Sitecore XP 10.1.2 Developer Workstation Deployment With Docker

How to install a Sitecore Experience Platform 10.1.2 on a developer workstation using containers with Docker

June 23, 2022

Table of Contents

1. Introduction to Docker Compose

Sitecore Experience Platform uses Docker Compose as the container orchestrator on developer workstations. Docker Compose is a simple container deployment tool that is bundled with Docker for Windows. You can use other tools to deploy Sitecore container images but we recommend that you use Docker Compose to deploy the containers that form the Sitecore Experience Platform.

This chapter contains the following sections:

1.1. Supported Sitecore topologies for Docker

You can install Sitecore XP on developer workstations using Docker containers.

Sitecore XP for Docker supports the following topologies:

- XP Workstation (XP Single)
- XM Server (XM Scaled)
- XP Server (XP Scaled)

All three topologies are included in the Sitecore Container Deployment package you can download from the Sitecore download page.

1.1.1. XP Workstation (XP Single)

The Sitecore Experience Platform Workstation for Docker topology, also known as *XP0*, is for developer workstation environments only. This topology is designed to reduce memory overhead, reduce download size, improve startup/shutdown time, and reduce complexity.

The XP0 topology supports the following Sitecore roles:

Role type	Sitecore role
Production	Sitecore Identity Server
Non-production	Content Management (Standalone)
	xConnect Server (Standalone)
	xConnect Search Indexer
	xDB Automation Engine
	Cortex Processing Engine
	Microsoft SQL Server
	Apache Solr

Role type	Sitecore role
	RedisLabs Redis Server
	Traefik Reverse Proxy

For a list of the supported environment variables, in the Container Deployment Package, in the Docker Compose folder for the XP0 topology, see the environment variable configuration file.

1.1.2. XM Server (XM Scaled)

The Sitecore Experience Manager Server for Docker topology, also known as *XM1*, is suitable for use in both production and non-production environments.

> **NOTE**
> To reduce deployment time and lower the resource overhead in non-production environments, you can remove the Content Delivery role from the Docker Compose configuration.

The XM1 topology supports the following Sitecore roles:

Role type	Sitecore role
Production	Content Management
	Content Delivery
	Sitecore Identity Server
Non-production	Microsoft SQL Server
	Apache Solr
	RedisLabs Redis Server
	Traefik Reverse Proxy

For a list of the supported environment variables, in the Container Deployment Package, in the Docker Compose folder for the XM1 topology, see the environment variable configuration file.

1.1.3. XP Server (XP Scaled)

The Sitecore Experience Platform Server for Docker topology, also known as *XP1*, is suitable for use in both production and non-production environments.

The resources required to run the XP1 topology in a non-production environment can be significant but are required to mimic the exact configuration that is used in production environments. In non-production environments, it is best practice to use a workstation that meets the minimum workstation hardware requirements.

The XP1 topology supports the following Sitecore roles:

Role type	Sitecore role
Production	Content Management
	Content Delivery
	Sitecore Identity Server
	xDB Processing

Role type	Sitecore role
	xConnect Collection
	xConnect Search
	xDB Automation Operations
	xDB Automation Reporting
	xDB Reference Data
	Cortex Processing
	Cortex Reporting
	xConnect Search Indexer
	xDB Automation Engine
	Cortex Processing Engine
Non-production	Microsoft SQL Server
	Apache Solr
	RedisLabs Redis Server
	Traefik Reverse Proxy

For a list of the supported environment variables, in the Container Deployment Package, in the Docker Compose folder for the XP1 topology, see the environment variable configuration file.

1.2. Sitecore Docker Compose requirements

There are a number of requirements that your environment must fulfill before you can deploy containers with Sitecore Docker compose. These include:

- Software requirements
- Hardware requirements
- Network requirements

1.2.1. Software requirements

You must have the following software installed in order to install Sitecore Experience Platform on Docker:

- One of the following operating systems:
  - Windows 10 1809 or later
  - Windows Server 1809 or later

> **NOTE**
> For more information about Windows and containers, see Microsoft's documentation.

- Docker Desktop for Windows

In addition, you must download the Sitecore Container Deployment Package from the Sitecore download page.

The package includes, among other things, two configuration files that are required by Sitecore Docker Compose:

- `docker-compose.yml`
  A Docker Compose configuration file that contains information about the different containers and configuration of each Sitecore role.
  Studying this file can help you understand how the containers and the connection strings between the different roles function.
- `.env`
  An environment variable configuration file that contains the configuration information for the environment you want to deploy. You can edit this file outside the main Docker Compose configuration.

1.2.2. Hardware requirements

The recommended minimum requirements for your workstation in order for it to run Sitecore Experience Platform on Docker are:

- RAM
  For the *XP1* server topology, we recommend a developer workstation with 32GB of RAM.
  For the *XM1* and *XP0* server topologies, we recommend a developer workstation with a minimum of 16GB of RAM.
- CPU
  We recommend a quad-core processor or higher.
- Disk
  The Sitecore container images require approximately 25 GB free space. It is best practice to use SSD disks for optimal performance when downloading and running Docker containers. The type of disks used for SQL Server and Solr can also have a significant impact on performance.

1.2.3. Network requirements

Before you deploy the Sitecore containers you must ensure that the following required TCP ports are available:

Required port	Role	Description
443	Traefik	HTTPS proxy
8079	Traefik	Traefik dashboard
8984	Solr	Solr API and dashboard
14330	SQL	SQL Server

1.3. Prepare for deploying

Before you start the process of deploying the Sitecore XP containers, there are some concepts and procedures you need to be familiar with.

These concepts and procedures are:

- Understanding environment variables
- Compressing the Sitecore license file
- Set up the Identity Server token signing certificate
- Generating TLS/HTTPS certificates
- Update the Windows host names
- Using non-production container images.

1.3.1. Understanding environment variables

Environment variables are the preferred mechanism for passing configuration settings into Sitecore containers.

The environment variable configuration file `.env` contains all the environment variables. Docker Compose loads these automatically during startup. The `.env` file for the Sitecore deployment is included in the Sitecore Container Deployment package.

> **IMPORTANT**
> Each environment variable must fit inside a 30,000 character block in the `.env` file.
> If the size of the variable exceeds 30,000 characters, the system will not deploy successfully.

If you want to reuse environment variables across multiple environments, you must set the environment variables in the Windows OS and remove the corresponding keys from the environment variable configuration file that is used by Docker Compose.

1.3.2. Compressing the Sitecore license file

Sitecore license files are typically passed to container instances as an environment variable in string form. However, the Sitecore license file is very large and you must therefore compress and *Base64* encode it to conform with the maximum size allowed by Windows for all the environment variables.

The appendix Encode and compress the Sitecore license file contains a sample PowerShell script you can use to convert a license file into a Base64 compressed string for use in an environment variable.

When you have compressed and encoded the license file, copy the string value to the SITECORE_LICENSE variable in the environment variable configuration file or set it as a Windows system environment variable.

Some Sitecore license files are so large that they are incompatible with containers even after compression. This usually happens when the license file contains additional embedded HTML.

As a workaround, you can mount the license file as a Docker volume from the host to the `c:\inetpub\wwwroot\app_data\license.xml` file inside the container. For more information and a configuration example, see the Sitecore Container Development documentation.

1.3.3. Set up the Identity Server token signing certificate

Sitecore Identity Server requires a private key certificate to sign the tokens that are passed between the server and the clients. You must generate this certificate, *Base64* encode it in string form, and pass it to the container as an environment variable.

To set up the certificate:

1. Use the sample script in Create the Identity Server token signing certificate to generate a self-signed certificate. The script Base64 encodes the certificate and creates the `SitecoreIdentityTokenSigning.txt` file containing the certificate.

   NOTE
   The script asks you to supply a certficate password. Make sure you write the password you choose down, as you need it to configure the certificate.

2. In the `.env` file, update the `SITECORE_ID_CERTIFICATE` variable with the certificate from the `SitecoreIdentityTokenSigning.txt` file.
3. In the `.env` file, update the `SITECORE_ID_CERTIFICATE_PASSWORD` variable with the password that you supplied when you generated the certificate.
4. Save the `.env` file.

NOTE
Instead of passing the Sitecore Identity Server certificate as an environment variable, you can mount it on the file system. For more information and a configuration example, see the Sitecore Container Development documentation.

1.3.4. Generating TLS/HTTPS certificates

To satisfy modern browser requirements and provide a secure environment by default, you must generate certificates for TLS (Transport Layer Security) before you deploy the Sitecore containers. This ensures secure communication between the browser and the HTTPS reverse proxy container.

The default reverse proxy or edge router used by the Sitecore Experience Platform in Docker Compose is Traefik. The Traefik edge router is used as a reverse proxy to the individual XP containers and terminates the TLS connections sent by the browser. For more information, see the Traefik documentation about TLS configuration.

You can view the reverse proxy configuration in the Traefik dashboard. When you have completed the deployment, you can see the dashboards at http://localhost:8079.

All internal communication between the Traefik edge router and the individual XP containers is sent unencrypted with the HTTP protocol.

You must use the HTTPS protocol between the reverse proxy and client browsers to support the secure browser cookies used by the Sitecore Content Management, Sitecore Content Delivery, and Identity Server roles.

The appendix Generate TLS/SSL certificates for reverse proxy contains a sample script that generates the required certificates. When you have generated the self-signed root authority certificate and per-host TLS/SSL certificates, you must install the root authority certificate in the Trusted Root Certificate Authority store on all the clients.

1.3.5. Update the Windows host names

To access the Sitecore application from a browser, you must add the host names that are used by the reverse proxy container to the Windows hosts file. The default host names vary depending on the topology you decide to deploy.

> **NOTE**
> All the host names must point to the loopback IP address 127.0.0.1.

The following table lists the default hostnames for each topology:

Topology	Hostname
XM Server (XM1)	xm1cm.localhost
	xm1cd.localhost
	xm1id.localhost
XP Workstation (XP0)	xp0cm.localhost
	xp0id.localhost
XP Server (XP1)	xp1cm.localhost
	xp1cd.localhost
	xp1id.localhost

To change the default hostnames:

1. Generate TLS certificates with the new hostnames.
2. In the `.env` file, update the Traefik reverse proxy configuration labels for each role with the new host names.
   For more information, see the documentation for Traefik Docker configuration discovery.
3. Update the `.crt` and `.key` filenames in the `cert_config.yaml` file.
4. Update the host names in the Windows `hosts` file.

1.3.6. Using non-production container images

To help developers get started quickly, Sitecore uses container images for the required services.

> **NOTE**
> The `.env` file in the Sitecore Experience Platform container package contains the information you need to access the images for your chosen version and topology.

The images include:

- Sitecore roles
- Third party software - SQL Server, Redis, and Solr
- Proxy service - Traefik

> **WARNING**
> These images are for *non-production* use only.

The non-production images are not supported by Sitecore in a production environment. The non-production services do not follow the recommended best practices for hosting a production environment and should not be considered as a basis for production environments.

Sitecore uses non-production Docker images for Microsoft SQL Server, Apache Solr, traefik from Traefik Labs, and RedisLabs Redis that are only for use on developer workstations. These images are preloaded with the required database and search configurations specific to each product and are designed to facilitate rapid deployment.

2. Deploy a workstation

You use Docker for Windows to deploy the Sitecore Experience Platform (SXP) container packages.

To deploy an SXP developer workstation on containers:

1. In Docker for Windows, switch to Windows container mode.
2. Download the SXP Container Deployment Package from the Sitecore download pagel and extract it to a folder on your local workstation. Navigate to the `compose\<windows version>\<topology>` folder for the topology that you want to deploy, for example, `compose\ltsc2019\xp1`.
3. In the topology folder, open the environment configuration file `.env` in a text editor. Update all the environment variables with the appropriate values. The variables include passwords, encryption keys, certificates, and the Sitecore license file. For more information, see the Environment variable list.
4. To generate the required TLS reverse proxy certificates, execute the PowerShell script in Generate TLS/SSL certificates for reverse proxy.
5. Install the self-signed root authority certificate that you just generated in the local Trusted Root Certification Authority certificate store on your local computer. The generated root certificate path is `./traefik/certs/rootca.crt`.
6. In the Windows console, go to the folder that the `docker-compose.yml` file is in and run the following Docker Compose command:

   ```
   docker-compose.exe up --detach
   ```

   Docker Compose pulls all the required images from the Sitecore Container Registry, creates the required Docker network configuration, and deploys all the containers to the local environment.

   > **NOTE**
   > The required images include Sitecore roles and third party services for your chosen topology.

   When the deployment is successfully completed, the Docker Compose command exits.
7. To check the Docker container status, run the following command:

   ```
   docker-compose.exe ps
   ```

   This command generates a list of all the containers and their current status.
8. When the status of all the containers is listed as *healthy*, open a browser and enter the URL for the content management instance.
   The default URLs for the XP0 topology are:

   - `https://xp0cm.localhost`
   - `https://xp0cd.localhost`

> **NOTE**
> You can find the default host names for the other topologies in the Update the Windows host names topic.

The content management and content delivery instances use the HTTPS protocol on port 443. If this port is in use by another process, you get an error message.

9. Login to the Sitecore admin section and verify that there are no errors in the Sitecore logs.

2.1. Rebuild the search indexes

When the deployment is done, you must rebuild the search indexes.

To rebuild the search indexes:

1. On the Sitecore Launchpad, open the Control Panel.
2. In the **Indexing** section, click **Populate Solr Managed Schema**.
3. In the **Schema Populate** dialog box, click **Select All**, then click **Populate**. Wait for the process to finish.
4. On the Control Panel,cewasIn the **Indexing** section, click **Indexing Manager**.
5. In the **Indexing Manager** dialog box, click **Select All**, then click **Rebuild**.

2.2. Deploy custom modules

For Sitecore components, by default, you deploy only the Solr collections and dacpacs included in the platform. In Sitecore 10.1 and later versions you can also deploy custom modules.

If the module you want to deploy requires database updates and/or custom Solr collections, you might need to build a new layer on top of the `mssql-init` and/or `solr-init` images.

2.2.1. Add database updates to a module

To add database updates to a module:

1. Build a new layer on top of the `mssql-init` image.
2. In your build-script, add a command for the following action:
   - Copy the module dacpacs from the module asset image into the `c:\module_name_data` folder for the new image.

2.2.2. Add Solr collections to a module

To add Solr collections to a module:

1. Build a new layer on top of the `solr-init` image.
2. Add the Solr collections module configuration files from the module assets image to the `c:\data` folder.

> **NOTE**
> You must use the JSON file format for collections. If the module name is `sxa`, you can, for example, name the Solr collections file `cores-sxa.json`.

2.3. Removing the Docker environment

With Docker Compose commands, you can stop, resume, or remove a workstation environment.

The basic commands are:

- To stop a Docker Compose environment without removing its contents:

```
docker-compose.exe stop
```

- To resume a previously stopped Docker Compose environment:

```
docker-compose.exe start
```

- To remove a Docker Compose environment and all the non-mounted volumes:

```
docker-compose.exe down
```

3. Appendices

This section contains additional information and helper functions to help you with deployment to Docker.

The topics include:

- Encode and compress the Sitecore license file
- Create the Identity Server token signing certificate
- Generate TLS/SSL certificates for reverse proxy
- Environment variable list
- Common issues

3.1. Encode and compress the Sitecore license file

To compress the Sitecore license file and encode it to *Base64*:

1. Create a PowerShell command file, and enter the following function in it:

```
function ConvertTo-CompressedBase64String {
  [CmdletBinding()]
  Param (
    [Parameter(Mandatory)]
    [ValidateScript( {
          if (-Not ($_ | Test-Path) ) {
            throw "The file or folder $_ does not exist"
          }
          if (-Not ($_ | Test-Path -PathType Leaf) ) {
            throw "The Path argument must be a file. Folder paths are not allowed."
          }
          return $true
        })]
    [string] $Path
  )
  $fileBytes = [System.IO.File]::ReadAllBytes($Path)
  [System.IO.MemoryStream] $memoryStream = New-Object System.IO.MemoryStream
  $gzipStream = New-Object System.IO.Compression.GzipStream $memoryStream, ([IO.Compression.CompressionMode]::Compress)
  $gzipStream.Write($fileBytes, 0, $fileBytes.Length)
  $gzipStream.Close()
  $memoryStream.Close()
  $compressedFileBytes = $memoryStream.ToArray()
  $encodedCompressedFileData = [Convert]::ToBase64String($compressedFileBytes)
  $gzipStream.Dispose()
  $memoryStream.Dispose()
  return $encodedCompressedFileData
}
```

2. Run the PowerShell command file, specifying the path to the license file in the `Path` parameter. For example:

```
ConvertTo-CompressedBase64String -Path .\license.xml
```

3. In the `.env` file, enter the resulting string into the `SITECORE_LICENSE` variable.

> **NOTE**
> Before pasting the resulting license string into the `.env` file, make sure that there are no line breaks (`\n`) in the string.

3.2. Create the Identity Server token signing certificate

To create the Identity Server token signing certificate:

- Use the following PowerShell script:

```
$newCert = New-SelfSignedCertificate -DnsName "localhost" -FriendlyName "Sitecore Identity
Token Signing" -NotAfter (Get-Date).AddYears(5)
Export-PfxCertificate -Cert $newCert -FilePath .\SitecoreIdentityTokenSigning.pfx -Password
(ConvertTo-SecureString -String "Test123!" -Force -AsPlainText)

[System.Convert]::ToBase64String([System.IO.File]::ReadAllBytes((Get-
Item .\SitecoreIdentityTokenSigning.pfx))) | Out-File -Encoding
ascii -NoNewline -Confirm -FilePath .\SitecoreIdentityTokenSigning.txt
```

3.3. Generate TLS/SSL certificates for reverse proxy

To generate the TLS/SSL certificates that are required by the Traefik reverse proxy container:

1. Open a Windows Command Prompt with Administrator rights.
2. Navigate to the folder containing the `docker-compose.yml` file.
3. Run the following commands one by one:

> **NOTE**
> This example uses the default filenames for the XM1 topology for the generated `.crt` and `.key` files, for example, `xm1cm.localhost.crt`. The filenames are found in the `compose\<version>\<topology>\traefik\config\dynamic\cert_config.yaml` file. If you use other filenames than the defaults for your topology, you must update the filenames in the `cert_config.yaml` file.

```
IF NOT EXIST mkcert.exe powershell Invoke-WebRequest https://github.com/FiloSottile/
mkcert/releases/download/v1.4.1/mkcert-v1.4.1-windows-amd64.exe -UseBasicParsing -OutFile
```

```
mkcert.exe

mkcert -install

del /Q /S traefik\certs\*

mkcert -cert-file traefik\certs\xm1cm.localhost.crt -key-file
traefik\certs\xm1cm.localhost.key "xm1cm.localhost"

mkcert -cert-file traefik\certs\xm1id.localhost.crt -key-file
traefik\certs\xm1id.localhost.key "xm1id.localhost"

mkcert -cert-file traefik\certs\xm1cd.localhost.crt -key-file
traefik\certs\xm1cd.localhost.key "xm1cd.localhost"
```

NOTE

The first time the `mkcert` utility runs, it might prompt you to install the generated self-signed root certificate authority.

3.4. Environment variable list

The following table describes the environment variables and lists their default values.

Variable name	Default value	Description
SITECORE_DOCKER_REGISTRY	`scr.sitecore.com/sxp/`	Sitecore container registry
SITECORE_VERSION	`10.1.2-ltsc2019`	Image tag with the version to be pulled from the container registry
SITECORE_ADMIN_PASSWORD		Sitecore application administrator password
SQL_SA_PASSWORD		SQL Server administrator password
REPORTING_API_KEY		Symmetric key used to access the Sitecore XDB Processing Service. Length: 64-128 characters
TELERIK_ENCRYPTION_KEY		Symmetric key used by the Telerik web controls. Length: 64-128 characters
SITECORE_IDSECRET		Shared secret between the Identity Server and client roles. Length: 64 characters
SITECORE_ID_CERTIFICATE		Identity Server certificate used to encrypt data
SITECORE_ID_CERTIFICATE_PASSWORD		Password to open the Identity Server certificate

Variable name	Default value	Description
SITECORE_LICENSE		License file content converted to GZIP Compressed and Base64 encoded string
ISOLATION	`default`	Override for Docker isolation level (Possible values: `default`, `hyperv`, `process`)
SOLR_CORE_PREFIX_NAME	sitecore	A common prefix for Solr core names. If you use an existing Solr deployment, you must change this to your actual Solr core name prefix.
MEDIA_REQUEST_PROTECTION_SHARED_SECRET		Shared secret. You must change this to a random string. Do not use the default value.

3.5. Common issues

This section lists some common issues related to the installation of a Developer Workstation with containers, along with proposed solutions.

3.5.1. I cannot upload a Translations file to the website root folder

If the **Upload Files** dialog hangs during the upload operation, it writes the following errors to the log file:

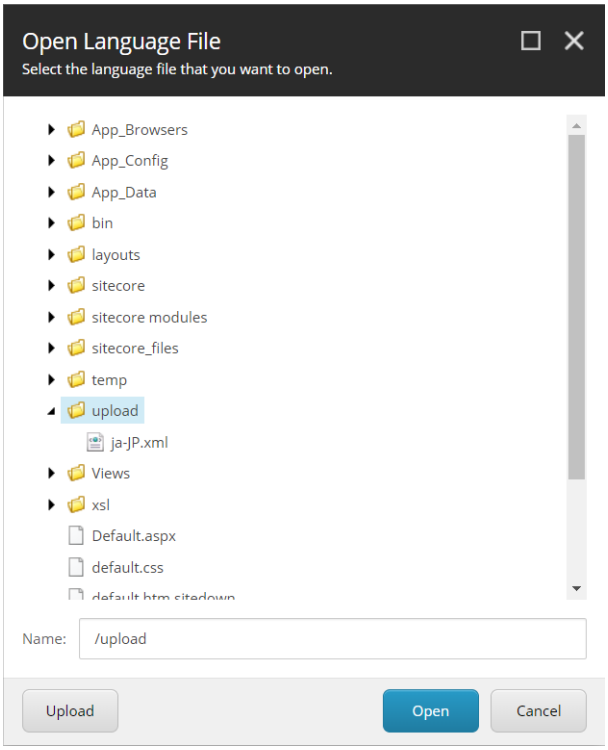
```
ERROR Could not save posted file: ja-JP.xml
Exception: System.UnauthorizedAccessException
Message: Access to the path 'C:\inetpub\wwwroot\ja-JP.xml' is denied.
```

This happens because the **Import language** dialog box uploads translations files to the `website` root folder by default and for security reasons *Write* access is denied for the website root folder.

The `upload` folder, however, has *Write* access enabled. To resolve this issue, upload the translations files to the `upload` folder.

To upload a translations file:

1. In the **Open Language File** dialog box, select the `upload` folder and then click **Upload**.



2. In the **Upload Files** dialog box, upload the translations file to the `upload` folder.
3. Import the translations file from the `upload` folder.

> **NOTE**
> The translations xml file is saved to the Media library. You can delete it after you import the translations.

3.5.2. I can only see the main Sitecore log files for the Sitecore roles containers

The LogMonitor tool collects the log files for containers. By default, it monitors the following log files:

- System event log – error level entries
- IIS logs
- Primary Sitecore log – `log.*.txt` files for the Sitecore roles
- xConnect log – `xconnect-log-*.txt` files for the xConnect roles

Auxiliary Sitecore log files, such as for search, crawling, and publishing, are not monitored on Sitecore containers.

To see all the Sitecore log files for a Sitecore role container, you must create a Dockerfile with the corresponding role image and reconfigure the LogMonitor tool or replace the entire configuration file with the updated configuration.

To reconfigure the LogMonitor tool:

1. In the `c:\inetpub\wwwroot\App_data\logs` folder, open the `C:\LogMonitor\LogMonitorConfig.json` file.
2. In the `sources` node, edit the `filter` setting:

```
{
    "LogConfig": {
        "sources": [
            ...
            {
                "type": "File",
                "directory": "c:\\inetpub\\wwwroot\\App_data\\logs",
                "filter": ".*log*.txt",
                "includeSubdirectories": false
            }
        ]
    }
}
```

Now you can use the Dockerfile to build a new docker image for the Sitecore role.

You can also view all the Sitecore log files directly from the container's file system by connecting to the corresponding container from a PowerShell or command prompt terminal.

3.5.3. I am experiencing performance issues on Hyper-V

If you are running your content management (CM) server on Hyper-V, and you experience performance issues, such as the server hanging or crashing during resource consuming tasks, the server might not have enough memory. The default memory amount for Hyper-V is 1 GB, however, we recommend a minimum of 4 GB to avoid performance issues.

To set a minimum memory limit of 4 GB:

- In the Container Deployment Package, in the `docker-compose.yml` file, in the CM service configuration, add `mem_limit:4GB`.