SYSPRO Embedded Analytics Disaster Recovery Guidelines

SYSPRO 8

Published: February 2023

SYSPRO™

CONTENTS

SYSPRO Embedded Analytics: DR Guidelines

SYSPRO Embedded Analytics: DR Guidelines

This article provides Disaster Recovery (DR) guidelines in relation to the **SYSPRO Embedded Analytics** feature.

Best practice advise is provided in the following areas:

- Backup and restoration of the PostgreSQL database in the event of a disaster
- Troubleshooting related to PostgreSQL
- Frequently asked questions

Please note that every DR plan is unique as it takes into account custom business needs, recovery objectives, and the system vulnerabilities of a specific organization. Therefore, the information contained within this article is not meant to be a one-size-fits-all solution. Instead, it serves as a starting point that should be adapted based on the unique needs and requirements of your organization's environment.

Overview

Disaster recovery is a vital aspect of any business continuity plan, ensuring that the essential operations of an organization can continue even in the event of a major disruption such as a natural disaster, cyber attack, or system failure. A well-constructed disaster recovery plan has the potential to reduce downtime, minimize the financial impact of an outage and provide a sense of security.

However, it's crucial to understand that every organization is unique, and the specific disaster recovery needs of one company can differ greatly from another. This is why it's imperative to thoroughly evaluate and verify any disaster recovery advice before implementing it in your own environment.

The need for a reliable and practical disaster recovery plan is crucial for any business, as without such a plan, there is limited protection from the impact of significantly disruptive events. This guide provides suggested guidelines and information to help you develop a disaster recovery plan that is suitable for your organization.

PostgreSQL Backup Process

This section provides a guideline for backing up your PostgreSQL databases:

1 STOP ALL ZOOMDATA SERVICES

- It is very important for the backup process that *all* of the **zoomdata** services are stopped. Otherwise, when attempting to perform a backup, the process will fail to create the required `.dump` file.
- It is imperative that you stop the **zoomdata_consul** service first due to the other **zoomdata** services that have a dependency on it. Therefore, we recommend stopping the services in the following order:
  - zoomdata_consul
  - zoomdata_edc_* (i.e. data sources or connectors)
  - zoomdata_query_engine
  - zoomdata_data_writer_postgresql
  - zoomdata_screenshot_service
  - zoomdata

2 BACKUP ZOOMDATA DATABASES

The following provides sample scripts using a command called **pg_dump** to execute the backups to a file type named `.dump`.

> See the following link for more information regarding **pg_dump** and related commands:
>
> `https://www.postgresql.org/docs/current/app-pgdump.html`

You can either save these scripts to file or copy and paste them into **Windows PowerShell**, and then update as required before running them.

> Ensure to validate and update these scripts accordingly for your environment before using them.

There are various methods that can be used to connect and authenticate to the **pg_dump** command:

2 Password Prompt

In this scenario, the **PostgreSQL** user password will be prompted every time the **pg_dump** command is executed (i.e. for every database backup):

BackupZoomdata_Prompt.ps1

```
#-------------------------------------------------------------------------------#
#--- Region: Steps to Backup PostgreSQL databases
#-------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Backup Zoomdata databases prompting password for each backup
# 3. Start all Zoomdata services
#endregion

#-------------------------------------------------------------------------------#
#--- Region: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#-------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is running and stop this - This will stop all the other required services
Write-Host "Stopping all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  | stop-service -
ErrorAction Stop -Force -Verbose

# Check if any other zoomdata services are still running and stop them:
# Only run this if the zoomdata_consul service is successfully stopped:
if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} | stop-service
-ErrorAction Stop -Verbose
    }
#endregion

#-------------------------------------------------------------------------------#
#--- Region: BackupZoomdataDatabases
#--- Description: This will backup the specified zoomdata databases
#--- IMPORTANT: Each backup will prompt for the postgresql password
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/current/app-pgdump.html
#-------------------------------------------------------------------------------#
#region
# Define a List of Zoomdata Databases to Backup:
$DatabasesToBackup = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define Backup folder location:
$BackupFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"

# Test if Backup folder exist:
if (-not (Test-Path $BackupFolder))
    {
        Write-Host "Creating the Backup folder: $BackupFolder"
        mkdir $BackupFolder
    }

# Navigate to the backup folder location:
cd $BackupFolder
Write-Host "Backups will be created under the following folder: $BackupFolder"

# Loop through list of databases and perform backup:
Foreach($DatabaseToBackup in $DatabasesToBackup)
    {
        # Filename:
        $BackupFileName = $DatabaseToBackup + ".dump"
```

2

```
        Write-Host "Creating the following backup file: $BackupFileName of the following database:
$DatabaseToBackup"

        Try
            {
                pg_dump -Fc -v -W --host $PostgreSQLHostName --port $PostgreSQLPort --username
$PostgreSQLUser --dbname $DatabaseToBackup -f $BackupFileName
            }
        Catch
            {
                $ErrorMessage = $_.Exception.Message
                Write-Host "The following Error occured: $ErrorMessage creating a backup of the following
database: $DatabaseToBackup"
            }
    }
#endregion

#--------------------------------------------------------------------------------#
#--- Region: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#--------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is stopped and start this:
Write-Host "Starting all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  | start-service -
ErrorAction Stop -Verbose

# Check if any other zoomdata services are stopped and start them:
# Only run this if the zoomdata_consul service is successfully running:
if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} | start-service
-ErrorAction Stop -Verbose
    }
#endregion
```

2 No Password Prompt

In the following scenarios, the **PostgreSQL** user password is *not* prompted every time the **pg_dump** command is executed (i.e. for every database backup).

Set password environment variable to authenticate to PostgreSQL

This sample script is for setting the PGPASSWORD environment variable to authenticate to **PostgreSQL** for the backup:

BackupZoomdata_NoPrompt_ENV.ps1

```
#-------------------------------------------------------------------------------#
#--- Region: Steps to Backup PostgreSQL databases
#-------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Backup Zoomdata databases without prompting password - using Env:PGPASSWORD
# 3. Start all Zoomdata services
#endregion

#-------------------------------------------------------------------------------#
#--- Region: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#-------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is running and stop this - This will stop all the other required services
Write-Host "Stopping all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  | stop-service -
ErrorAction Stop -Force -Verbose

# Check if any other zoomdata services are still running and stop them:
# Only run this if the zoomdata_consul service is successfully stopped:
if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} | stop-service
-ErrorAction Stop -Verbose
    }
#endregion

#-------------------------------------------------------------------------------#
#--- Function: CheckIfDatabaseExist
#--- Description: This will check if the zoomdata database exist
#--- Output: True - The database exist
#--- Output: False - The database does not exist
#-------------------------------------------------------------------------------#
#region
Function CheckIfDatabaseExist([string]$PostgreSQLHostName, [string]$PostgreSQLUser,[string]$PostgreSQLPort,
[string]$DatabaseToBackup)
{
    Try
        {
            # Check if database exist:
            $Query = "SELECT 1 FROM pg_database WHERE datname = '$DatabaseToBackup';"
            $DatabaseExistCheck = psql --host $PostgreSQLHostName --port $PostgreSQLPort --username
$PostgreSQLUser -c $Query -t

            if ($DatabaseExistCheck.Length -ge 1 -and $DatabaseExistCheck[0].Contains("1"))
                {
                    return $True
                }
            Write-Host "The zoomdata database: $DatabaseToBackup does not exist."
            return $False
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the CheckIfDatabaseExist function."
            return $False
```

2

```
        }
}
#endregion

#-------------------------------------------------------------------------------#
#--- Region: BackupZoomdataDatabases
#--- Description: This will backup the specified zoomdata databases
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/current/app-pgdump.html
#-------------------------------------------------------------------------------#
#region
# Define a List of Zoomdata Databases to Backup:
$DatabasesToBackup = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define Backup folder location:
$BackupFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"
$env:PGPASSWORD = 'ComposerDBPassword'; # This will be the password for the PostgreSQLUser user
# More reading on passing env:PGPASSWORD: http://www.postgresql.org/docs/current/static/libpq-envars.html

# Test if Backup folder exist:
if (-not (Test-Path $BackupFolder))
    {
        Write-Host "Creating the Backup folder: $BackupFolder"
        mkdir $BackupFolder
    }

# Navigate to the backup folder location:
cd $BackupFolder
Write-Host "Backups will be created under the following folder: $BackupFolder"

# Loop through list of databases and perform backup:
Foreach($DatabaseToBackup in $DatabasesToBackup)
    {
        # Filename:
        $BackupFileName = $DatabaseToBackup + ".dump"

        Try
            {
                # Check if database exist:
                [bool]$DbExist = CheckIfDatabaseExist
$PostgreSQLHostName $PostgreSQLUser $PostgreSQLPort $DatabaseToBackup

                if ($DbExist -eq $true)
                    {
                        Write-Host "Creating the following backup file: $BackupFileName of the following
database: $DatabaseToBackup"
                        pg_dump -Fc -v --host $PostgreSQLHostName --port $PostgreSQLPort --username
$PostgreSQLUser --dbname $DatabaseToBackup -f $BackupFileName
                    }
            }
        Catch
            {
                $ErrorMessage = $_.Exception.Message
                Write-Host "The following Error occured: $ErrorMessage creating a backup of the following
database: $DatabaseToBackup"
            }
    }
#endregion

#-------------------------------------------------------------------------------#
#--- Region: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#-------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is stopped and start this:
Write-Host "Starting all zoomdata services."
```

2

```
Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  | start-service -
ErrorAction Stop -Verbose

# Check if any other zoomdata services are stopped and start them:
# Only run this if the zoomdata_consul service is successfully running:
if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} | start-service
-ErrorAction Stop -Verbose
    }
#endregion
```

See the following link for more information regarding Environment Variables:

https://www.postgresql.org/docs/current/libpq-envars.html

2 Use a password config file to authenticate to PostgreSQL

This sample script is for using the `pgpass.conf` file to authenticate to **PostgreSQL** for the backup:

BackupZoomdata_NoPrompt_PGPASS_File.ps1

```
#------------------------------------------------------------------------------#
#--- Region: Steps to Backup PostgreSQL databases
#------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Backup Zoomdata databases without prompting password  - using pgpass.conf file
# 3. Start all Zoomdata services
#endregion

#------------------------------------------------------------------------------#
#--- Region: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is running and stop this - This will stop all the other required services
Write-Host "Stopping all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  | stop-service -
ErrorAction Stop -Force -Verbose

# Check if any other zoomdata services are still running and stop them:
# Only run this if the zoomdata_consul service is successfully stopped:
if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} | stop-service
-ErrorAction Stop -Verbose
    }
#endregion

#------------------------------------------------------------------------------#
#--- Function: CheckIfDatabaseExist
#--- Description: This will check if the zoomdata database exist
#--- Output: True - The database exist
#--- Output: False - The database does not exist
#------------------------------------------------------------------------------#
#region
Function CheckIfDatabaseExist([string]$PostgreSQLHostName, [string]$PostgreSQLUser,[string]$PostgreSQLPort,
[string]$DatabaseToBackup)
{
    Try
        {
            # Check if database exist:
            $Query = "SELECT 1 FROM pg_database WHERE datname = '$DatabaseToBackup';"
            $DatabaseExistCheck = psql --host $PostgreSQLHostName --port $PostgreSQLPort --username
$PostgreSQLUser -c $Query -t

            if ($DatabaseExistCheck.Length -ge 1 -and $DatabaseExistCheck[0].Contains("1"))
                {
                    return $true
                }
            Write-Host "The zoomdata database: $DatabaseToBackup does not exist."
            return $false
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the CheckIfDatabaseExist function."
            return $false
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Region: BackupZoomdataDatabases
#--- Description: This will backup the specified zoomdata databases
```

2

```
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/current/app-pgdump.html
#-----------------------------------------------------------------------------#
#region
# Define a List of Zoomdata Databases to Backup:
$DatabasesToBackup = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define Backup folder location:
$BackupFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"
# Please note: The pgpass file %APPDATA%\postgresql\pgpass.conf file needs to exist according to the link
below in order for the connection to work:
# http://www.postgresql.org/docs/current/static/libpq-pgpass.html

# Test if Backup folder exist:
if (-not (Test-Path $BackupFolder))
    {
        Write-Host "Creating the Backup folder: $BackupFolder"
        mkdir $BackupFolder
    }

# Navigate to the backup folder location:
cd $BackupFolder
Write-Host "Backups will be created under the following folder: $BackupFolder"

# Loop through list of databases and perform backup:
Foreach($DatabaseToBackup in $DatabasesToBackup)
    {
        # Filename:
        $BackupFileName = $DatabaseToBackup + ".dump"

        Try
            {
                # Check if database exist:
                [bool]$DbExist = CheckIfDatabaseExist
$PostgreSQLHostName $PostgreSQLUser $PostgreSQLPort $DatabaseToBackup

                if ($DbExist -eq $true)
                    {
                        Write-Host "Creating the following backup file: $BackupFileName of the following
database: $DatabaseToBackup"
                        pg_dump -Fc -v --host $PostgreSQLHostName --port $PostgreSQLPort --username
$PostgreSQLUser --dbname $DatabaseToBackup -f $BackupFileName
                    }
            }
        Catch
            {
                $ErrorMessage = $_.Exception.Message
                Write-Host "The following Error occured: $ErrorMessage creating a backup of the following
database: $DatabaseToBackup"
            }
    }
#endregion

#-----------------------------------------------------------------------------#
#--- Region: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#-----------------------------------------------------------------------------#
#region
# Check zoomdata_consul is stopped and start this:
Write-Host "Starting all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  | start-service -
ErrorAction Stop -Verbose

# Check if any other zoomdata services are stopped and start them:
# Only run this if the zoomdata_consul service is successfully running:
if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
    {
```

2

```
        Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} | start-service
-ErrorAction Stop -Verbose
    }
#endregion
```

See the following link for more information regarding Environment Variables:

https://www.postgresql.org/docs/current/libpq-pgpass.html

2 Use a connection URI to authenticate to PostgreSQL

BackupZoomdata_NoPrompt_URI.ps1

```
#------------------------------------------------------------------------------#
#--- Region: Steps to Backup PostgreSQL databases
#------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Backup Zoomdata databases without prompting password - using connection URI
# 3. Start all Zoomdata services
#endregion

#------------------------------------------------------------------------------#
#--- Region: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is running and stop this - This will stop all the other required services
Write-Host "Stopping all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  | stop-service -
ErrorAction Stop -Force -Verbose

# Check if any other zoomdata services are still running and stop them:
# Only run this if the zoomdata_consul service is successfully stopped:
if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} | stop-service
-ErrorAction Stop -Verbose
    }
#endregion

#------------------------------------------------------------------------------#
#--- Function: CheckIfDatabaseExist
#--- Description: This will query to check if the database exist
#------------------------------------------------------------------------------#
#region
Function CheckIfDatabaseExist([string]$PostgreSQLHostName, [string]$PostgreSQLUser,[string]$PostgreSQLPort,
[string]$DatabaseToBackup)
{
    Try
        {
            # Check if database exist:
            $Query = "SELECT 1 FROM pg_database WHERE datname = '$DatabaseToBackup';"
            $ArgumentLine =
"$($PostgreSQLUser):$($PostgreSQLPassword)@$($PostgreSQLHostName):$($PostgreSQLPort)/postgres"
            $DatabaseExistCheck = psql --dbname=postgresql://$ArgumentLine -c $Query -t

            if ($DatabaseExistCheck.Length -ge 1 -and $DatabaseExistCheck[0].Contains("1"))
                {
                    return $true
                }
            Write-Host "The zoomdata database: $DatabaseToBackup does not exist."
            return $false
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the CheckIfDatabaseExist function."
            return $false
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Region: BackupZoomdataDatabases
#--- Description: This will backup the specified zoomdata databases
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/current/app-pgdump.html
#------------------------------------------------------------------------------#
#region
# Define a List of Zoomdata Databases to Backup:
```

2

```
$DatabasesToBackup = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-auditing')

# Define Backup folder location:
$BackupFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"
$PostgreSQLPassword = "ComposerDBPassword" # This will be the password for the PostgreSQLUser user

# Test if Backup folder exist:
if (-not (Test-Path $BackupFolder))
    {
        Write-Host "Creating the Backup folder: $BackupFolder"
        mkdir $BackupFolder
    }

# Navigate to the backup folder location:
cd $BackupFolder
Write-Host "Backups will be created under the following folder: $BackupFolder"

# Loop through list of databases and perform backup:
Foreach($DatabaseToBackup in $DatabasesToBackup)
    {
        # Argument Line to pass to pg_dump:
        # More reading on using URI: http://www.postgresql.org/docs/current/static/libpq-connect.html#AEN42532
        $ArgumentLine = "$($PostgreSQLUser):$($PostgreSQLPassword)@$($PostgreSQLHostName):$($PostgreSQLPort)/$($DatabaseToBackup)"

        # Filename: .dump:
        $BackupFileName = $DatabaseToBackup + ".dump"

        Try
            {
                # Check if database exist:
                [bool]$DbExist = CheckIfDatabaseExist $PostgreSQLHostName $PostgreSQLUser $PostgreSQLPort $DatabaseToBackup

                if ($DbExist -eq $true)
                    {
                        Write-Host "Creating the following backup file: $BackupFileName of the following database: $DatabaseToBackup"
                        pg_dump -Fc -v --dbname=postgresql://$ArgumentLine -f $BackupFileName
                    }
            }
        Catch
            {
                $ErrorMessage = $_.Exception.Message
                Write-Host "The following Error occured: $ErrorMessage creating a backup of the following database: $DatabaseToBackup"
            }
    }
#endregion

#-------------------------------------------------------------------------------#
#--- Region: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#-------------------------------------------------------------------------------#
#region
# Check zoomdata_consul is stopped and start this:
Write-Host "Starting all zoomdata services."
Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  | start-service -ErrorAction Stop -Verbose

# Check if any other zoomdata services are stopped and start them:
# Only run this if the zoomdata_consul service is successfully running:
if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
    {
        Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} | start-service -ErrorAction Stop -Verbose
    }
#endregion
```

2

See the following link for more information regarding Environment Variables:
https://www.postgresql.org/docs/current/libpq-connect.html#AEN42532

3 **REVIEW YOUR BACKUP FILES**

Ensure that your backup files have processed correctly.

Based on the sample scripts and default databases shipped with **SYSPRO Embedded Analytics**, you should have the following backup files within the `SEA` folder of your `Temp` drive:

- zoomdata.dump
- zoomdata-keyset.dump
- zoomdata-qe.dump
- zoomdata-upload.dump
- zoomdata-user-auditing.dump

4 **START ALL ZOOMDATA SERVICES**

Ensure that you always start the **zoomdata_consul** service first, due to the other services' dependency on it.

Therefore, we recommend that you start the services in the following order:

a. zoomdata_consul
b. zoomdata_edc_* (i.e. data sources or connectors)
c. zoomdata_query_engine
d. zoomdata_data_writer_postgresql
e. zoomdata_screenshot_service
f. zoomdata

PostgreSQL Restore Process

This section provides a guideline for restoring your PostgreSQL databases from the backup copy:

1 STOP ALL ZOOMDATA SERVICES

- It is very important for the restore process that *all* of the **zoomdata** services are stopped. Otherwise, when attempting to perform a restoration, the process will fail to create the required **zoomdata** database.
- It is imperative that you stop the **zoomdata_consul** service first due to the other **zoomdata** services that have a dependency on it. Therefore, we recommend stopping the services in the following order:
  - zoomdata_consul
  - zoomdata_edc_* (i.e. data sources or connectors)
  - zoomdata_query_engine
  - zoomdata_data_writer_postgresql
  - zoomdata_screenshot_service
  - zoomdata

2 RESTORE ZOOMDATA DATABASES

The following provides sample scripts using a command called **pg_restore** to execute restore commands from the `.dump` files.

> See the following link for more information regarding **pg_restore** and related commands:
>
> `https://www.postgresql.org/docs/current/app-pgrestore.html`

You can either save these scripts to file or copy and paste them into **Windows PowerShell**, and then update as required before running them.

> Ensure to validate and update these scripts accordingly for your environment before using them.

There are various methods that can be used to connect and authenticate to the **pg_restore** command:

2 Password Prompt

In this scenario, the **PostgreSQL** user password will be prompted every time the **pg_restore** command is executed (i.e. for every database restore):

BackupZoomdata_Prompt.ps1

```
#------------------------------------------------------------------------------#
#--- Region: Steps to Restore PostgreSQL databases
#------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Restore databases prompting password for each backup
# 3. Start all Zoomdata services
#endregion

#------------------------------------------------------------------------------#
#--- Function: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#------------------------------------------------------------------------------#
#region
Function StopAllZoomServices()
{
    Try
        {
            # Check zoomdata_consul is running and stop this - This will stop all the other required
services
            Write-Host "Running the StopAllZoomServices function and stopping all the zoomdata services."
            Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  |
stop-service -ErrorAction Stop -Force -Verbose

            # Check if any other zoomdata services are still running and stop them:
            # Only run this if the zoomdata_consul service is successfully stopped:
            if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} |
stop-service -ErrorAction Stop -Verbose
                }
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StopAllZoomServices function."
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Function: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#------------------------------------------------------------------------------#
#region
Function StartAllZoomdataServices()
{
    Try
        {
            # Check zoomdata_consul is stopped and start this:
            Write-Host "Running the StartAllZoomdataServices function and starting all the zoomdata
services."
            Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  |
start-service -ErrorAction Stop -Verbose

            # Check if any other zoomdata services are stopped and start them:
            # Only run this if the zoomdata_consul service is successfully running:
            if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} |
start-service -ErrorAction Stop -Verbose
                }
        }
    Catch
```

2

```
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StartAllZoomdataServices
function."
        }
}
#endregion

#-------------------------------------------------------------------------#
#--- Region: RestoreZoomdataDatabases
#--- Description: This will restore the specified zoomdata databases
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/15/app-pgrestore.html
#-------------------------------------------------------------------------#
#region
# Define a list of Zoomdata databases to restore:
$DatabasesToRestore = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define restore folder location - Where all the .dump files reside:
$RestoreFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"

# Test if restore folder exist:
if (-not(Test-Path $RestoreFolder))
    {
        Write-Host "The restore folder: $RestoreFolder does not exit. Process cannot continue."
        return
    }

# Stop all the services:
StopAllZoomServices

# Navigate to the restore folder location:
cd $RestoreFolder

# Loop through list of databases and perform a restore:
Foreach($DatabaseToRestore in $DatabasesToRestore)
    {
        # Filename: .dump:
        $RestoreFileName = $DatabaseToRestore + ".dump"

        # Test if the backup file exist - For example: zoomdata.dump:
        $FullPathToRestoreFile = "$($RestoreFolder)\$($RestoreFileName)"

        if (-not(Test-Path $FullPathToRestoreFile))
            {
                Write-Host "The restore file: $FullPathToRestoreFile does not exist. The database:
$DatabaseToBackup cannot be restored."
            }
        else
            {
                Write-Host "Restoring the following database: $DatabaseToRestore from the following file:
$FullPathToRestoreFile"

                Try
                    {
                        pg_restore --create --host=$PostgreSQLHostName --port $PostgreSQLPort --
username=$PostgreSQLUser --dbname postgres $FullPathToRestoreFile
                    }
                Catch
                    {
                        $ErrorMessage = $_.Exception.Message
                        Write-Host "The following Error occured: $ErrorMessage creating a backup of the
following database: $DatabaseToBackup"
                    }
            }
    }

# Start all the services:
StartAllZoomdataServices
#endregion
```

2 No Password Prompt

In the following scenarios, the **PostgreSQL** user password is *not* prompted every time the **pg_restore** command is executed (i.e. for every database restore).

Set password environment variable to authenticate to PostgreSQL

This sample script is for setting the PGPASSWORD environment variable to authenticate to **PostgreSQL** for the restore:

RestoreZoomdata_NoPrompt_ENV.ps1

```
#------------------------------------------------------------------------------#
#--- Region: Steps to Restore PostgreSQL databases
#------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Restore Zoomdata databases without prompting password - using Env:PGPASSWORD
# 3. Start all Zoomdata services
#endregion

#------------------------------------------------------------------------------#
#--- Function: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#------------------------------------------------------------------------------#
#region
Function StopAllZoomServices()
{
    Try
        {
            # Check zoomdata_consul is running and stop this - This will stop all the other required
services
            Write-Host "Running the StopAllZoomServices function and stopping all the zoomdata services."
            Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  |
stop-service -ErrorAction Stop -Force -Verbose

            # Check if any other zoomdata services are still running and stop them:
            # Only run this if the zoomdata_consul service is successfully stopped:
            if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} |
stop-service -ErrorAction Stop -Verbose
                }
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StopAllZoomServices function."
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Function: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#------------------------------------------------------------------------------#
#region
Function StartAllZoomdataServices()
{
    Try
        {
            # Check zoomdata_consul is stopped and start this:
            Write-Host "Running the StartAllZoomdataServices function and starting all the zoomdata
services."
            Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  |
start-service -ErrorAction Stop -Verbose

            # Check if any other zoomdata services are stopped and start them:
```

2

```
            # Only run this if the zoomdata_consul service is successfully running:
            if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} |
start-service -ErrorAction Stop -Verbose
                }
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StartAllZoomdataServices
function."
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Region: RestoreZoomdataDatabases
#--- Description: This will restore the specified zoomdata databases
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/15/app-pgrestore.html
#------------------------------------------------------------------------------#
#region
# Define a list of Zoomdata databases to restore:
$DatabasesToRestore = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define restore folder location - Where all the .dump files reside:
$RestoreFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"
$env:PGPASSWORD = 'ComposerDBPassword'; # This will be the password for the PostgreSQLUser user
# More reading on passing env:PGPASSWORD: http://www.postgresql.org/docs/current/static/libpq-envars.html

# Test if restore folder exist:
if (-not(Test-Path $RestoreFolder))
    {
        Write-Host "The restore folder: $RestoreFolder does not exit. Process cannot continue."
        return
    }

# Stop all the services:
StopAllZoomServices

# Navigate to the restore folder location:
cd $RestoreFolder

# Loop through list of databases and perform a restore:
Foreach($DatabaseToRestore in $DatabasesToRestore)
    {
        # Filename: .dump:
        $RestoreFileName = $DatabaseToRestore + ".dump"

        # Test if the backup file exist - For example: zoomdata.dump:
        $FullPathToRestoreFile = "$($RestoreFolder)\$($RestoreFileName)"

        if (-not(Test-Path $FullPathToRestoreFile))
            {
                Write-Host "The restore file: $FullPathToRestoreFile does not exist. The database:
$DatabaseToBackup cannot be restored."
            }
        else
            {
                Write-Host "Restoring the following database: $DatabaseToRestore from the following file:
$FullPathToRestoreFile"

                Try
                    {
                        pg_restore --create --host=$PostgreSQLHostName --port $PostgreSQLPort --
username=$PostgreSQLUser --dbname postgres $FullPathToRestoreFile
                    }
```

2

```
                Catch
                    {
                        $ErrorMessage = $_.Exception.Message
                        Write-Host "The following Error occured: $ErrorMessage creating a backup of the
following database: $DatabaseToBackup"
                    }
            }
    }

# Start all the services:
StartAllZoomdataServices
#endregion
```

2 Use a password config file to authenticate to PostgreSQL

This sample script is for using the `pgpass.conf` file to authenticate to **PostgreSQL** for the restore:

RestoreZoomdata_NoPrompt_PGPASS_File.ps1

```
#------------------------------------------------------------------------------#
#--- Region: Steps to Restore PostgreSQL databases
#------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Restore Zoomdata databases without prompting password  - using pgpass.conf file
# 3. Start all Zoomdata services
#endregion

#------------------------------------------------------------------------------#
#--- Function: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#------------------------------------------------------------------------------#
#region
Function StopAllZoomServices()
{
    Try
        {
            # Check zoomdata_consul is running and stop this - This will stop all the other required
services
            Write-Host "Running the StopAllZoomServices function and stopping all the zoomdata services."
            Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  |
stop-service -ErrorAction Stop -Force -Verbose

            # Check if any other zoomdata services are still running and stop them:
            # Only run this if the zoomdata_consul service is successfully stopped:
            if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} |
stop-service -ErrorAction Stop -Verbose
                }
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StopAllZoomServices function."
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Function: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#------------------------------------------------------------------------------#
#region
Function StartAllZoomdataServices()
{
    Try
        {
            # Check zoomdata_consul is stopped and start this:
            Write-Host "Running the StartAllZoomdataServices function and starting all the zoomdata
services."
            Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  |
start-service -ErrorAction Stop -Verbose

            # Check if any other zoomdata services are stopped and start them:
            # Only run this if the zoomdata_consul service is successfully running:
            if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} |
start-service -ErrorAction Stop -Verbose
                }
        }
```

2

```
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StartAllZoomdataServices
function."
        }
}
#endregion

#-------------------------------------------------------------------------------#
#--- Region: RestoreZoomdataDatabases
#--- Description: This will restore the specified zoomdata databases
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/15/app-pgrestore.html
#-------------------------------------------------------------------------------#
#region
# Define a list of Zoomdata databases to restore:
$DatabasesToRestore = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define restore folder location - Where all the .dump files reside:
$RestoreFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"
# Please note: The pgpass file %APPDATA%\postgresql\pgpass.conf file needs to exist according to the link
below in order for the connection to work:
# http://www.postgresql.org/docs/current/static/libpq-pgpass.html

# Test if restore folder exist:
if (-not(Test-Path $RestoreFolder))
    {
        Write-Host "The restore folder: $RestoreFolder does not exit. Process cannot continue."
        return
    }

# Stop all the services:
StopAllZoomServices

# Navigate to the restore folder location:
cd $RestoreFolder

# Loop through list of databases and perform a restore:
Foreach($DatabaseToRestore in $DatabasesToRestore)
    {
        # Filename: .dump:
        $RestoreFileName = $DatabaseToRestore + ".dump"

        # Test if the backup file exist - For example: zoomdata.dump:
        $FullPathToRestoreFile = "$($RestoreFolder)\$($RestoreFileName)"

        if (-not(Test-Path $FullPathToRestoreFile))
            {
                Write-Host "The restore file: $FullPathToRestoreFile does not exist. The database:
$DatabaseToBackup cannot be restored."
            }
        else
            {
                Write-Host "Restoring the following database: $DatabaseToRestore from the following file:
$FullPathToRestoreFile"

                Try
                    {
                        pg_restore --create --host=$PostgreSQLHostName --port $PostgreSQLPort --
username=$PostgreSQLUser --dbname postgres $FullPathToRestoreFile
                    }
                Catch
                    {
                        $ErrorMessage = $_.Exception.Message
                        Write-Host "The following Error occured: $ErrorMessage creating a backup of the
following database: $DatabaseToBackup"
                    }
```

2

```
            }
    }

# Start all the services:
StartAllZoomdataServices
#endregion
```

2 Use a connection URI to authenticate to PostgreSQL

RestoreZoomdata_NoPrompt_URI.ps1

```
#------------------------------------------------------------------------------#
#--- Region: Steps to Restore PostgreSQL databases
#------------------------------------------------------------------------------#
#region
# 1. Stop all Zoomdata services
# 2. Restore Zoomdata databases without prompting password - using connection URI
# 3. Start all Zoomdata services
#endregion

#------------------------------------------------------------------------------#
#--- Function: StopAllZoomdataServices
#--- Description: This will stop all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be stopped last due to dependencies
#------------------------------------------------------------------------------#
#region
Function StopAllZoomServices()
{
    Try
        {
            # Check zoomdata_consul is running and stop this - This will stop all the other required
services
            Write-Host "Running the StopAllZoomServices function and stopping all the zoomdata services."
            Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"}  |
stop-service -ErrorAction Stop -Force -Verbose

            # Check if any other zoomdata services are still running and stop them:
            # Only run this if the zoomdata_consul service is successfully stopped:
            if (Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -like "Zoomdata*"} |
stop-service -ErrorAction Stop -Verbose
                }
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StopAllZoomServices function."
        }
}
#endregion

#------------------------------------------------------------------------------#
#--- Function: StartAllZoomdataServices
#--- Description: This will start all the required Zoomdata services
#--- IMPORTANT: The zoomdata_consul service needs to be started first due to dependencies
#------------------------------------------------------------------------------#
#region
Function StartAllZoomdataServices()
{
    Try
        {
            # Check zoomdata_consul is stopped and start this:
            Write-Host "Running the StartAllZoomdataServices function and starting all the zoomdata
services."
            Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -eq "zoomdata_consul"}  |
start-service -ErrorAction Stop -Verbose

            # Check if any other zoomdata services are stopped and start them:
            # Only run this if the zoomdata_consul service is successfully running:
            if (Get-Service | Where-Object { $_.Status -eq "Running" -and $_.Name -eq "zoomdata_consul"})
                {
                    Get-Service | Where-Object { $_.Status -eq "Stopped" -and $_.Name -like "Zoomdata*"} |
start-service -ErrorAction Stop -Verbose
                }
        }
    Catch
        {
            $ErrorMessage = $_.Exception.Message
            Write-Host "The following Error occured: $ErrorMessage in the StartAllZoomdataServices
function."
        }
```

2

```
}
#endregion

#-------------------------------------------------------------------------------#
#--- Region: RestoreZoomdataDatabases
#--- Description: This will restore the specified zoomdata databases
#--- IMPORTANT: Please specify the PostgreSQL variables below
#--- More information on all the commands:
#--- https://www.postgresql.org/docs/15/app-pgrestore.html
#-------------------------------------------------------------------------------#
#region
# Define a list of Zoomdata databases to restore:
$DatabasesToRestore = @('zoomdata','zoomdata-keyset','zoomdata-qe','zoomdata-upload', 'zoomdata-user-
auditing')

# Define restore folder location - Where all the .dump files reside:
$RestoreFolder = "C:\Temp\SEA\Backup"

# PostgreSQL variables:
$PostgreSQLHostName = "127.0.0.1"
$PostgreSQLUser = "postgres"
$PostgreSQLPort = "5432"
$PostgreSQLPassword = "ComposerDBPassword" # This will be the password for the PostgreSQLUser user

# Test if restore folder exist:
if (-not(Test-Path $RestoreFolder))
    {
        Write-Host "The restore folder: $RestoreFolder does not exit. Process cannot continue."
        return
    }

# Stop all the services:
StopAllZoomServices

# Navigate to the restore folder location:
cd $RestoreFolder

# Loop through list of databases and perform a restore:
Foreach($DatabaseToRestore in $DatabasesToRestore)
    {
        # Argument Line to pass to pg_restore:
        # More reading on using URI: http://www.postgresql.org/docs/current/static/libpq-
connect.html#AEN42532
        $ArgumentLine =
"$($PostgreSQLUser):$($PostgreSQLPassword)@$($PostgreSQLHostName):$($PostgreSQLPort)/$($DatabaseToRestore)"

        # Filename: .dump:
        $RestoreFileName = $DatabaseToRestore + ".dump"

        # Test if the backup file exist - For example: zoomdata.dump:
        $FullPathToRestoreFile = "$($RestoreFolder)\$($RestoreFileName)"

        if (-not(Test-Path $FullPathToRestoreFile))
            {
                Write-Host "The restore file: $FullPathToRestoreFile does not exist. The database:
$DatabaseToBackup cannot be restored."
            }
        else
            {
                Write-Host "Restoring the following database: $DatabaseToRestore from the following file:
$FullPathToRestoreFile"

                Try
                    {
                        pg_restore --create --dbname $ArgumentLine postgres $FullPathToRestoreFile
                    }
                Catch
                    {
                        $ErrorMessage = $_.Exception.Message
                        Write-Host "The following Error occured: $ErrorMessage creating a backup of the
following database: $DatabaseToBackup"
                    }
            }
    }
```

2

```
# Start all the services:
StartAllZoomdataServices
#endregion
```

3

START ALL ZOOMDATA SERVICES

Ensure that you always start the **zoomdata_consul** service first, due to the other services' dependency on it.

Therefore, we recommend that you start the services in the following order:

a. zoomdata_consul
b. zoomdata_edc_* (i.e. data sources or connectors)
c. zoomdata_query_engine
d. zoomdata_data_writer_postgresql
e. zoomdata_screenshot_service
f. zoomdata

Potential Error Messages

PSQL not recognized as an internal or external command

Cause

After installing **PostgreSQL**, this error message can occur when you attempt to run `psql` from the **Command Prompt**.

The probable cause for this is that there is a missing environment variable for Postgres.

Solution

The following provides a guideline (using a machine running **Windows 10**) on how to add Postgres's bin directory to the `PATH` system variable.

> The `PATH` system variable allows for the location of executables by Windows (with the help of the **Command Prompt**) or the Terminal window.

1. Within the Windows search bar (accessible by pressing `WINDOWS+S`) type in the following:
   `environment`
2. Select the **Edit the system environment variables** option.
   The **System Properties** window is displayed.
3. Navigate to the **Advanced** tab and select the **Environment Variables** button.

   > You can also define the **Performance**, **User Profiles**, and **Startup and Recovery** options within this tab.

   The **Environment Variables** window is displayed.
4. From the **System variables** section, locate and select the `PATH` variable, followed by the **Edit** function.
   The **Edit environment variable** window is displayed. This screen lists all `PATH` variables which you can then edit, add to or delete.
5. Select the **New** function and enter a new path within the new editable row.

   > **FOR EXAMPLE:**
   >
   > C:\Program Files\PostgreSQL\12\bin\

6. Select **OK** to save the new `PATH` variable.

You must exit and reopen your **Command Prompt** for these changes and the new `PATH` variable to reflect.

FAQs

Zoomdata

What default zoomdata databases are shipped with SYSPRO Embedded Analytics?

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

What data is stored within the default zoomdata databases?

The metadata includes the following:

- Refresh schedule data
- Object information for your environment (e.g. sources, dashboards and visual definitions)
- Aggregated result sets

How do I add an additional zoomdata service/database to the PowerShell script?

Within the applicable PowerShell script, add the additional zoomdata service as the following:

`zoomdata-xxx`

Where `xxx` indicates the new database name.

Windows PowerShell

How do I use Windows PowerShell?

The following link provides detailed information of how to access and use **Windows PowerShell**:

https://learn.microsoft.com/en-us/powershell/scripting/learn/ps101/01-getting-started?view=powershell-7.3

What updates or changes must I make before using the sample scripts provided?

Apart from updating the details within the scripts to suit your environment, there are additional execution policies which apply to **Windows PowerShell** scripts.

For more information about **Windows PowerShell** execution policies, view the following link:

https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7.3

An important consideration is that most **Windows PowerShell** scripts must be digitally signed before they can be run. The link above explains this in more detail, however the following script can be used to enable the running of scripts that are not digitally signed:

Execution Policy

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

The parameters within this script are explained as follows:

- `Scope Process`:

  This limits the setting of the execution policy to the current process (i.e this setting is not stored anywhere).

- `ExecutionPolicy Bypass`:

  This ensures that nothing is blocked within the script and that no warnings or prompts are output.

View the following to learn more about digital script signing:

https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_signing?view=powershell-7.3

SYSPRO Services

How do I stop the SYSPRO Embedded Analytics services?

Stopping the **zoomdata_consul** service should stop all other services as they are dependent on the **zoomdata_consul** service.

What should I do if one of the services fail to start?

After a new install or upgrade, you might not be able to access SYSPRO Embedded Analytics until the **zoomdata** service has completed its checks and database updates (which can take up to 15 minutes).

To verify if this is the cause for your services not starting, open the **Task Manager** and look for a high running java process that belongs to **zoomdata**. If present, wait for the process to calm down, after which you should be able to continue.

If however, this does not resolve the problem, check to see if the port is already in use by another application.

See the **What ports does SYSPRO Embedded Analytics use** FAQ for information regarding port numbers.

If the above doesn't resolve the issue either, proceed as follows:

1. Check the log files for any error recorded.
2. Stop all of the zoomdata services.
3. Delete the log files.
4. Restart the zoomdata services in correct sequence:
    a. zoomdata_consul
    b. zoomdata_edc_* (i.e. data sources or connectors)
    c. zoomdata_query_engine
    d. zoomdata_data_writer_postgresql
    e. zoomdata_screenshot_service
    f. zoomdata

What ports are used by SYSPRO Embedded Analytics?

The following ports are used:

Port number	Service name	Comments
5432	zoomdata-postgres	Composer metadata repository
5580	zoomdata-query-engine	Composer query engine microservice
8090	zoomdata	Composer server
8081	zoomdata-data-writer	Composer Data Writer microservice
8083	zoomdata-screenshot-service	Composer Screenshot microservice
8100	zoomdata-edc-mssql	Microsoft SQL Server connector
8105	zoomdata-edc-postgresql	PostgreSQL connector, Flat File uploads, and Upload API processing
8300	zoomdata-consul	Internal port used by the Consul for inter-node communication in a distributed environment
8301	zoomdata-consul	Internal port used by the Consul for inter-node communication in a distributed environment
8302	zoomdata-consul	Internal port used by the Consul for inter-node communication in a distributed environment
8443	zoomdata	Composer HTTPS requests
8500	zoomdata-consul	Consul

SYSPRO™

www.syspro.com