Installation Guide for Production Environments with Kubernetes

Sitecore XP 10.0.1

How to install Sitecore XP with Kubernetes

Table of Contents

Chapter 1

Introduction

Sitecore Experience Platform 10.0.1 uses Kubernetes (K8s) as the default orchestrator for deploying production environments. The Sitecore XP Kubernetes specification files that are used to map the minimum required configuration parameters between the Sitecore software containers are provided as a reference.

Sitecore customers are expected to extend these specifications to support their own requirements. It is the responsibility of each Sitecore customer to ensure that their production deployments meet the standards for stability and security set by their organization.

This guide describes how to deploy the Sitecore Experience Platform 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 Sitecore XP for Kubernetes specification files are designed to avoid Azure specific dependencies where possible.

1.1 Sitecore Kubernetes specifications

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

You use a 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.

1.1.1 Sitecore 10.0.1 Container Deployment Package

The Sitecore XP 10.0.1 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 the section *Topologies*.

1.1.2 Sitecore Container Registry

The Sitecore XP 10.0.1 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 is hosted at scr.sitecore.com and supports the Docker content trust model that lets you pull signed images.

1.1.3 Client Software Requirements

- Operating System
  - Windows 10 1809 or later

    or
  - Windows Server 1809 or later
- Kubectl 1.16x or later – use the latest stable non-preview version
  - To get a list of the supported locations run the following command:

    ```
    az account list-locations
    ```
  - To get the latest stable version with the desired region (location) run the following command:

    ```
    az aks get-versions --location <location> --output table
    ```
- Helm 3.0.x or later

  This is only required for ingress controller deployments. We recommend that you use choco to install Helm.

  ```
  choco install kubernetes-helm
  ```
- Azure CLI 2.8.0 or later is required for AKS deployments.

- Sitecore SXP 10.0.1 Container Deployment Package

  You must extract the *k8s-sitecore-** configuration folder for the topology that you want to deploy.

1.1.4 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

1.1.5 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.

1.1.6 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 5.0 or higher

1.1.7 Azure Kubernetes Service Requirements

- AKS cluster configured with the latest stable release of Kubernetes – version 1.16.x or later
- One Windows Server 2019 version 1809 OS node.
- For non-production environments and testing, the recommended minimum VM size is Standard_D4_v3.
- For production environments, the VM size and number of nodes for depends on your individual requirements.

To get the latest version of Kubernetes supported by AKS, substitute the location parameter with the desired region, and then run the following Azure CLI command:

```
az aks get-versions --location eastus --output table
```

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

1.1.8 Ingress Controller Requirements

A Kubernetes Ingress Controller is required to deploy the Sitecore Kubernetes specification files.

Sitecore Kubernetes deployments are tested with latest stable NGINX Ingress Controller releases from the Helm project and are subject to change over time.

The latest stable release from the Helm project for NGINX Ingress Controller at the time of publishing is

- Chart version 1.41.1
- App version 0.34.1

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 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 section *Ingress Controller service*.

1.2 Prerequisites

Before you can deploy the Sitecore XP containers, you must prepare the Kubernetes specification files and the supporting files.

You must prepare the:

- Kubernetes specification files.
- Kubernetes secrets.
- Sitecore license file.
- Sitecore Identity Server token signing certificate.
- TLS/HTTPS certificates.
- Plan for the external data services.
- Plan for the Ingress Controller service.

1.2.1 Kubernetes specification files

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

You can download the Sitecore Experience Platform 10.0.1 Container Deployment Package archive (.zip) from dev.sitecore.net. Extract the archive and locate the `k8s-sitecore-*` folder for the Sitecore topology that you want to deploy.

We currently support the following topologies:

- Sitecore XP Server (XP1)
- Sitecore XM Server (XM1)

Inspect the folder contents and the Kubernetes specification files (`.yaml`).

For more information about the supported topologies, see the section *Topologies*.

1.2.2 Kubernetes secrets

Sitecore Kubernetes deployments use secrets to securely store the strings that are used by the containers in the cluster.

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`, `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.

For a complete list of secrets and more information about the individual secrets, see Appendix B.

We provide a Kubectl `kustomization.yaml` file that deploys all the secret names and values in a single command.

1.2.3 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.

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

Appendix A contains a sample PowerShell script that converts a license file into a Base64 compressed string for use in a K8s secret.

Note
Some Sitecore license files are so large that they are incompatible with containers even after compression. This usually happens when additional HTML is embedded in the license file. To obtain a license file for use with containers, contact your Sitecore partner. 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 about how to mount a license file as a volume, see the Sitecore Docker Demo repository on GitHub.

1.2.4 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.

Appendix C 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 certificate password to the `sitecore-identitycertificatepassword.txt` Kubernetes secret text file.

Like the Sitecore license 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.

1.2.5 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 NIGNX TLS user guide.

The HTTPS protocol is required to support the secure browser cookies used by the Sitecore Content Management role and the Identity Server role. 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.

Appendix D contains a sample script that generates the required certificates.

Important
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 Appendix D uses the *mkcert* tool to automatically create the self-signed root authority certificate and install it in the correct certificate store.

1.2.6 Production and non-production container images

To minimize the time it takes to deploy Sitecore XP to Kubernetes clusters for non-production use, Sitecore provides container images for the required external services. The container images for the external services are for *non-production* use only. The external services images are not supported by Sitecore in a production environment. The non-production services do not follow the best practices that are recommended for hosting a production environment and should not be considered as a source for production environments.

Sitecore provides non-production images for Microsoft SQL Server, SolrCloud, and RedisLabs Redis services.

When you delete a non-production external service deployment from the default configuration, the Sitecore non-production images remove all the data. If you want the data to persist across multiple deployments, we recommend that you configure persistent volumes in your configuration or move all the required services outside the container environment. You can skip the Kubernetes specification files for deployments that you moved to use external services.

Every container image that has the *type=nonproduction* label is not supported in a production environment. No warranty or extended support is provided for images that are labelled for non-production.

1.2.7 External data services

In production deployments, customers are expected to 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 all the required data, we provide Kubernetes *data initialization* jobs for Microsoft SQL Server and SolrCloud.

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 section Deploy Data Initialization Jobs.

1.2.8 Ingress Controller service

A Kubernetes Ingress Controller is required 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 setting for *externalTrafficPolicy* to *Local*.

For more information about how to preserve the client source IP address, 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, Sitecore recommends 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.

1.2.9 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*:

```
      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.

1.3 Topologies

Sitecore XP 10.0.1 supports the following topologies for use with Kubernetes:

XM Server (XM Scaled)

The Sitecore Experience Manager Server for Kubernetes is suitable for use in production and non-production environments.

The XM Server (XM Scaled) topology supports the following Sitecore roles:

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

XP Server (XP Scaled)

Sitecore Experience Platform Server for Kubernetes is suitable for use in production and nonproduction environments. The resources required to run XP Server in nonproduction can be significant but are required to mimic the exact configuration that is used in production.

The XP Server (XP Scaled) topology supports the following Sitecore roles:

Role Type	Sitecore Role
Production roles	Content Management
	Content Delivery
	Sitecore Identity Server
	XDB Processing
	XDB Reporting service
	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
	Marketing Automation Engine

Role Type	Sitecore Role
	Sitecore Cortex Processing
	MSSQL Data Initialization Job
	SolrCloud Data Initialization Job
Non-production roles	Microsoft SQL Server
	SolrCloud
	RedisLabs Redis Server

Chapter 2

Deploying Sitecore XP to the Azure Kubernetes Service

To deploy the Sitecore XP containers to a Kubernetes cluster, you must use the Kubectl CLI.

This chapter describes how to deploy Sitecore Experience Platform to the Azure Kubernetes Service.

This chapter contains the following sections:

- Deploy External Data Services
- Deploy Sitecore XP to AKS
- Post-deployment steps

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 Nonproduction*.

2.1.1 Application Database User Credentials

Sitecore Kubernetes specification files are designed to deploy the 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.

For a complete list of application database user credentials secrets, see Appendix B.

2.2 Deploy Sitecore XP to the Azure Kubernetes Service

To deploy Sitecore XP to the Azure Kubernetes Service, you must:

Create an AKS cluster

To create a new Azure Kubernetes Service (AKS) cluster with a Windows Server 2019 node pool, you can 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 a AKS cluster, see the Azure AKS documentation.

Configure the Kubectl context cluster

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

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

2. Get the credentials for the K8s cluster that were created with the AKS cluster.

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

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.

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```

3. Use Helm to deploy the NGINX ingress controller.

```
helm install nginx-ingress stable/nginx-ingress --set controller.replicaCount=1 --set
controller.nodeSelector."beta\.kubernetes\.io/os"=linux --set
defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux --set-string
controller.config.proxy-body-size=10m --set
controller.service.externalTrafficPolicy=Local
```

   For more information about ingress configuration, see NGINX Ingress Controller Configuration.

Deploy the secrets

1. Ensure that all the secrets files (`.txt`, `.crt`, `.key`) files in the `./secrets` folder are updated according to the requirements listed in Appendix B.
2. From the root folder of the desired topology, run the following command:

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

Deploy External Services for a non-production deployment

1. From the root folder of the desired topology, run the following command:

```
kubectl apply -f ./external/
```

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

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

3. Wait until the status of all the pods is `Running/OK`.

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

kubectl wait --for=condition=Ready pods --all
```

Deploy the data initialization jobs

1. From the root folder of the desired topology, run the following command:

```
kubectl apply -f ./init/
```

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

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

3. Wait until the status of all the jobs is `Complete/OK`.

```
kubectl wait --for=condition=Complete job.batch/solr-init --timeout=600s

kubectl wait --for=condition=Complete job.batch/mssql-init --timeout=600s
```

Deploy the Sitecore pods

1. From the root folder of the desired topology, run the following command:

```
kubectl apply -f ./ -f ./ingress-nginx/ingress.yaml
```

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

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

3. Wait until the status of all the pods is `Running/OK`.

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

Update the local host file

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

```
kubectl get service -l app=nginx-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

2.3 Post-deployment steps

When the deployment is finished, you must configure the SolrCloud search indexes.

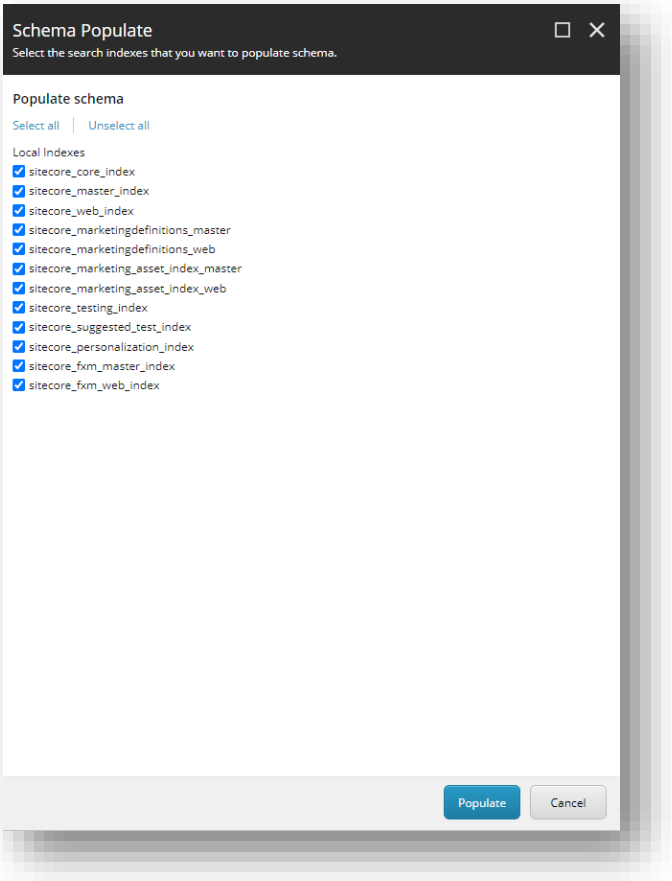
2.3.1 Configure the SolrCloud search indexes

To configure the SolrCloud search indexes:

1. Open a browser and navigate to:

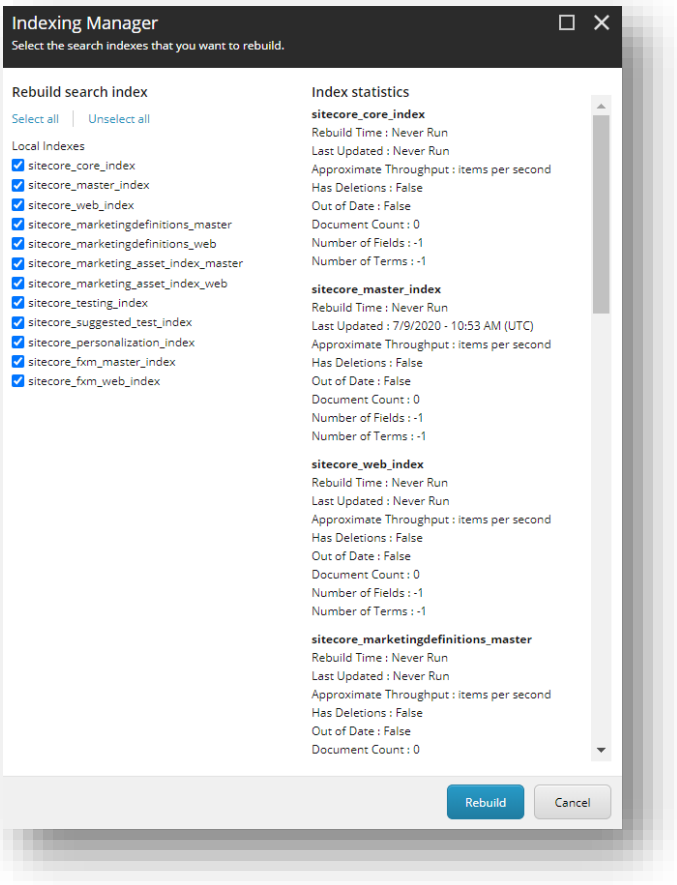
   https://cm.globalhost/sitecore

2. Login to Sitecore with the *admin* user and password that you configured as a secret.
3. In the Sitecore Control Panel click **Populate Managed Schema,** and in the **Schema Populate** dialog box, select all the indexes and then click **Populate.**



Wait for the process to complete and then close the dialog box.

4. In the Sitecore Control Panel click **Indexing manager** and in the **Indexing Manager** dialog **box,** select all the indexes that you want to rebuild and then click **Rebuild.**



Wait for the process to complete and then close the dialog box.

Chapter 3

Appendices

3.1 Appendix A – The License file compression and encoding PowerShell helper function

```
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
}
ConvertTo-CompressedBase64String -Path .\license.xml | Out-File -Encoding ascii -NoNewline -
Confirm -FilePath .\secrets\sitecore-license.txt
```

3.2 Appendix B – The Kubernetes secrets list

Name	Description	Topology	Default value
sitecore-license.txt	License file content converted to GZIP Compressed and Base64 encoded string (See Appendix A)	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 Appendix C)	XM1, XP1	
sitecore-identitycertificatepassword.txt	Password to open the Identity Server certificate	XM1, XP1	
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 Reporting WebAPI 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. **Note:** in case of using a custom *Sitecore solr core prefix* specified in the *sitecore-solr-core-prefix-name.txt*, current connection string should be updated with that prefix.	XP1	http://solr:8983/solr/sitecore_xdb;solrCloud=true
sitecore-solr-core-prefix-name.txt	A common prefix for solr cores names. **Note:** in case of XP1 topology, if you change the value of current secret you need to update the *solr xdb connection string* in the *sitecore-solr-connection-string-xdb.txt* with this prefix.	XM1, XP1	sitecore

sitecore-databaseusername.txt	SQL Server administrator password	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. Fill it with the name of an elastic pool resource which has 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	
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	

3.3 Appendix C – Create the Identity Server token signing certificate

To create the Identity Server token signing certificate, run the following PowerShell script:

```
$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
```

3.4 Appendix D – 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 as an Administrator.
2. In the same directory as the Kubernetes specification files for the desired topology, execute the following commands:

```
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"
```

   The *mkcert* utility will prompt the user the first time to install the generated self-signed root certificate authority.

3.5 Appendix E – Common issues

I cannot upload a Translations file to the website root folder

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

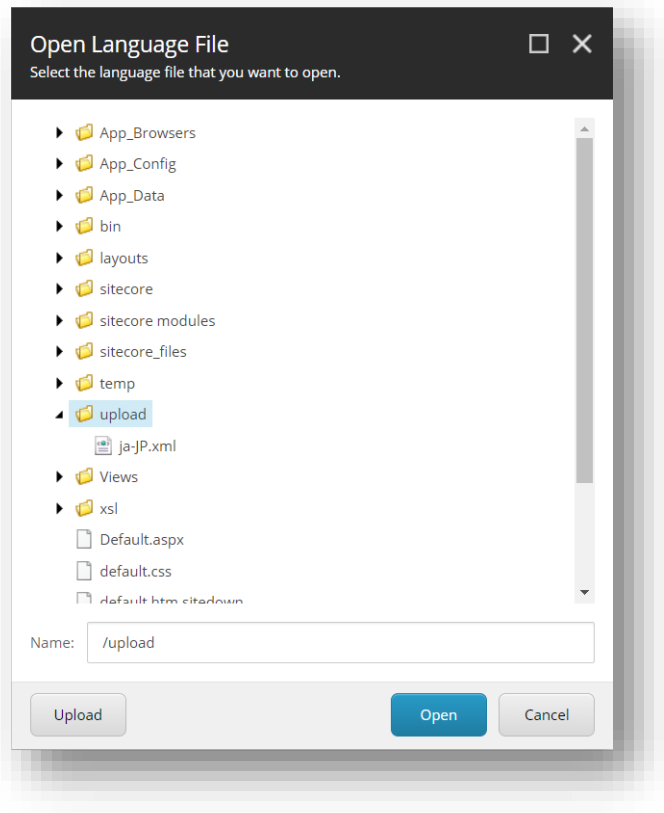
```
ERROR Could not save posted file: ja-JP.xml
Exception: System.UnauthorizedAccessException
Message: Access to the path 'C:\inetpub\wwwroot\ja-JP.xml' is denied.
```

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

To work around this issue, upload the translations files to the `\upload\` folder. *Write* access is enable for this folder.

To upload a translations file:

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



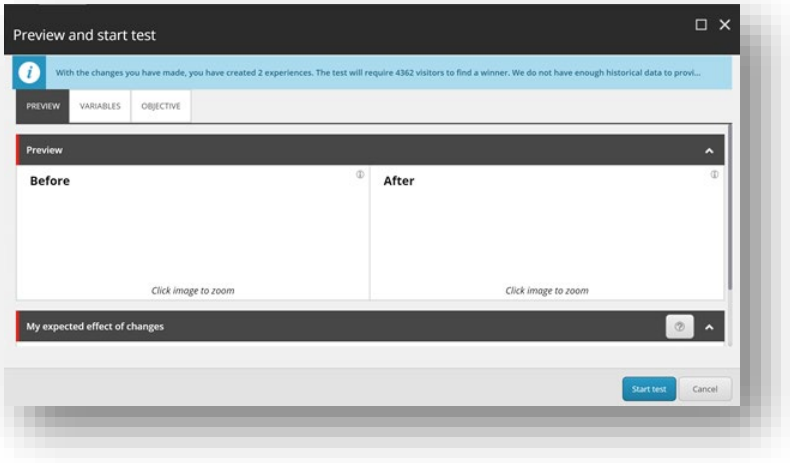
2. In the **Upload Files** dialog box, you can now upload the translations file to the `\upload\` folder.

You can now import translations file from this folder.

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

Screenshots are not generated in the content testing windows

On XP1 Kubernetes deployments where global DNS names are not configured, you may see blank screenshots in the **Preview and Start test** dialog or in the **Test result** dialog:



This occurs because the `WebUtil.GetServerUrl()` metod returns the outer instance address (`http://cm.globalhost`) while a valid host name is expected. As a result, the hostname is resolved incorrectly.

As a workaround, you can add instructions to the container lifecycle ***PostStart*** hook which adds a record to the hosts file when you start the pod as shown in the following example from the `cm.yaml` file for the CM role:

```
containers:
 - name: sitecore-xp1-cm
   image: {registry}/{project}/sitecore-xp1-cm:{version}
   ports:
   - containerPort: 80
   lifecycle:
     postStart:
       exec:
         command: ["powershell","-Command","Add-Content C:/Windows/System32/drivers/etc/hosts
'127.0.0.1 cm.globalhost'"]
```

Only the main Sitecore log is exposed for Sitecore roles containers

The LogMonitor tool is used to collect and output log files for containers. It 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 search, crawling or publishing, and so on, 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:

In the `C:\LogMonitor\LogMonitorConfig.json` file, change the `filter` for the source from the "`c:\\inetpub\\wwwroot\\App_data\\logs`" directory:

```
{
    "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.

When requesting SSC, problems occur if there are underscores in header names

For example, the following GET request:

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

results in a *bad request*.

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

http://nginx.org/en/docs/http/ngx_http_core_module.html#underscores_in_headers,

You can solve problem this by adding `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
```

You can also solve this problem by allowing underscores to be used in header names.

To allow underscores, you must introduce a configMap for the ingress configuration, because the setting is not available in annotations.

For more information, see https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/#enable-underscores-in-headers