XC Installation Guide for Production Deployments with Kubernetes

Sitecore Experience Commerce 10.0

October 14, 2021

Table of Contents

1. Introduction

Introduction to using Kubernetes to orchestrate your Sitecore Experience Commerce deployment to production.

Sitecore Experience Commerce (XC) 10.0.0 uses Kubernetes (K8s) as the default orchestrator for deploying to production. The Sitecore XC Kubernetes specification files that are used to map the minimum required configuration parameters between the Sitecore XC software containers are provided as a reference.

IMPORTANT
Sitecore customers are expected to extend these specifications to support their own production deployment 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 Sitecore Experience Commerce (XC) 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 XC for Kubernetes specification files are designed to avoid Azure specific dependencies where possible.

2. Sitecore XC Kubernetes specifications

Sitecore provides Kubernetes specification files (`.yaml`) that you can 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 or you can use the standalone version.

2.1. Sitecore Commerce Container SDK

The Sitecore Commerce Container SDK package contains Kubernetes specification files for deploying a reference Sitecore XC software cluster solution. You can download the Sitecore Commerce Container SDK package (`Sitecore.Commerce.Container.SDK.*.*.*.zip`) from the Sitecore Downloads site.

IMPORTANT
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.

For a description of other topologies supported in the Sitecore Commerce Container SDK, see the section Topologies.

2.2. Sitecore container registry

The Sitecore container images are hosted in a public Docker container registry, and available to all customers 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.

2.3. Client software requirements

The following client software is required for deployment product deployment with Kubernetes:

- Operating system:
  - Windows 10 1809 or later
    or
  - Windows Server 1809 or later
- Azure Az CLI 2.8.0 or later

  > **NOTE**
  > (Only required for AKS deployments)

- Windows PowerShell 5.0
- Kubectl 1.18.2 or later or later
  - 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 3.1.2 or later

  > **NOTE**
  > Only required for ingress controller deployments.

- Sitecore Commerce Container SDK

2.4. Kubernetes cluster software requirements

The following software is required for Kubernetes cluster:

- Kubernetes 1.18.2 or later
- Windows Server 2019 version 1809

2.5. Kubernetes cluster hardware requirements

Kubernetes cluster requires the following hardware:

IMPORTANT
Operational requirements depend on service use.

- RAM
  We recommend a minimum of 16 GB RAM per Kubernetes cluster during startup.
- CPU
  We recommend a quad core or higher per Kubernetes node during startup.
- Disk
  We recommend premium SSD disks for optimal performance when downloading and running Docker containers.

2.6. Required external data services

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

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

- 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

2.7. Azure Kubernetes service requirements

- AKS cluster configured with the latest stable release of Kubernetes–version 1.16.x or later
- One 1 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.

2.8. 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.

3. Prerequisites

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

You must:

- Prepare Kubernetes specification files
- Generate the Identity Server token signing certificate
- Set up Kubernetes secrets and YAML files
- Generate TLS/HTTPS certificates
- Review considerations about production and non-production images
- Deploy external data services

3.1. Prepare Kubernetes specification files

The Sitecore Experience Commerce (XC) Kubernetes specification files are designed to be deployed using the Kubernetes Kubectl CLI.

To prepare Kubernetes specification files:

1. Download the Sitecore Commerce Container SDK package (`Sitecore.Commerce.Container.SDK.*.*.*.zip`)
2. Extract the archive and locate the *k8s-commerce-** folder for the Sitecore topology that you want to use.

   NOTE
   Sitecore XC Server (XC1) is the only topology supported for Kubernetes deployments.

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

3.2. Generate 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, encode it to a Base64 encoded string form, and pass it to the container as an environment variable.

To generate a self-signed certificate:

- Run the following sample 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
```

> **NOTE**
> The sample script generates a new self-signed certificate, prepares the string that is used as a secret, and saves it to a `SitecoreIdentityTokenSigning.txt` file.
>
> The `SitecoreIdentityTokenSigning.txt` file contains the certificate string you specify as the `–idCert` value when you prepare the YAML files.

3.3. Set up 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 usenames, passwords, and TLS certificates and are configured in text files and certificate files (`tls.crt`, `tls.key`) in the `./secrets/` folder in the Kubernetes specification files for each topology.

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

You must update each secret text file (`.txt`, `.crt`, `.key`) with the required values before deploying additional resources to the K8s cluster.

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

The `Sitecore.Commerce.Container.SDK` package provides a Kubectl `kustomization.yaml` file that deploys all the secret names and values in a single command.

3.3.1. Prepare YAML files

You use the `UpdateK8SYaml.ps1` script to prepare the YAML files.

To prepare the YAML files:

- Open a PowerShell window, and run the following script:

```
.\UpdateK8SYaml.ps1  -jsonFile '.\configltsc2019.json' `
  -k8sRootPath '..' `
  -licenseFilePath "Location of your licence file" `
  -braintreeEnvironment "sandbox" `
  -braintreeMerchantId "Your merchant id" `
  -braintreePublicKey "Your public key" `
  -braintreePrivateKey "Your private key" `
```

```
-telerikKey "Your Telerik Encryption Key" `
-idCert "Your Sitecore Identity  certificate" `
-idSecret "Your Sitecore Identity secret" `
-idPassword "Your Sitecore Identity password" `
-xcIdSecret "Your XC Connect Client Secret" `
-reportingApiKey "Your Sitecore Reporting API key"
```

3.4. Generate 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 and the Kubernetes ingress controller.

The default Kubernetes ingress controller used by XC 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 containers inside the cluster. For more information, see the Kubernetes documentation for ingress TLS configuration and the 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 remove it if it is not required for your specific use case.

The following sample script generates the required certificates. This script is also available in the Sitecore Commerce Container SDK package.

To generate TLS/SSL certificates that are required by the NGINX ingress controller and install them in the correct certificate store:

1. Open a Windows Command Prompt as Administrator.
2. In the same directory as the Kubernetes specification files, for example in the `k8s-commerce-xc1` folder, run the following sample script:

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

```
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

.\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"
```

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

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

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


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

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

IMPORTANT

Once the self-signed root authority certificate and per-host TLS/SSL certificates have been generated, you must install the root authority certificate on the Trusted Root Certificate Authority store on all the clients.

3.5. About production and non-production containers images

To minimize the time it takes to deploy Sitecore Experience Commerce (XC) to Kubernetes clusters for non-production use, Sitecore provides the required external services. These are for non-production use only. The non-production images are not supported by Sitecore in a production environment.

IMPORTANT

The non-production services do not follow the best practices for hosting a production environment and should not be considered as a basis for production environments.

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

WARNING

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.

3.6. Deploy external data services

In production deployments, customers are expected to host the required external services outside the Kubernetes cluster.

External hosted services for Microsoft SQL Server, Apache Solr 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 XC 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 Apache Solr.

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.

4. Topology

Sitecore XC supports a scaled topology for use with Kubernetes.

Sitecore Experience Commerce for Kubernetes is suitable for use in production and non-production environments. For non-production environments, the `Sitecore.Commerce.Container.SDK` package includes samples that you can use for external services roles. The resources required to run Sitecore XC in a non-production environment can be significant, but are required to mimic the exact configuration that is used in production.

Sitecore roles in the scaled topology for use with Kubernetes

The Sitecore XC for Kubernetes topology includes container images for the following Sitecore roles:

NOTE
While the container images of Sitecore roles are considered production images, you must create your own set of container images from your customized Commerce solution.

- 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
- Sitecore Cortex Processing
- MSSQL Data Initialization Job
- Solr Data Initialization Job

- Commerce Business Tools
- Commerce Commerce Ops Role
- Commerce Authoring Role
- Commerce Shops Role
- Commerce Minions Role

External service roles (non-production)

The Sitecore XC for Kubernetes topology includes the following sample container images of external services to facilitate deployment in a non-production environment:

NOTE
The container images of external services are not intended nor suitable for production deployment.

- Microsoft SQL Server
- Apache Solr
- RedisLabs Redis Server

5. Deploying Sitecore XC to the Azure Kubernetes Service

You use Kubectl CLI to deploy the Sitecore XC containers to a Kubernetes cluster.

To deploy Sitecore XC to the Azure Kubernetes Service, perform the following:

- Create a resource group
- Configure the Kubectl context cluster
- Deploy an ingress controller
- Deploy the secrets
- Deploy External Services for a non-production deployment
- Deploy the data initialization jobs
- Deploy the Sitecore XC pods
- Update the local host file
- Validate access to the Commerce Authoring environment instance

5.1. Create a resource group

To create a new resource group to use with the AKS cluster, you must use the Azure command-line interface (Azure CLI) that contains one or more Windows Server 2019 version 1809 nodes.

NOTE
You can use your existing AKS cluster and resource group.

For more information about using the Azure CLI, see the Azure documentation.

To create a resource group:

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

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

2. Create a resource group:

```
az group create --name <MY_RESOURCE_GROUP_NAME> --location <LOCATION>
```

5.2. Configure the Kubectl context cluster

This procedure assumes you completed the creation of a resource group.

To configure the Kubectl context cluster:

- Create a cluster and a node pool for Windows.

```
az aks create --resource-group <RESOURCE_GROUP_NAME> --name <CLUSTER_NAME> `

        --node-count 1 --enable-addons monitoring
--kubernetes-version <K8S_VERSION> --generate-ssh-keys `

        --windows-admin-password <PASSWORD> --windows-admin-username azureuser --vm-set-type
VirtualMachineScaleSets --load-balancer-sku standard --network-plugin azure  --node-vm-size
<LINUX_VM_SIZE>

az aks nodepool add --resource-group <RESOURCE_GROUP_NAME> --cluster-name <CLUSTER_NAME> --
os-type Windows
--name win --node-count 1 --node-vm-size <WINDOWS_VM_SIZE> --kubernetes-version <K8S_VERSION>

az aks get-credentials
--resource-group <RESOURCE_GROUP_NAME> --name <CLUSTER_NAME>
```

5.3. 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.

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

3. Use Helm to deploy the NGINX ingress controller.

   ```
   helm install nginx-ingress stable/nginx-ingress
    --set controller.replicaCount=2
    --set controller.nodeSelector."kubernetes\.io/os"=linux
    --set defaultBackend.nodeSelector."kubernetes\.io/os"=linux
    --set-string controller.config.proxy-body-size=1000m
    --set controller.service.externalTrafficPolicy=Local
    --set controller.admissionWebhooks.patch.nodeSelector."kubernetes\.io/os"=linux
   ```

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

5.4. Deploy the secrets

To deploy the secrets to the secret files:

- Ensure that all the secrets files (.txt, .crt, .key) files in the `.\secrets` folder are updated according to the requirements listed in Appendix A.

```
kubectl apply -k .\secrets
```

5.5. Deploy External Services for a non-production deployment

To deploy the external services:

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
```

5.6. Deploy the data initialization jobs

To 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
```

5.7. Deploy the Sitecore XC pods

To deploy the Sitecore XC pod:

1. From the root folder of the topology, for example from the `k8s-commerce-xc1` folder, run the following command:

   ```
   kubectl apply -f .\
   ```

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
   ```

4. From the root folder, run the following command:

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

5. From the root folder, run the following command:

   ```
   kubectl apply -f .\ingress-nginx\configuration.yaml
   ```

5.8. 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 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
   - authoring.globalhost
   - shops.globalhost
   - minions.globalhost

- ops.globalhost
- bizfx.globalhost

5.9. Validate access to the Commerce Authoring environment instance

To validate the deployment of the Commerce Authoring environment instance:

- Open a browser, and enter the URL for the Commerce Engine instance running the Commerce Authoring service. The default hostname for the Commerce Authoring is: `https://authoring.globalhost/commerceops/$metadata`.

6. Post-deployment tasks

Once you have confirmed that the status of all containers is healthy, you must perform the following tasks to complete your deployment:

- Bootstrap and initialize the Commerce Engine
- Validate the deployment of the Business Tools
- Configure user accounts
- Generate catalog templates
- Create an SXA Storefront tenant and site (for XC1-CXA topology only)
- Configure SolrCloud search indexes
- Perform full rebuild of Commerce indexes

6.1. Bootstrap and initialize the Commerce Engine

After you have confirmed that containers have a healthy status, you must bootstrap and initialize your Commerce environments.

The Sitecore Commerce Engine SDK includes samples of API calls for DevOps operations, so that you can access the Sitecore XC API directly. The following instructions assume that you are using Postman to exercise the Sitecore XC API.

NOTE
The following instructions assume that you have access to a Sitecore XC development (or DevOps) environment, with the Postman API samplesdeployed. The Postman samples are included as part of the Sitecore Commerce Engine SDK, available for download in the *Sitecore XC Packages for On Premise WDP* package.

6.1.1. Setup the environment in Postman

You must setup the environment in Postman to point to your deployment before you can exercise the API samples.

To setup the environment in Postman, for example, the *Habitat Environment*:

1. In the top right corner of Postman, in the environment selector drop-down, click the environment name, for example *Habitat Environment.*

2. Click the **Environment quick look** icon, and in the Environment values dialog, click **Edit**. The following table shows the default values from the `Sitecore.Commerce.Container.SDK/k8s-commerce-xc1/ingress-nginx/ingress.yaml`)

Variable	Current value
Environment	HabitatAuthoring
ShopperId	ShopperId
Language	en-US
Currency	USD
ServiceHost	https://authoring.globalhost
OpsApiHost	https://ops.globalhost
AuthoringHost	https://authoring.globalhost
MinionsHost	https://minions.globalhost
ShopsHost	https://shops.globalhost
SitecoreIdServerHost	https://id.globalhost
HostName	<deployment-name>
SitecoreIdServerPassword	<Password>

NOTE
This values should match the value used in the `k8s-commerce-xc1\secrets\xp\sitecore-adminpassword.txt` file).

3. In the **MANAGE ENVIRONMENTS** dialog, update the values to match those defined in the `/k8s-commerce-xc1/ingress-nginx` topology folder.

6.1.2. Bootstrap and initialize the Commerce Engine

To run the bootstrap and initialize operations:

1. In the top right corner of Postman, click the environment selector and select the environment, for example the *Habitat Environment*.
2. In the Postman **Collections** pane, open the *Authentication* folder, and in the *Sitecore* sub-folder, execute the `GetToken` request.
3. In the Postman **Collections** pane, navigate to the *SitecoreCommerce_DevOps* folder.
4. Open the *1 Environment Bootstrap* folder, and execute the **Bootstrap Sitecore Commerce** call.
5. Open the *3 Environment Initialize* folder, and execute the **Ensure\Sync default content paths** call.

   NOTE

   If the `status` of a request is `WaitingForActivation`, you can execute the Check Long Running Command Status request. When you execute the `CheckCommandStatus` request, you must ensure you are calling the same service that the previous command was executed in.

6. In the *3 Environment Initialize* folder, execute the **Initialize Environment** call.

   NOTE

   If the `status` of a request is `WaitingForActivation`, you can execute the Check Long Running Command Status request. When you execute the `CheckCommandStatus` request, you must ensure you are calling the same service that the previous command was executed in.

7. Repeat step 5 and step 6 for other environments, if applicable (for example, for the AdventureWorks environment).

6.2. Validate the deployment

After bootstrapping and initializing the Commerce Engine, make sure that you can access the Business Tools, and the Sitecore Launchpad.

6.2.1. Validate the deployment of Business Tools

The Sitecore Commerce XC Business Tools are deployed in the Authoring environment.

To validate the deployment of the Business Tools:

1. Open a browser, and enter the URL for the Commerce Business tools instance. The default host name for the XC Business Tools is: `https://bizfx.globalhost`.
2. Login to the Business Tools and ensure that you can browse the tools.

> **NOTE**
> Within the Sitecore Launchpad, the links to the Business Tools will be broken when you bring up the containers. To fix it, follow these instructionsand, in the **Link** field, enter the URL `https://bizfx.globalhost/` (instead of `https://localhost:4200`).

6.2.2. Validate access to the Content Management instance

To validate the deployment of the Content Management instance:

1. Open a browser, and enter the URL for the Content Management instance. The Content Management instance runs on port 443 and uses the HTTPS protocol.
   The default host name for the Content Management instance is: `https://cm.globalhost/Sitecore`
2. Validate that you can login to Sitecore and access the Sitecore Launchpad.

6.3. Configure user accounts

After you have deployed your Sitecore XC solution, you must create user accounts and assign the appropriate roles.

> **NOTE**
> Every Sitecore XC user who requires access to the Business Tools must have the *Commerce Business User* role assigned, at a minimum.

You create users and assign roles using the **User Manager** tool on the **Sitecore Launchpad**.

Refer to the User roles and permissions topic for information on the pre-defined roles and associated permissions for the Sitecore XC Business Tools.

6.4. Generate catalog templates

After you have deployed your Sitecore XC solution, you must refresh the cache and generate catalog templates, then republish the site.

You can perform both of these operations from the **Content Editor** on the **Sitecore Launchpad**.

To generate catalog templates:

1. Open a browser, and login to the **Sitecore Launchpad** (in a container deployment, the URL is `https://cm.globalhost/sitecore`, or in an Azure deployment, the URL is `https://<deployment name>-cm.azurewebsites.net/sitecore`).
2. Click on **Content Editor**.
3. In the Content Editor, click on the **Commerce** tab.
4. Click on **Refresh Commerce Cache** (in the **Caches** tile).
5. Click on **Update Data Templates** (in the **Catalog** tile).

6.5. Create an SXA Storefront tenant and site

If your deployment topology includes the SXA Storefront and you to want to use the SXA Storefront site as a starting point to create your own e-commerce site, you must create a new tenant and storefront site using this procedure..

> **NOTE**
> In deployment using Kubernetes, the SXA Storefront site is functional only after you complete all post-deployment steps.

6.6. Configure SolrCloud search indexes

The Sitecore XC deployment with Kubernetes runs a solr-init container for deploying the SolrCloud service. In order to successfully build Sitecore web index, you must first populate the managed schema.

To configure 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 XP Kubernetes secret.
3. In the Sitecore Control Panel, in the **Indexing** tab, click **Populate Solr Managed Schema**.
4. In the **Schema Populate** dialog box, select all the indexes and then click **Populate**.

   > **NOTE**
   > Wait for the process to complete, then close the dialog box.

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

6.7. Perform full rebuild of Commerce indexes

After you initialized your environments with Commerce data (for example, the sample AdventureWorks or Habitat environments), you must rebuild the following Commerce indexes using Postman:

- Catalog Items
- Promotions
- Price Cards

To rebuild Commerce search indexes using Postman :

1. In the Postman **Collections** pane, expand the *SitecoreCommerce_DevOps* collection.
2. Open the *Minions* folder, and execute the following request:
    - *Run FullIndex Minion - Catalog Items* request.
    - *Run FullIndex Minion - Promotions* request.
    - *Run FullIndex Minion - PriceCards*

7. Appendix A – Kubernetes secrets list

The Kubernetes secrets files included in the Sitecore Commerce Container SDK package are divided by folders as follows:

- XC Kubernetes secret files
- XP Kubernetes secret files

7.1. XC Kubernetes secret files

The following table lists and describes the XC Kubernetes secret files located in the `/k8s-commerce-xc1/secrets/xc/` folder.

File name	Description	Default value
`commerce-bizfx-currency.txt`	The currency to use by default by the shop.	`USD` **NOTE** If the value for `DefaultShopCurrency` is not one of the default currencies supported by the Commerce Engine (USD, CAD, or EUR), you must add the currency to the `DefaultCurrency` set on the Commerce Control Panel after you deploy.
`commerce-bizfx-language.txt`	The default language of the Business Tools user interface.	`en`
`commerce-bizfx-shopname.txt`	The default shop name.	`CommerceEngineDefaultStorefront`
`commerce-connect-clientid.txt`	The ID assigned to the Commerce Engine Connect client used to authenticate with Sitecore Identity.	CommerceEngineConnect

`commerce-connect-redis-connection-string.txt`	Connect string used by Commerce Connect to connect to the Redis instance.	`redis:6379,defaultDatabase=1,allowAdmin=true,syncTimeout=`
		`600000`

`commerce-connect-clientsecret.txt`	Shared secret between the Identity Server and Commerce Engine Connect client roles. Length: 64 characters	*No default*

File name	Description	Default value
commerce-engine-braintreeenvironment.txt	The Braintree environment.	*No default*
commerce-engine-braintreemerchantid.txt	Your merchant ID for the Braintree payment provider.	*No default*
commerce-engine-braintreeprivatekey.txt	The private key associated to your Braintree account.	*No default*
commerce-engine-braintreepublickey.txt	The public key associated to your Braintree account.	*No default*
commerce-solr-connection-string.txt	The connection string for Sorl.	http://solr:8983/solr
commerce-redis-connection-string.txt	The connection string used by the Commerce Engine to connect to the Redis instance.	redis:6379,ssl=False,abortConnect=False
xc/commerce-solr-prefix-name.txt **NOTE** Available in Sitecore.Commerce.Containers.SDK 1.1.10 and later.	The prefix to use in Commerce Solr core names (as in commerce_ CatalogItemsScope, for example.)	commerce

7.2. XP Kubernetes secret files

The following table lists and describes the XP Kubernetes secret files located in the `/k8s-commerce-xc1/secrets/xp/` folder.

File name	Description	Default
sitecore-adminpassword.txt	The Sitecore admin user password	Password12345
sitecore-collection-shardmapmanager-database-password.txt	User password for database name `*_Xdb.Collection.ShardMapManager` in MS SQL Server.	Password12345
sitecore-collection-shardmapmanager-database-username.txt	User name for database name `*_Xdb.Collection.ShardMapManager` in MS SQL Server.	shardmapmanageruser
sitecore-core-database-password.txt	User password for database name `*_Core` in MS SQL Server.	Password12345
sitecore-core-database-username.txt	User name for database name `*_Core` in MS SQL Server.	coreuser
sitecore-database-elastic-pool-name.txt	Database elastic pool name	*no default*
sitecore-databasepassword.txt	User password to connect to the MS SQL Server	Password12345
sitecore-databaseservername.txt	Server name of the MS SQL Server	mssql

File name	Description	Default
`sitecore-databaseusername.txt`	User name used to connect to MS SQL Server	`sa`
`sitecore-exm-master-database-password.txt`	User password for database name `*_EXM.Master` in MS SQL Server	`Password12345`
`sitecore-exm-master-database-username.txt`	User name for database name `*_EXM.Master` in MS SQL Server	`exmmasteruser`
`sitecore-forms-database-password.txt`	User password for database name `*_ExperienceForms` in MS SQL Server	`Password12345`
`sitecore-forms-database-username.txt`	User name for database name `*_ExperienceForms` in MS SQL Server	`formsuser`
`sitecore-identitycertificate.txt`	Certificate for input to the site	*No default*
`sitecore-identitycertificatepassword.txt`	Password to authenticate to the site.	*No default*
`sitecore-identitysecret.txt`	Shared secret between the Identity Server and client roles. Length: 64 characters	*No default*
`sitecore-license.txt`	Sitecore license string.	*No default*
`sitecore-marketing-automation-database-password.txt`	User password for database name `*_MarketingAutomation` in MS SQL Server	`Password12345`
`sitecore-marketing-automation-database-username.txt`	User name for database name `*_MarketingAutomation` in MS SQL Server	`mauser`
`sitecore-master-database-password.txt`	User password for database name `*_Master` in MS SQL Server	`Password12345`
`sitecore-master-database-username.txt`	User name for database name `*_Master` in MS SQL Server	`masteruser`
`sitecore-messaging-database-password.txt`	User password for database name `*_Messaging` in MS SQL Server	`Password12345`
`sitecore-messaging-database-username.txt`	User name for database name `*_Messaging` in MS SQL Server	`messaginguser`
`sitecore-processing-engine-storage-database-password.txt`	User password for database name `*_ProcessingEngineStorage` in MS SQL Server	`Password12345`
`sitecore-processing-engine-storage-database-username.txt`	User name for database name `*_ProcessingEngineStorage` in MS SQL Server	`processingenginestorageuser`
`sitecore-processing-engine-tasks-database-password.txt`	User password for database name *_ProcessingEngineTasks in MS SQL Server	`Password12345`
`sitecore-processing-engine-tasks-database-username.txt`	User name for database name `*_ProcessingEngineTask`s in MS SQL Serve	`processingenginetasksuser`
`sitecore-processing-pools-database-password.txt`	User password for database name `*_Processing.Pools` in MS SQL Server	`Password12345`
`sitecore-processing-pools-database-username.txt`	User name for database name `*_Processing.Pools` in MS SQL Server	`processingpoolsuser`
`sitecore-processing-tasks-database-password.txt`	User password for database name `*_Processing.Tasks` in MS SQL Server	`Password12345`

File name	Description	Default
`sitecore-processing-tasks-database-username.txt`	User name for database name `*_Processing.Tasks` in MS SQL Server	`processingtasksuser`
`sitecore-redis-connection-string.txt`	Connection string to the Redis instance	`redis:6379,ssl=False,abortConnect=False`
`sitecore-reference-data-database-password.txt`	User password for database name `*_ReferenceData` in MS SQL Server	`Password12345`
`sitecore-reference-data-database-username.txt`	in MS SQL Server`*_ReferenceData`User name for database name	`refdatauser`
`sitecore-reportingapikey.txt`	API authentication key for the reporting API.	`00112233445566778899AABBCCDDEEFF`
`sitecore-reporting-database-password.txt`	User password for database name `*_Reporting` in MS SQL Server	`Password12345`
`sitecore-reporting-database-username.txt`	User name for database name `*_Reporting` in MS SQL Server	`reportinguser`
`sitecore-solr-connection-string.txt`	Connection string to Sitecore Solr instance.	`http://solr:8983/solr;solrCloud=true`
`sitecore-solr-connection-string-xdb.txt`	Connection string to Solr-xdb.	`http://solr:8983/solr/sitecore_xdb;solrCloud=true`
`sitecore-telerikencryptionkey.txt`	Symmetric key used by the Telerik web controls. Length: 64-128 characters	*No default*
`sitecore-web-database-password.txt`	User password for database name `*_Web` in MS SQL Server	`Password12345`
`sitecore-web-database-username.txt`	User name for database name `*_Web` in MS SQL Server	`webuser`
`xp/sitecore-solr-core-prefix-name.txt` **NOTE** Available in Sitecore.Commerce.Containers.SDK 1.1.10 and later.	The prefix to use in Sitecore Solr core names (as in sitecore_master_index, for example.) **NOTE** If you change the prefix default value, you must update the `xp/solr-connection-string-xdb.txt` secret to include the new prefix as part of that connection string.	`sitecore`

8. Appendix B - Common issues

The section provides information to help troubleshooting common issues.

8.1. Can’t 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

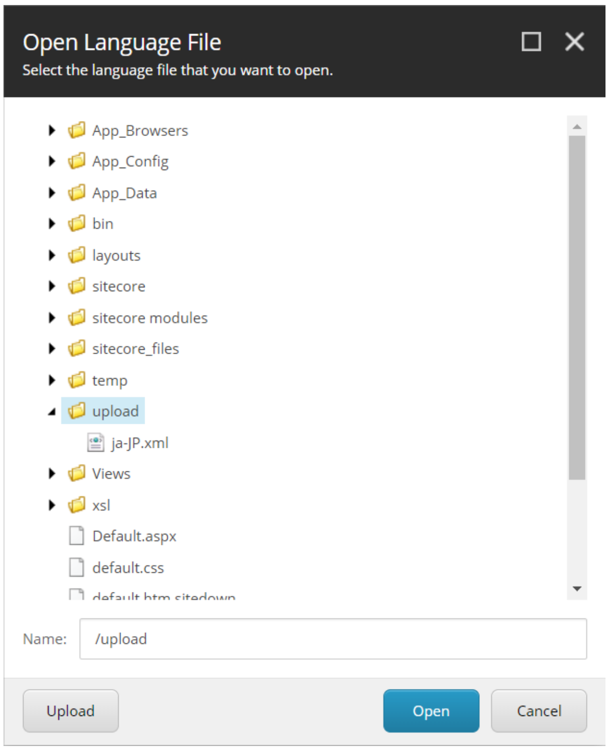
Message: Access to the path 'C:\inetpub\wwwroot\ja-JP.xml' is denied.

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 workaround this issue, you should 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**.



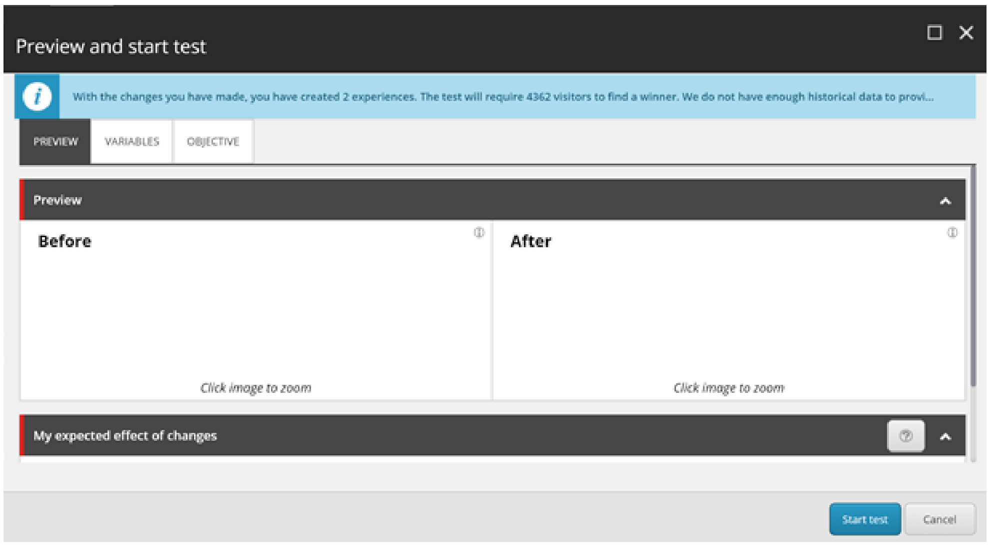
2. In the **Upload Files** dialog box, you can now upload the translations file to the \upload\ folder.

You can now import translations file from this folder.

Note: the translations xml file will be saved to the Media library. It can be deleted after translations are imported.

8.2. Screenshots are not generated on Content testing dialogs

On XC1 Kubernetes deployments where global DNS names are not configured, you might face with a problem where the **Preview and Start test** dialog or the **Test result** dialog display blank screens:


With the changes you have made, you have created 2 experiences. The test will require 4362 visitors to find a winner. We do not have enough historical data to provi...

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

As a workaround, you can add an instruction to the container lifecycle ***PostStart*** hook, which adds a record to the hosts file on the pod start as shown below to `cm.yaml` file for cm role:

```
containers:
- name: sitecore-xc1-cm
image: {registry}/{project}/sitecore-xc1-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'"]
```

8.3. Only main Sitecore log is exposed for Sitecore roles containers

The LogMonitor tool is used to collect and output logs from containers. It is configured to monitor following logs:

- 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

NOTE
Auxiliary Sitecore logs (like search, crawling or publishing, etc.) are not monitored on Sitecore containers. To see auxiliary logs, you can connect to corresponding container via terminal (powershell or command prompt) and view them directly from the container's filesystem.