store used to store metadata, configuration data, and tracing data must be available to all the nodes in your Composer cluster. We recommend that you cluster the Consul instance and your PostgreSQL data store to ensure there is no single point of failure.

Add Nodes to an Existing High Availability Installation

Add Composer nodes (or instances) to an existing high availability environment

1. If not already installed, install the Composer instance as though it were a single instance (and not running in a high availability environment). See Install Composer.

2. Edit the `zoomdata.properties` and `query-engine.properties` files and ensure that the JDBC settings point to the PostgreSQL data store shared by the entire Composer cluster. Also ensure that the user name and password used to access the PostgreSQL data store are correct in these files. For information about the shared PostgreSQL data store, see Configure A High Availability Environment.

3. Edit the `consul.json` file on the instance you are adding.

   ```
   vi /etc/zoomdata/consul.json
   ```

4. Verify the `consul.json` file looks like this:

   ```
   {
   "bind_addr": "0.0.0.0",
   "bootstrap": false,
   "bootstrap_expect": 2,
   "client_addr": "0.0.0.0",
   "data_dir": "/opt/zoomdata/data/consul",
   "server": true
   }
   ```

   A bind address (`bind_addr`) and client address (`client_addr`) of `0.0.0.0` allow the Consul to listen over all network interfaces. Depending on your network setup, you may want to explicitly specify an IP address for this.

   The `bootstrap_expect` value is the total number of Composer nodes (instances) in your Composer cluster and must be the same value on every instance in the cluster.

5. Restart all the Composer microservices for the instance. See Restart ComposerSymphony Microservices.

6. Join each instance to the Consul cluster by running this command:

   ```
   /opt/zoomdata/bin/zoomdata-consul join <external-Consul-node-IP>
   ```

You can verify that the node has joined the Consul cluster by running this command:

```
/opt/zoomdata/bin/zoomdata-consul members
```

Remove Nodes from a High Availability Environment

Remove Composer nodes (or instances) from a high availability environment

1. Run the following command on the instance:

   ```
   /opt/zoomdata/bin/zoomdata-consul leave -http-addr=http://<external-Consul-node-IP>:<port>
   ```

   where `<external-Consul-node-IP>` is the IP address of the Consul used by the high availability environment.

2. If the instance you are removing will be run standalone, edit the `zoomdata.properties` and `query-engine.properties` files to point at a local PostgreSQL data store for that instance. Ensure that the JDBC settings point to the local PostgreSQL and that the user name and password used to access the local PostgreSQL data store are correct in these files. After changing the settings, restart all the Composer microservices for the instance. See Restart ComposerSymphony Microservices.

   If the instance you are removing will not be used at all, you can skip this step.

3. Edit the `consul.json` file on all the remaining Composer instances in your high availability environment.

   ```
   vi /etc/zoomdata/consul.json
   ```

4. Reduce the value specified for the `bootstrap_expect` setting by one. This value is the total number of Composer nodes (instances) in your Composer cluster and must be the same value on every instance in the cluster.

5. Restart the `zoomdata-consul` microservice on every Composer instance in the cluster to ensure they all pic up the node count change.

Migrate Properties to the Configuration Server

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. See Use The ComposerSymphony Data Discovery Configuration Microservice To Maintain Application Properties and Configure And Start The Configuration Microservice for information about the configuration microservice.

A utility is provided that can be used to migrate the microservice properties from your standalone Composer server to the Composer configuration data in the high availability PostgreSQL data store, where the configuration microservice can maintain them. The utility is located in the zoomdata tarball at `…/scripts/zoomdata/config-server-upload.jar`.

Run this utility using the following syntax:

```
java -jar config-server-upload.jar <service> <properties-file> <config-service-url>
```

where `<service>` is the microservice name, `<properties-file>` is the fully qualified file name of the properties file you want to migrate, and `<config-service-url>` is the URL of the configuration microservice (usually `http://localhost:8888/api/environment`).

The following example migrates the properties from `/etc/zoomdata/zoomdata.properties` for the Composer microservice to the configuration data in the high availability PostgreSQL data store at the URL `http://localhost:8888/api/environment`:

```
java -jar /opt/zoomdata/scripts/zoomdata/config-server-upload.jar zoomdata /etc/zoomdata/zoomdata.properties
http://localhost:8888/api/environment
```

Note: In high availability environments, you will need to run this utility for every type of microservice in the environment to ensure that you can maintain the properties for all microservices in the Service Monitor.

Uninstall Composer

To remove the Composer server and associated microservices from your network environment:

- Remove the Composer server and associated microservices
- Remove Composer-related folders and files
- Remove the PostgreSQL metadata store used by Composer

> **Note:** If you connected Composer to an existing PostgreSQL in your network, then skip these steps.

Uninstall Composer Server and All Associated Components

Select your server's operating system and follow those steps to completely remove all Composer-related components:

- In CentOS Environments
- Uninstall Composer
- In Windows Environments

In CentOS Environments

1. Stop all Composer microservices:

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

2. Remove Composer and associated components:

```
yum remove 'zoomdata*'
```

You are asked to verify the removal of the Composer components. Enter '**y** ' to confirm the removal.

3. Verify the removal of Composer components by running the following command. If all components have been successfully erased, the command returns no results.

```
yum list installed | grep zoomdata
```

4. Remove all the Composer-specific folders:

```
sudo rm -rf /etc/zoomdata
sudo rm -rf /opt/zoomdata
sudo rm -rf /etc/yum.repos.d/zoomdata*
```

The next steps are to remove the PostgreSQL metadata store and related files. However, skip these steps if you used your existing PostgreSQL as Composer's metadata store. In this case, you have completed the removal of all Composer-related components from your server.

5. Remove the PostgreSQL metadata store:

```
systemctl stop postgresql-12
sudo yum remove 'postgresql*'
```

6. Remove the following PostgreSQL-related file:

```
sudo yum remove 'pgdg*'
```

7. Remove the directories related to PostgreSQL:

```
sudo rm -rf /var/lib/pgsql
sudo rm -rf /usr/pgsql-12
```

In Ubuntu 18, 20, or 22 Environments

1. Stop all Composer microservices:

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

2. Remove Composer and associated components:

```
sudo apt-get
remove 'zoomdata*'
```

   You are asked to verify the removal of the Composer components. Enter '**y** ' to confirm the removal.

3. Verify the removal of Composer components by running the following command. If all components have been successfully erased, the command returns no results.

```
sudo apt list --installed | grep zoomdata
```

4. Remove all the Composer-specific folders:

```
sudo rm -rf /etc/zoomdata
sudo rm -rf /opt/zoomdata
sudo rm -rf /etc/apt/sources.list.d/zoomdata*
```

   The next steps remove the PostgreSQL metadata store and related files. However, skip these steps if you used your existing PostgreSQL as Composer's metadata store. In this case, you have completed the removal of all Composer-related components from your server.

5. Remove the PostgreSQL metadata store:

```
sudo systemctl stop postgresql-12
sudo apt-get remove 'postgresql*'
```

6. Remove the following PostgreSQL-related file:

```
sudo apt-get
remove 'pgdg*'
```

7. Remove the directories related to PostgreSQL:

```
sudo rm -rf /etc/apt/sources.list.d/pg*
sudo rm -rf /var/lib/postgresql/
sudo rm -rf /var/log/postgresql/
sudo rm -rf /etc/postgresql
sudo rm -rf /etc/postgresql-common/
sudo rm -rf /var/run/postgresql
```

In Windows Environments

1. Stop all Composer microservices:

```
./bootstrap-composer.ps1 -ServicesAction stop
```

2. Create a binary PostgreSQL dump of Composer metadata to the `<install-path>/data/backups/` folder:

```
./bootstrap-composer.ps1 -DumpComposerMetadata
```

Important: Copy the information to a folder outside of the installation path or it will be deleted and unrecoverable after you uninstall Composer.

3. Remove Composer components by running the following command.

```
./bootstrap-composer.ps1 -DeinstallComposer
```

See Windows Bootstrap Reference for more information.

You have completed the removal of all Composer-related components from your server.

Upgrade Composer

You can upgrade your Composer server to the current release.

For information about the difference between a clean installation of Composer and an upgrade to the latest GA release, see Clean Installation And Upgrade Differences.

Upgrade and Migration Considerations

- Windows Server 2012R2 is not compatible with both Java17 binaries and the latest releases of Composer (23.2 and later). We recommend you use Windows 2016 or later to run Composer 23.2 and later.
- In general, you can upgrade directly to the latest version of Composer from a prior version.
- If you are upgrading to a newer version of Composer and you also want to change your encryption mode, perform the upgrade first and then complete the steps described in Change The Encryption Mode.

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

Important: If you are upgrading to a newer version of Composer and have created an attribute named `User.timeZone`, this may be overwritten on upgrade. See Upgrade Workflow for more information about preparing your environment for the upgrade process.

To upgrade your Composer installation, read the following sections:

- Prerequisites To Upgrading Composer
- Upgrade Steps For Composer

If you are upgrading a distributed environment, see Upgrade A Composer Distributed Environment.

Prerequisites to Upgrading Composer

Prior to upgrading your software, we **strongly recommend** that you back up your metadata store. See Back Up The Metadata Store.

Failure to have a proper backup could result in losing data during the upgrade process. For more information, see Back Up The Metadata Store.

JDK Installation Option

An option to install OpenJDK is included in the installation and upgrade scripts provided by Composer. If you skip this option or if you install or upgrade the product manually, make sure that Java 11.0.5 is installed for Composer version 23.1 and Java 17 for Composerversion 23.2. If you do not, Composer will not start after the upgrade.

Environment Prerequisites

The installer script works in the following environments:

- CentOS Stream 9 or later
- Ubuntu 18, 20, or 22

See System Requirements for recommended settings for deploying Composer on-premise.

The target server for the Composer program should meet the following conditions:

- Server is connected to the Internet
- The user installing Composer is able to use the 'sudo' command in the server

Upgrade Steps for Composer

To begin the upgrade process, you must receive an email containing upgrade instructions from Technical Support. This email provides the upgrade script that you need to run on the server where the Composer environment resides.

Note: If you have not received upgrade instructions, open a ticket with Composer Support.

1. Make sure you have read and performed the recommendations in Prerequisites To Upgrading Composer. Be sure to back up your metadata store before you upgrade (see Back Up The Metadata Store).

2. When you receive the email, enter the upgrade command on your target server to start the automated upgrade process. The following Composer components are downloaded on your target server:

i. The Composer server

ii. Connector microservices

iii. Query Engine

iv. Data Writer microservice

3. After the upgrade script has completed, it will take a few minutes for Composer to complete its update of the metadata store. We recommend that you wait a few minutes before accessing Composer from your web browser.

   If you receive a message indicating that Composer is not yet accessible, it may not have completed its setup yet. Wait a few more minutes before trying again or opening a Support ticket. If you continue to have issues accessing Composer from your browser, open a ticket with Composer Technical Support. For information about accessing Composer, see Access ComposerSymphony.

   Note: If you notice some unusual behavior in the Composer UI after upgrading the Composer software (for example, if a drop-down menu doesn't open or the application doesn't react when you select a button), clear the browser cache and try again. If the problem persists, contact Support.

4. The upgrade script provided by Technical Support assumes that your Postgres metadata store is running locally on the same machine as the Composer code and automatically adds the new databases required by Zoomdata 4.9 (or later) and Composer 5.7 (or later) to your local installation. If you have upgraded and your metadata store is installed on a different machine (not locally), you will need to manually create the following databases in your Postgres metadata store after the upgrade.

i. `zoomdata`

ii. `zoomdata-keyset`

iii. `zoomdata-qe`

iv. `zoomdata-upload`

See Create The Metadata Store User & Stores.

5. The firewall setup you used with earlier versions of Composer should have been retained and your Composer IP address should remain unchanged, but see the following for more information:

i. Configure The Firewall

ii. Identify The Composer IP Address

6. SQL connectors require a JDBC driver to be configured before you can connect to your data source. You can download the driver from the vendor's site. Be aware that you need to download and configure JDBC drivers for the following Composer connectors:

i. Microsoft SQL Server

ii. MemSQL

iii. MySQL

iv. Oracle

v. Amazon Redshift

vi. Teradata

If you are using one of these connectors, you need to download and configure a JDBC driver as soon as your Composer server has finished upgrading. For steps, see Add A JDBC Driver.

7. Reimport any CA certificates to the Java 11.0.5 for Composer 23.1 and Java 17 for Composer 23.2 or later you installed prior to the Composer upgrade. (In Prerequisites To Upgrading Composer, we recommended that you store them or back them up before the Composer upgrade.)

Upgrade a Composer Distributed Environment

To upgrade a Composer distributed environment from a previous version of Composer or from a Composer environment that was not distributed, follow these steps.

Important: We recommend that you set up a test environment to run through the upgrade process prior to upgrading your production environment. This will ensure that any errors are understood and resolved that could potentially impact your customers running the production environment.

Note: If you are upgrading to Composer 23.2 to 24.2 from an earlier Composer version, you will also need to upgrade to Java JDK 17 and PostgreSQL 12. When upgrading to these releases, we recommend that you use the bootstrap installation script provided by insightsoftware to upgrade to avoid manual upgrades to Java JDK 17 and PostgreSQL 12. If you use the bootstrap script, these upgrades happen automatically.

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

Upgrade a Composer distributed environment, complete the following steps

Step 1: Disable and Stop All Composer Microservices

Before you can start the upgrade, you must disable and stop all Composer microservices. Run the commands described in this section.

To disable all Composer microservices, enter the following command:

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

To stop all Composer microservices, enter the following command:

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

Step 2: Back Up the Postgres Metadata Store

Before upgrading your Composer server, Composer 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.

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 ComposerSymphony 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 ComposerSymphony Microservices.

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

Step 3: Create A New Postgres Metadata Store

While you can upgrade in place, using an existing metadata store, we recommend that you set up a new instance of the PostgreSQL metadata store for your upgrades. This will preserve the old metadata store in case you need to roll back. If you prefer to upgrade using your existing metadata store, contact your Technical Support representative.

The metadata store should be centrally installed, accessible by all Composer nodes. In a high availability environment, it can be clustered. Complete the following steps.

A. ### Complete PostgreSQL Setup Steps

The instructions to set up PostgreSQL as Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

- 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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

- 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***

If you installed Composer'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 Composer (v24.3 and later) use PostgreSQL 16. If you are upgrading your environment to v24.3 or later, you can retain your existing PostgreSQL version.

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. ***Create the Metadata Store User***

A Composer user must be established for the Postgres metadata store.

To create the Composer user for the Postgres metadata store, complete the following steps:

- For all Linux operating systems, create the Composer user in PostgreSQL. Run the following command:

```
sudo -u postgres -H psql -c "CREATE USER <db_username> WITH PASSWORD '<db_password>'"
```

  Substitute the PostgreSQL user name and password for `<db_username>` and `<db_password>`.

- Create the stores that will hold the Composer metadata, upload data, keyset, and query engine data. Run the following series of commands, substituting the user name for `<db_username>`:

```
sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata\" WITH OWNER <db_username>"
sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata-upload\" WITH OWNER <db_username>"
sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata-keyset\" WITH OWNER <db_username>"
sudo -u postgres -H psql -c "CREATE DATABASE\"zoomdata-qe\" WITH OWNER <db_username>"
```

D. ### *Configure the Metadata Store for SSL*

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
```

E. ### *Configure the Metadata Store for a Distributed Environment*

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

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

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 4: Restore the Metadata From the Metadata Store Backup

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 Composer server.
2. Stop all Composer microservices. For appropriate commands based on your OS, see Stop ComposerSymphony 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 Composer microservices. For appropriate commands based on your OS, see Restart ComposerSymphony Microservices.

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

Step 5. Upgrade Your Composer Software On All Servers

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

To deploy multiple Composer 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 Composer PostgreSQl metadata store

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

2. In high availability (HA) environments only, run the following commands for each instance to set up the environment variables for the Consul configuration:

```
export ZOOMDATA_CONSUL_ADDRESS=<instance_ip>:8500
export ZOOMDATA_CONSUL_CLIENT=0.0.0.0
```

where `<instance_ip>` is the IP address of the Composer instance.

3. 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*.

4. Disable the automatic PostgreSQL metadata store upgrade by running the following command:

```
export ZOOMDATA_POSTGRES_DISABLE_UPGRADE=TRUE'
```

5. 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 Composer version and release.

6. 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>
```

7. 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
```

8. Repeat these steps for every Composer instance (node) in your Composer cluster. For additional information on how many nodes to deploy in a high availability environment, see Determine How Many Nodes To Deploy.

Step 6. Set Up a Load Balancer (Optional)

If you are upgrading from an existing distributed environment to another distributed environment, you can skip this step. Your load balancers were set up when you first installed the product.

If you are upgrading from a non-distributed environment to a distributed environment, 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 Composer 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 Composer 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 7. Configure Consul Services for High Availability (Optional)

If you are upgrading from an existing high availability environment to another high availability environment, you can skip this step. Consul services were set up when you first installed the product.

If you are upgrading from a non-distributed environment to a high availability environment, Consul services must be configured.

You can install individual Consul instances locally for each Composer node and then configure the Consul instances as a cluster for the Composer high availability environment. Before you do, make sure you are familiar with the general clustering techniques used for Consul. See https://www.consul.io/docs/install/bootstrapping for more information.

The primary advantage of using a Consul cluster in a Composer high availability environment is that it requires fewer configuration changes to Composer components. Each component will look for the Consul on the localhost interface at port 127.0.0.1. The only configuration necessary is to the Consul cluster nodes themselves.

Another advantage is that each node of the Composer high availability cluster will use several discovered instances of the same component to balance the load and become more tolerant of any component failures.

Configure a Consul cluster for a Composer high availability environment

1. Install all the individual Consul instances on each Composer node. This happens automatically when you use the Composer bootstrap installation procedure.
2. Make sure that a firewall is opened in your environment for ports 8500, 8300, 8301, and 8302 on all hosts that will form the Consul cluster.
3. Edit the Consul custom configuration file `consul.json` on each Composer node.

   ```
   vi /etc/zoomdata/consul.json
   ```

   If you did not install the Consul instances using the Composer bootstrap installation procedure, its custom configuration file might have a different name and location.
4. Configure the Consul custom configuration file for each Consul instance so it includes these lines:

   ```
   {
   .........
   "node_name": "<node-name>"
   "bind_addr": "0.0.0.0",
   "bootstrap": false,
   "client_addr": "0.0.0.0",
   "retry_join": [
   "<host-ip-address-1>",
   "<host-ip-address-2>",
   "<host-ip-address-n>"
   ],
   "server": true
   .........
   }
   ```

   The `<node-name>` settings for each Consul node should be unique within the cluster. Each Consul instance in the cluster should have a different name.

   A bind address (`bind_addr`) and client address (`client_addr`) of `0.0.0.0` allow the Consul to listen over all network interfaces. The `bind_addr` setting can be limited to the host's IP address instead.

   The `bootstrap` setting should be set to `true` on one node in the cluster only. Set it to `false` on all other cluster nodes.

For the `retry_join` option, list all the Composer host IP addresses in the cluster. At least one must be listed. If you are using cloud-hosted instances such as AWS or GCE, the `retry_join` option can be changed to something like this (assuming each cluster node is an AWS EC2 instance and has a `tag_key` called `Role` that is assigned to `zoomdata-cluster-node`):

```
.........
"retry_join": [
"provider=aws tag_key=Role tag_value=zoomdata-cluster-node"
],
.........
```

5. Restart each Consul instance and wait for several seconds for the cluster to form. Then validate the cluster by entering the following command:

```
#/opt/zoomdata/bin/zoomdata-consul members
```

The following shows sample output from this command:

Node	Address	Status	Type	Build	Protocol	DC	Segment
node-1	10.0.0.1:8301	alive	server	1.2.2	2	dc1	<all>
node-3	10.0.0.3:8301	alive	server	1.2.2	2	dc1	<all>
node-2	10.0.0.2:8301	alive	server	1.2.2	2	dc1	<all>

6. When the Consul cluster has formed correctly, restart all the Composer microservices for the instance. See Restart ComposerSymphony Microservices.

Step 8: Vacuum the Metadata Store

insightsoftware highly recommends that you vacuum the Composer PostgreSQL metadata store after every upgrade. Vacuuming the metadata store optimizes it by maximizing database performance and minimizing the disk space it uses. The following simple procedure recollects metadata store statistics and "vacuums" the unused or dead rows in the database.

Note: This procedure blocks writing to the database. Consequently, work with the database is impossible until the procedure completes.

This procedure only works for a PostgreSQL database.

insightsoftware tested this procedure after an upgrade to Composer 5.9. on a machine with 2.6 GHz 6-Core Intel Core i7, 16 GB 2667 MHz DDR4, and SSD storage. The tested database contained 300 accounts, 4800 users, 60,000 sources, 9000 dashboards with 30 visuals each. The procedure took 5-10 minutes.

If you have performance problems with your PostgreSQL metadata store, examine the database settings related to automatic vacuuming, automatic analyzing, and the write-ahead log (WAL). See Optimize Composer's Metadata Store Performance.

Vacuum your PostgreSQL metadata store

1. Upgrade your version of Composer (if you have not already done so). This automatically upgrades the PostgreSQL metadata store.
2. Stop Composer and any process connecting to its PostgreSQL metadata store. See Stop ComposerSymphony Microservices.
3. Back up the PostgreSQL metadata store. See Back Up The Metadata Store.
4. Connect to the PostgreSQL database.
5. Run the following command in the console for the PostgreSQL database:

   ```
   VACUUM (FULL, ANALYZE);
   ```

   For more information about VACUUM, see *https://www.postgresql.org/docs/12/sql-vacuum.html*.
6. After vacuuming completes, start Composer. See Start ComposerSymphony Microservices.

Step 9: Enable and Start All Composer Microservices

After the upgrade, you must enable and start all Composer microservices. Run the commands described in this section.

To enable all Composer microservices, enter the following command:

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

To start all Composer microservices, enter the following command:

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

Back Up the Metadata Store

Before upgrading your Composer 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 ComposerSymphony 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 ComposerSymphony 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 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 Composer server.
2. Stop all Composer microservices. For appropriate commands based on your OS, see Stop ComposerSymphony 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 Composer microservices. For appropriate commands based on your OS, see Restart ComposerSymphony Microservices.

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