Sitecore XP 10.1.1 Production Deployment With Kubernetes

How to deploy Sitecore Experience Platform 10.1.1 to a container installation using Kubernetes.

October 10, 2022

Table of Contents

1. Introduction to Azure Kubernetes Service

Sitecore Experience Platform (SXP) uses Kubernetes (K8s) as the default orchestrator for deploying production environments.

This guide describes how to deploy the SXP containers to the Azure Kubernetes Service (AKS). AKS is a Microsoft cloud hosted Kubernetes service with additional functionality that takes advantage of Azure specific features like blob storage and load balancing.

The SXP Kubernetes specification files you use to map the minimum required configuration parameters between the Sitecore software containers are provided in the SXP container package as a reference. You can extend these specifications to support your own requirements. It is your responsibility to ensure that your production deployments meet the standards for stability and security set by your organization.

The SXP for Kubernetes specification files are designed to avoid Azure specific dependencies where possible.

1.1. Supported Sitecore topologies for Kubernetes

Sitecore XP supports the following topologies for use with Kubernetes:

- XM Server (XM Scaled)
- XP Server (XP Scaled)

1.1.1. XM Server (XM Scaled)

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

The *XM1* topology supports the following Sitecore roles:

Role type	Sitecore role
Production	Content Management
	Content Delivery
	Sitecore Identity Server
	MSSQL Data Intialization Job
	SolrCloud Data Initalization Job
Non-production	Microsoft SQL Server
	SolrCloud
	RedisLabs Redis Server

1.1.2. XP Server (XP Scaled)

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

If you are deploying the XP1 topology in a non-production environment for, for example, testing purposes, be aware that mimicking the configuration of a production environment requires significant resources.

The *XP1* topology supports the following Sitecore roles:

Role type	Sitecore role
Production	Content Management
	Content Delivery
	Sitecore Identity Server
	XDB Processing
	XDB Collection service
	XDB Search service
	Marketing Automation Engine
	Marketing Automation Reporting
	XDB Reference Data service
	Sitecore Cortex Processing service
	Sitecore Cortex Reporting service
	XDB Search Worker
	Sitecore Cortex Processing
	MSSQL Data Initialization Job
	SolrCloud Data Initialization Job
Non-production	Microsoft SQL Server
	SolrCloud
	RedisLabs Redis Server

1.2. Requirements for Sitecore on Azure Kubernetes Service

There are a number of requirements that your environment must fulfill before you can deploy Sitecore containers in an Azure Kubernetes Service environment. These include:

- Sitecore Container Deployment Package
- Client software requirements
- Kubernetes Cluster software requirements
- Kubernetes Cluster hardware requirements

- Required external data services
- Azure Kubernetes Service requirements
- Ingress Controller requirements

1.2.1. Sitecore Container Deployment Package

The Sitecore XP 10.1.1 Container Deployment Package contains the Kubernetes specification files that you use to deploy a Sitecore software cluster solution.

For more information about the supported topologies in the Sitecore Container Deployment Package, see Supported Sitecore topologies for Kubernetes.

1.2.2. Client software requirements

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

- One of the following operating systems:
  - Windows 10 1809 or later
  - Windows Server 1809 or later
- Kubernetes 1.16x or later. Use the latest stable non-preview version.
  - To get a list of supported locations run the following command:

    ```
    az account list-locations
    ```

  - To get the latest stable version with the desired location, run the following command:

    ```
    az aks get-versions --location <location> --output table
    ```

- Helm 3.0.x or later. This is required for ingress controller deployments.
  We recommend that you use `choco` to install Helm. For example:

  ```
  choco install kubernetes-helm
  ```

- Deploying the Sitecore specification files on AKS requires Azure CLI 2.8.0 or later.

You also need to download the following software package:

- Sitecore SXP 10.1.1 Container Deployment Package

1.2.3. Kubernetes Cluster software requirements

- Kubernetes 1.16.x or later
  Kubernetes 1.18.x or later is recommended to enable startup probes.
- Windows Server 2019 version 1809 or Windows 10 version 1809.

1.2.4. Kubernetes Cluster hardware requirements

Windows Server 2019 clusters have the following hardware requirements:

- RAM

We recommend a minimum of 16 GB RAM per Kubernetes cluster during startup.
Your operational requirements depend on your use of the Azure service.

- CPU
  We recommend a quad core or higher per Kubernetes node during startup.
  Your operational requirements depend on your use of the Azure service.
- Disk
  We recommend premium SSD disks for optimal performance when downloading and running Docker containers.
  The operational capacity depends on the service that you use.

1.2.5. Required external data services

In production environments, external data services must be hosted outside the Sitecore XP cluster.

To reduce the time required for development and testing, sample external service deployments for K8s are provided for non-production use only.

The following services are required:

- Microsoft SQL Server
  - Microsoft SQL Server 2017 or 2019
    or
  - SQL Azure Elastic Database Pool
- Apache Solr Cloud 8.4.0
- RedisLabs Redis 4.0 or higher

1.2.6. Azure Kubernetes Service requirements

To deploy Sitecore on Azure Kubernetes Service (AKS), you must meet the following requirements for an AKS installation:

- An AKS cluster configured with the latest stable release of Kubernetes – version 1.16.x or later.

  NOTE
  For startup probes to check whether the Sitecore software container has started successfully, Kubernetes version 1.18.x or later is required.

- One Windows Server 2019 version 1809 OS node.
- For non-production environments and testing, the recommended minimum VM size for Windows and Linux nodes is Standard_D4s_v3.
- For production environments, the VM size and number of nodes depend on your individual requirements.

To get the latest version of Kubernetes supported by AKS, run the following Azure CLI command:

```
az aks get-versions --location <location> --output table
```

NOTE
In the command, replace `location` with the region you want to check for.

1.2.7. Ingress Controller requirements

You must have a Kubernetes ingress controller to deploy the Sitecore Kubernetes specification files.

The Sitecore Kubernetes specification files work with most Ingress Controllers supported by Kubernetes, but not all ingress controllers operate the same way. For more information, see the third party documentation for the ingress controller configuration for production deployments.

To support client IP address tracking and personalization, you must configure the Ingress Controller to preserve the client source IP address in the *X-Forwarded-For* HTTP header.

To fully enable IP address tracking and personalization, you must make some additional changes to the Sitecore software configuration.

For more information about the Ingress Controller and Sitecore software configuration, see the *Ingress Controller service* section.

1.3. Prepare for deploying to Azure Kubernetes Service

Before you start the process of deploying the Sitecore XP containers to Azure Kubernetes Service (AKS), there are some concepts and procedures you need to be familiar with.

These concepts and procedures are:

- Understanding Sitecore Kubernetes specification files
- Accessing the Sitecore container registry
- Prepare the Kubernetes specification files for deployment
- Deploying the Kubernetes Secrets
- Compressing the Sitecore license file
- Generating the Identity Server token signing certificate
- Generating TLS/HTTPS certificates
- Using non-production container images for external services
- Hosting external data services
- Using submit queue persistent storage
- Configuring the ingress controller service
- Using a private container registry

1.3.1. Understanding Sitecore Kubernetes specification files

Sitecore provides Kubernetes specification files (`.yaml`) that you use to deploy the containers to a Kubernetes cluster.

You use the remote client Kubectl (Kube control) to configure the Kubernetes clusters and specify the desired configuration state.

Kubectl is available as part of the Azure CLI and as a standalone.

1.3.2. Accessing the Sitecore container registry

The Sitecore XP container images are hosted in a public Docker container registry and are available without authentication.

This public registry is the default registry used by the Sitecore Kubernetes specification files.

To start the Sitecore software container images, you must have a valid Sitecore license file.

The Sitecore Container Registry, which you access through Docker, is hosted at `scr.sitecore.com` and supports the Docker content trust model that lets you pull signed images.

1.3.3. Prepare the Kubernetes specification files for deployment

To deploy the Sitecore Kubernetes specification files, you must use the Kubernetes Kubectl CLI.

To prepare for deployment:

1. Download the Sitecore Experience Platform 10.1.1 Container Deployment Package from the Sitecore download page, and extract it to a temporary folder on your installation.
2. Locate the `k8s\<version>\<topology>` folder, where `<version>` is your Windows version and `<topology>` is the Sitecore topology you use. The folder can be, for example, `k8s\ltsc2019\xp1`.

   > **NOTE**
   > Sitecore currently supports the Sitecore XP Server (XP1) and Sitecore XM Server (XM1) topologies on Kubernetes.
   >
   > For more information about the supported topologies, see Supported Sitecore topologies for Docker.

3. To familiarize yourself with the specifications, inspect the folder contents and the Kubernetes specification files (`.yaml`)

1.3.4. Deploying the Kubernetes Secrets

Sitecore Kubernetes deployments use Secrets to securely store the strings the containers in the cluster use.

The Secrets are used to store database user names, passwords, and TLS certificates. The Secrets are configured in text files and certificate files (`tls.crt` and `tls.key`) and are stored in the Kubernetes specification files for each topology in the `./secrets/` folder.

You must deploy the Secrets to the K8s cluster before you deploy any Sitecore containers.

You must update the Secret text files (`.txt`, `.crt`, .key) with the required values before you deploy any additional resources to the K8s cluster.

> **NOTE**
> The content in secrets must be encoded in UTF-8 format.

For a complete list of Secrets and more information about the individual Secrets, see The Kubernetes Secrets list.

In the Sitecore Experience Platform container package, in the `k8s\<version>\<topology>\secrets` folder, there is a Kubectl `kustomization.yaml` file that deploys all the Secret names and values with a single command.

1.3.5. Compressing the Sitecore license file

The Sitecore license file is typically passed to the container instances as an environment variable in encoded string form. The Sitecore license file is very large. You must therefore compress and *Base64* encode it to ensure that it conforms with the maximum size allowed by Windows for all the environment variables.

> **NOTE**
> The appendix Encode and compress the Sitecore license file contains a sample PowerShell script that converts a license file into a Base64 compressed string for use in a K8s secret.

When you have compressed and encoded the license file, copy the string value to the license secret text file `sitecore-license.txt`.

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 container 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.

1.3.6. Generating 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 store it as a Secret in the Kubernetes cluster.

The appendix Create the Identity Server token signing certificate contains a sample script that generates a self-signed certificate and prepares the string for use as a Secret.

The sample script creates the certificate and copies the certificate password to the `sitecore-identitycertificatepassword.txt` Kubernetes Secret text file.

You can mount the Identity Server certificate on the filesystem instead of passing it as an environment variable. For more information about how to mount the token signing certificate as a volume, see the Sitecore Docker Demo repository on GitHub.

1.3.7. 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 Kubernetes ingress controller.

The default Kubernetes ingress controller used by Sitecore XP is the NGINX Ingress Controller. The NGINX ingress controller is used to terminate TLS connections sent by the browser and proxy network traffic to the individual XP containers inside the cluster. For more information, see the Kubernetes documentation for ingress TLS configuration and NGINX TLS user guide.

The HTTPS protocol is required to support the secure browser cookies used by the Content Management and Identity Server roles. HTTPS is enabled by default on the Content Delivery role but you can disable it if it is not required for your specific use case.

The appendix Create the TLS/HTTPS certificates contains a sample script that generates the required certificates.

Once the self-signed root authority certificate and per-host TLS/SSL certificates have been generated, you must install the root authority certificate in the Trusted Root Certificate Authority store on all clients. The sample script in the appendix Create the TLS/HTTPS certificates uses the *mkcert* tool to automatically create the self-signed root authority certificate and install it in the correct certificate store.

1.3.8. Using non-production container images for external services

In order to minimize the time it takes to deploy Sitecore Experience Platform (SXP) to Kubernetes clusters for non-production use, Sitecore uses open source container images for the external Microsoft SQL Server, Apache Solr, and RedisLabs Redis services.

You can find the yaml specification files for the external service containers in the SXP container package, in the `k8s\<version>\<topology>\external` folder.

> **WARNING**
> These open source container images are for *non-production* use only. They are not supported by Sitecore.

1.3.9. Hosting external data services

In production deployments, you must host the required Sitecore XP external services outside the Kubernetes cluster.

The external hosted services for Microsoft SQL Server, SolrCloud, and RedisLabs Redis are required for production Kubernetes support from Sitecore.

You must deploy and configure the external services for production use before you deploy Sitecore XP to Kubernetes.

To deploy the required database, search schemas, and the required data, the Sitecore container deployment package includes Kubernetes *data initialization* jobs for Microsoft SQL Server and SolrCloud.

> **NOTE**
> RedisLabs Redis external services do not require initialization. The required cache databases are created during first use.

You must complete the data initialization jobs before you deploy the Sitecore software containers.

For more information about data initialization, see the *Deploy Data Initialization Jobs* section in Deploy Sitecore XP to the Azure Kubernetes Service.

Application database user credentials

Sitecore Kubernetes specification files are designed to deploy containers that use application database users to connect to the databases. The MSSQL Data Initialization Job uses the credentials provided in the Sitecore-database Secret to create an application database user for each database.

The application database user name Secrets have default values. You must provide passwords for all of them.

For example, you specify the application database user credentials used to connect to the Sitecore Master database in the `sitecore-master-database-username.txt` and `sitecore-master-database-password.txt` Secrets files.

> **NOTE**
> The content in secrets must be encoded in UTF-8 format.

For a complete list of application database user credentials Secrets, see the appendix The Kubernetes Secrets list.

1.3.10. Using submit queue persistent storage

You can use the K8s persistent volume claim feature to make the Sitecore Submit Queue persistent across multiple content delivery instances.

A persistent volume can refer to one of two things:

- The persistent volume, which stores all the sessions on the Windows node.
- The storage class, which deploys the Azure Storage Account to store all the sessions. This approach is best practice for Azure Kubernetes Services.

1.3.11. Configuring the ingress controller service

You must have a Kubernetes ingress controller to deploy Sitecore Kubernetes specification files. The ingress controller acts as a reverse proxy between the browser and the Sitecore software containers.

To support client IP address tracking and personalization, you must configure the ingress controller to preserve the client source IP address in the `X-Forwarded-For` HTTP header.

To preserve the client source IP address, set the ingress controller `externalTrafficPolicy` setting to `Local`. For more information, see the Kubernetes documentation and the NGINX Ingress Controller Configuration documentation.

You must explicitly enable use of a proxy server in your Sitecore Configuration. For more information about setting up a proxy server, see the Sitecore Experience Platform documentation.

To change this configuration for use in Kubernetes, We recommend that you build a new container image that contains this configuration change for both the Content Management and Content Delivery roles.

For more information about building new container images with configuration changes, see the Sitecore Docker Examples repository on GitHub.

1.3.12. Using a private container registry

You can use a private container registry for authentication. To let a Kubernetes cluster authenticate with a private container registry and pull images from it, you must create an image pull Secret.

The Kubernetes deployment specifications for every Sitecore role support the use of an image pull Secret. The Secret name must be `sitecore-docker-registry`. For example:

```
imagePullSecrets:
- name: sitecore-docker-registry
```

You must change the registry path in the `.yaml` files for the images that are pulled from the private registry.

2. Deploy Sitecore XP to the Azure Kubernetes Service

This section describes the steps you need to perform to deploy Sitecore Experience Platform to the Azure Kubernetes Service. You must perform the steps in the order they are described here.

2.1. Deploy external data services

In production deployments, you must deploy the required external services before you deploy the Sitecore containers in Kubernetes.

The required external services are:

- Microsoft SQL Server *or* SQL Azure Database Elastic Pool
- SolrCloud
- RedisLabs Redis

In non-production deployments you can create the external services inside the K8s cluster. For more information, see the section Deploy external services for a non-production deployment.

2.2. Create an AKS cluster

To create a new Azure Kubernetes Service (AKS) cluster with a Windows Server 2019 node pool, use the Azure command-line interface (Azure CLI) or the Azure portal UI. The AKS cluster must contain one Windows Server 2019 version 1809 node pool with one or more nodes.

For more information about using the Azure CLI to create an AKS cluster, see the Azure AKS documentation.

2.3. Configure the Kubectl context cluster

To configure the Kubectl context cluster:

1. Log in to the Azure CLI and set a subscription. For example:

```
az login
az account set --subscription "Your Subscription"
```

2. Get the credentials for the K8s cluster that were created with the AKS cluster and save them locally. For example:

```
az aks get-credentials --resource-group sc10aks --name sc10cluster
```

2.4. Deploy an ingress controller

To deploy an ingress controller:

1. Use the Windows AMD64 binaries to Install Helm. You can also use an alternative method as described in Installing Helm Through Package Managers.
2. Add an NGINX ingress controller feed to Helm. For example:

```
helm repo add stable https://kubernetes.github.io/ingress-nginx
```

3. Use Helm to deploy the NGINX ingress controller. For example:

```
helm install nginx-ingress ingress-nginx/ingress-nginx `
  --set controller.replicaCount=2 `
  --set controller.nodeSelector."kubernetes\.io/os"=linux  `
  --set defaultBackend.nodeSelector."kubernetes\.io/os"=linux `
  --set controller.admissionWebhooks.patch.nodeSelector."kubernetes\.io/os"=linux
```

> **NOTE**
> In the example, the *proxy-body-size* parameter limits payload requests, such as media data upload or Sitecore packages installation, to 10 MB. You can adjust this to fit your installation. For more information about ingress configuration, see NGINX Ingress Controller Configuration.

4. From the root folder of your chosen topology, run this command:

```
kubectl apply -k ./ingress-nginx/
```

2.5. Deploy the secrets

To deploy the secrets:

1. Ensure that all the secrets files (.txt, .crt, .key) files in the `/secrets` and `overrides\<topology>\secrets` folders are updated according to the requirements listed in The Kubernetes Secrets list.
2. From the folder of your chosen topology, run this command:

```
kubectl apply -k ./secrets/
```

2.6. Deploy external services for a non-production deployment

In production environments you must deploy external services outside the Kubernetes cluster. However, in non-production environments you can deploy the services inside the Kubernetes cluster.

To deploy external services for a non-production deployment:

1. From the folder of your chosen topology, run this command:

```
kubectl apply -k ./external/
```

2. To check the status of the pods, run this command:

```
kubectl get pods -o wide
```

3. To wait until the status of all the pods is *Running/OK*, run this command:

```
kubectl wait --for=condition=Available deployments --all --timeout=900s
kubectl wait --for=condition=Ready pods --all
```

2.7. Deploy the data initialization jobs

To deploy the data initialization jobs:

1. If you use SearchStax as your SolrCloud provider, follow the instructions in Initialize data for SearchStax provider for Solr. If you do not use Searchstax, navigate to the folder of your chosen topology, and run this command:

```
kubectl apply -k ./init/
```

2. To check the status of the jobs, run this command:

```
kubectl get jobs -o wide
```

3. To wait until the status of all the jobs is Complete/OK, run this command:

```
kubectl wait --for=condition=Complete job.batch/solr-init --timeout=900s
kubectl wait --for=condition=Complete job.batch/mssql-init --timeout=900s
```

2.8. Deploy a persistent volume claim

To deploy a persistent volume claim, run this command:

```
kubectl apply -f ./volumes/azurefile
```

2.9. Deploy the Sitecore pods

To deploy the Sitecore pods:

1. From the folder of your chosen topology, run this command:

```
kubectl apply -k ./
```

2. To check the status of the pods, run this command:

```
kubectl get pods -o wide
```

3. To wait until the status of all the pods is Running/OK, run this command:

```
kubectl wait --for=condition=Available deployments --all --timeout=1800s
```

2.10. Update the local host file

To update the local host file:

1. To obtain the external IP address of the ingress controller service for the CM role, run this command:

```
kubectl get ingress
```

2. Update the local host file with the external IP address and the hostnames that are required by the ingress controller. The default hostnames are:
   - `cm.globalhost`
   - `cd.globalhost`
   - `Id.globalhost`

2.11. Configure the SolrCloud search indexes

When you have finished deploying the containers, you must update all the search indexes.

To configure the SolrCloud search indexes:

1. Open a browser and navigate to https://cm.globalhost/sitecore.
2. Log in to Sitecore with the admin user and password that you configured as a secret.
3. In the Sitecore Control Panel click **Populate Managed Schema**. In the **Schema Populate** dialog box, select all the indexes and click **Populate**.

4. In the Sitecore Control Panel, in the **Indexing** section, click **Indexing Manager**.
5. In the **Indexing Manager** dialog box, select the indexes you want to rebuild, and then click **Rebuild**.
6. When the search indexes have been rebuilt, click **Close**.

3. 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. You must use the module assets image as a source for the module dacpacs and Solr collections configuration files.

3.1. Add database updates to a module

To add the module database updates:

1. Build a new layer on top of the `mssql-init` image.
2. Copy the module dacpacs from the module asset image into the `c:\resources\module_name_data` folder.
3. In the Kubernetes specification for the `mssql-init` image, add the custom module to the DATABASES_TO_DEPLOY variable.

3.2. Add Solr collections to a module

To add a Solr collections 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`.

3. In the Kubernetes specification for the `solr-init` image, add the custom module to the SOLR_COLLECTIONS_TO_DEPLOY variable.

   NOTE
   If you add more than one module to one of the variables, you must separate the module names with commas. For example, SOLR_COLLECTIONS_TO_DEPLOY="sxa,jss".

4. Appendices

The appendices contain various helper functions and background information.

4.1. Encode and compress the Sitecore license file

You must compress the Sitecore license file and encode it to *Base64*. This procedure processes the license file and stores the encoded license key in the `.\secrets\sitecore-license.txt` file.

To compress and encode the license file:

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 | Out-File -Encoding ascii -NoNewline
-Confirm -FilePath .\secrets\sitecore-license.txt
```

4.2. Create the Identity Server token signing certificate

Use the following PowerShell script to create the Identity Server token signing certificate:

```
$certificatePassword = "Test123!"

$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 $certificatePassword -Force -AsPlainText)

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

4.3. Create the TLS/HTTPS certificates

To generate the TLS/SSL certificates that are required by the NGINX ingress controller:

1. Open a Windows Command Prompt with Administrator rights.
2. Navigate to the folder containing Kubernetes specification files for the topology you are using.
3. Run the following commands one by one:

```
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 *.crt

del /Q /S *.key

mkcert -cert-file secrets\tls\global-cm\tls.crt -key-file secrets\tls\global-cm\tls.key
"cm.globalhost"

mkcert -cert-file secrets\tls\global-cd\tls.crt -key-file secrets\tls\global-cd\tls.key
"cd.globalhost"

mkcert -cert-file secrets\tls\global-id\tls.crt -key-file secrets\tls\global-id\tls.key
"id.globalhost"
```

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

4.4. Initialize data for SearchStax provider for Solr

If you use SearchStax as your SolrCloud provider, to initialize your data:

1. In the `.\overlays\init\SearchStax` folder, locate the following files and fill in the SearchStax specific secrets:

Filename	Description of secret	Topology
`sitecore-searchstax-account-name.txt`	SearchStax account name	XM1, XP1
`sitecore-searchstax-apikey.txt`	SearchStax API key	XM1, XP1
`sitecore-searchstax-deployment-uid.txt`	SearchStax deployment UID	XM1, XP1

2. Run the following command:

```
kubectl apply -k .\overlays\init\SearchStax
```

4.5. The Kubernetes Secrets list

The following table describes the Kubernetes Secrets files.

> **NOTE**
> The content in secrets must be encoded in UTF-8 format.

Name	Description	Topology	Default value
`sitecore-license.txt`	License file content converted to GZIP Compressed and Base64 encoded string (See Encode and compress the Sitecore license file)	XM1, XP1	
`sitecore-adminpassword.txt`	Sitecore application administrator password	XM1, XP1	
`sitecore-telerikencryptionkey.txt`	Symmetric key used by the Telerik web controls Length: 64-128 characters	XM1, XP1	
`sitecore-identitycertificate.txt`	Identity Server certificate used to encrypt data (See Create the Identity Server token signing certificate)	XM1, XP1	
`sitecore-identitycertificatepassword.txt`	Password to open the Identity Server certificate	XM1, XP1	

Name	Description	Topology	Default value
`sitecore-identitysecret.txt`	Shared secret between the Identity Server and client roles Length: 64 characters	XM1, XP1	
`sitecore-reportingapikey.txt`	Symmetric key used to access the Sitecore XDB Processing Service Length: 64-128 characters	XP1	
`sitecore-solr-connection-string.txt`	Connection string to Solr	XM1, XP1	http://solr:8983/solr;solrCloud=true
`sitecore-solr-connection-string-xdb.txt`	Connection string to Solr-xdb If you have specified a custom prefix in the `sitecore-solr-core-prefix-name.txt` file, you must update the connection string with the same prefix.	XP1	http://solr:8983/solr/sitecore_xdb;solrCloud=true
`sitecore-solr-prefix-name.txt`	A common prefix for Solr core names. If you use the XP1 topology and change this value, you must update the solr xdb connection string in the `sitecore-solr-connection-string-xdb.txt` file with the same prefix.	XM1, XP1	sitecore
`sitecore-databaseusername.txt`	SQL Server administrator name	XM1, XP1	sa
`sitecore-databaseservername.txt`	Server name for connect to MS SQL Server	XM1, XP1	mssql
`sitecore-databasepassword.txt`	Password for connect to MS SQL Server	XM1, XP1	
`sitecore-database-elastic-pool-name.txt`	Database elastic pool name. Use the name of an elastic pool resource with access to the SQL Server	XM1, XP1	
`sitecore-core-database-username.txt`	UserName for database name *_Core in MS SQL Server	XM1, XP1	coreuser
`sitecore-core-database-password.txt`	Password for database name *_Core in MS SQL Server	XM1, XP1	
`sitecore-master-database-username.txt`	UserName for database name *_Master in MS SQL Server	XM1, XP1	masteruser
`sitecore-master-database-password.txt`	Password for database name *_Master in MS SQL Server	XM1, XP1	
`sitecore-web-database-username.txt`	UserName for database name *_Web in MS SQL Server	XM1, XP1	webuser
`sitecore-web-database-password.txt`	Password for database name *_Web in MS SQL Server	XM1, XP1	
`sitecore-forms-database-username.txt`	UserName for database name *_ExperienceForms in MS SQL Server	XM1, XP1	formsuser
`sitecore-forms-database-password.txt`	Password for database name *_ExperienceForms in MS SQL Server	XM1, XP1	

Name	Description	Topology	Default value
`sitecore-exm-master-database-username.txt`	UserName for database name *_EXM.Master in MS SQL Server	XP1	exmmasteruser
`sitecore-exm-master-database-password.txt`	Password for database name *_EXM.Master in MS SQL Server	XP1	
`sitecore-messaging-database-username.txt`	UserName for database name *_Messaging in MS SQL Server	XP1	messaginguser
`sitecore-messaging-database-password.txt`	Password for database name *_Messaging in MS SQL Server	XP1	
`sitecore-marketing-automation-database-username.txt`	UserName for database name *_MarketingAutomation in MS SQL Server	XP1	mauser
`sitecore-marketing-automation-database-password.txt`	Password for database name *_MarketingAutomation in MS SQL Server	XP1	
`sitecore-processing-engine-storage-database-username.txt`	UserName for database name *_ProcessingEngineStorage in MS SQL Server	XP1	processingenginestorageuser
`sitecore-processing-engine-storage-database-password.txt`	Password for database name *_ProcessingEngineStorage in MS SQL Server	XP1	
`sitecore-processing-engine-tasks-database-username.txt`	UserName for database name *_ProcessingEngineTasks in MS SQL Server	XP1	processingenginetasksuser
`sitecore-processing-engine-tasks-database-password.txt`	Password for database name *_ProcessingEngineTasks in MS SQL Server	XP1	
`sitecore-processing-pools-database-username.txt`	UserName for database name *_Processing.Pools in MS SQL Server	XP1	processingpoolsuser
`sitecore-processing-pools-database-password.txt`	Password for database name *_Processing.Pools in MS SQL Server	XP1	
`sitecore-processing-tasks-database-username.txt`	UserName for database name *_Processing.Tasks in MS SQL Server	XP1	processingtasksuser
`sitecore-processing-tasks-database-password.txt`	Password for database name *_Processing.Tasks in MS SQL Server	XP1	
`sitecore-reference-data-database-username.txt`	UserName for database name *_ReferenceData in MS SQL Server	XP1	refdatauser
`sitecore-reference-data-database-password.txt`	Password for database name *_ReferenceData in MS SQL Server	XP1	
`sitecore-reporting-database-username.txt`	UserName for database name *_Reporting in MS SQL Server	XP1	reportinguser
`sitecore-reporting-database-password.txt`	Password for database name *_Reporting in MS SQL Server	XP1	
`sitecore-collection-shardmapmanager-database-username.txt`	UserName for database name *_Xdb.Collection.ShardMapManager in MS SQL Server	XP1	shardmapmanageruser
`sitecore-collection-shardmapmanager-database-password.txt`	Password for database name *_Xdb.Collection.ShardMapManager in MS SQL Server	XP1	

Name	Description	Topology	Default value
`sitecore-media-request-protection-shared-secret.txt`	The Sitecore secret used to protect media requests. **IMPORTANT** You must change the shared secret to a random string. Do not use the default value.	XM1, XP1	HQ(NjM(u6_5koVla-cTf4ta8x1h6Sb+ZcUQrULUz-0Afpx0cx-NuMtIoQkpDFmX5

4.6. Solr-init image variables

Before you deploy the Solr-init image you can set parameters in the `\k8s-sitecore-xm1\init\solr-init.yaml` file to control, for example, how many replicas and shards the deployment creates.

This table shows some of the parameters you can use.

Name	Default	Description
`SOLR_SITECORE_CONFIGSET_SUFFIX_NAME`	*_config*	Solr config set suffix. Defines the name of the configurations to use for this collection. If not provided, Solr defaults to the collection name as the configuration name.
`SOLR_REPLICATION_FACTOR`	*1*	The number of replicas created for each shard.
`SOLR_NUMBER_OF_SHARDS`	*1*	The number of shards to be created as part of the collection. This is a required parameter if you use the **compositeId** router.
`SOLR_MAX_SHARDS_NUMBER_PER_NODES`	*1*	When you create a collection, Solr spreads the shards and/or replicas across all available live nodes. If a node is not live when the CREATE operation is called, it does not get any parts of the new collection, which might lead to too many replicas being created on a single live node. Setting this variable sets a limit on the number of replicas CREATE spreads to each node. If Solr cannot fit the entire collection into the live nodes, no collection is created at all. A node never contains two replicas of the same shard.
`SOLR_CORE_PREFIX_NAME`	*sitecore*	Core prefix
`SOLR_COLLECTIONS_TO_DEPLOY`	<empty>	List names of the collections to deploy to Solr separated by a comma (,).

For a description of all the parameters you can use, see the Solr Reference Guide 6.6 and Solr Reference Guide 7.7.

4.7. Common issues

This section contains solutions to issues you might encounter.

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

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

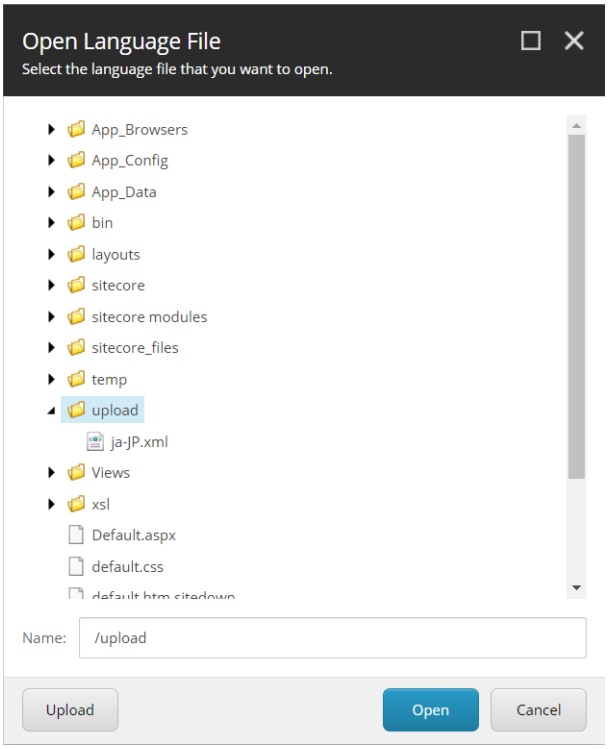
```
ERROR Could not save posted file: ja-JP.xml
Exception: System.UnauthorizedAccessException
Message: Access to the path 'C:\inetpub\wwwroot\ja-JP.xml' is denied.
```

This happens because the **Import language** dialog uploads translation files to the `website` root folder by default, and for security reasons write access is denied for this folder.

To solve this issue, upload the translation files to the `upload` folder, where write access is enabled.

To upload a translations file:

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



2. In the **Upload Files** dialog, you can now upload the translations file to the `upload` folder.
3. Use the **Import language** dialog to import the translation file from the `upload` folder.

> **NOTE**
> The translation .xml file is saved to the Media library. You can delete it after you have imported the translation.

4.7.2. Only the main Sitecore log is exposed for Sitecore roles containers

Sitecore uses the LogMonitor tool to collect and output log files for containers. By default, the tool is configured to monitor the following log files:

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

Auxiliary Sitecore logs, such as for search, crawling, or publishing, are not monitored on Sitecore containers.

To reconfigure the LogMonitor tool to include logs for other roles:

1. Create a Dockerfile with the corresponding role image.
2. In the `C:\LogMonitor\LogMonitorConfig.json` file, change the filter for the source from the `c:\\inetpub\\wwwroot\\App_data\\logs` directory. For example:

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

3. Use the Dockerfile to build a new Docker image for the role.

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

4.7.3. When I request SSC, problems occur if there are underscores in header names

For example, the following GET request results in a `bad request` error:

- url: https://cm.globalhost/sitecore/api/ssc/aggregate/content/Items
- add the header:
- key: sc_apikey
- value: id of the created item

This occurs because Nginx does not allow underscores in header names.

You can solve this problem in two ways:

- Add `sc_apikey` as a parameter to every request. For example:

```
https://cm.globalhost/sitecore/api/ssc/aggregate/content/Items?sc_apikey=336A2458-1E02-412E-AA1B-E84E002264FC
```

- Allow underscores in header names by introducing a ConfigMap for the ingress configuration. For more information, see the Kubernetes ConfigMap documentation.

4.7.4. When I reference a CM service from my container, the connection fails

Use a fully qualified domain name instead of a service name. For example, if the CM service is in the `default` namespace, and the domain name for your cluster is `cluster.local`, use the service's DNS name `cm.default.svc.cluster.local`.