

SXC 10.2 Installation Guide for a Developer Workstation with Containers

Sitecore Experience Commerce 10.2

November 4, 2021



Table of Contents

1. Introduction	3
1.1. Topologies	4
1.1.1. XC Workstation topology (XC0)	4
1.1.2. XC Scaled topologies (XC1 and XC1-CXA)	4
1.1.3. Roles included in XC0 and XC1-CXA topologies	4
1.2. Sitecore XC Docker Compose files	6
1.2.1. Docker Compose file	6
1.2.2. Environment variables file	6
1.3. Software Requirements	7
1.4. Hardware requirements	8
2. Prepare for Commerce containers deployment	9
2.1. About production and non-production containers images	10
2.2. Use a script to automate preparation for deployment	10
2.3. The environment variables list	12
3. Deploy a Sitecore XC developer workstation	17
4. Post-deployment tasks	19
4.1. Bootstrap and initialize the Commerce Engine	20
4.1.1. Setup the environment in Postman	20
4.1.2. Bootstrap and initialize the Commerce Engine	21
4.2. Validate the deployment	23
4.2.1. Validate the deployment of Business Tools	23
4.2.2. Validate access to the Content Management instance	23
4.3. Configure user accounts	24
4.4. Generate catalog templates	25
4.5. Perform full rebuild of Commerce indexes	26
4.6. Populate Solr managed schema	27
4.7. Manually rebuild Sitecore XP indexes	27
4.8. Create an SXA Storefront tenant and site	28
4.9. Clean up a workstation environment	29
5. Appendix A: Using existing Solr and SQL services in a Commerce Docker deployment	30
5.1. Configure a Commerce Docker container deployment to use an existing SQL service	30
5.2. Configure a Commerce Docker container deployment to use an existing Solr service	31
5.3. Prepare for deployment	32
5.4. Additional consideration when deploying the containerized solution	32
5.5. Post-deployment steps when using an existing SQL service instance	33
6. Appendix B: the Windows hosts file	34

1. Introduction

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

This guide provides step-by-step instructions for installing Sitecore Experience Commerce on a developer workstation, based on the sample images that are included in the Sitecore Commerce Container SDK package, available on the [Sitecore Downloads](#) site.

For information on installing Sitecore Experience Commerce using Kubernetes in a production environment, download the [Sitecore XC Installation Guide for Production Deployments with Kubernetes](#).

1.1. Topologies

The Sitecore Commerce Container SDK provides container images that allows you to install Sitecore XC on a developer workstation in the following topologies:

- [XC Workstation topology \(XC0\)](#)
- [XC Scaled topology \(XC1 and SX1-CXA\)](#)

1.1.1. XC Workstation topology (XC0)

The XC0 topology is used to deploy the Commerce Engine only, in a Docker workstation developer environment. This topology is not intended for production deployment. It is designed to reduce memory overhead, reduce download size, improve startup/shutdown time, and reduce complexity.

NOTE

The XC0 topology for developer workstation *does not* include container images for the SXA storefront or supporting modules.

For a deployment that includes the modules required to deploy the SXA storefront, you must install the [XC1-CXA topology](#).

1.1.2. XC Scaled topologies (XC1 and XC1-CXA)

The Sitecore Commerce Container SDK includes two XC scaled topologies:

- The XC1-CXA topology is for a scaled Commerce deployment. The XC1-CXA topology includes container images of modules required to support the SXA Storefront. This is the topology you need to deploy if you want the SXA Storefront in your development environment.

NOTE

The XC1-CXA topology does not include a default storefront site configuration. The topology provides container images of all supporting modules required to create a tenant and SXA Storefront site as post-deployment steps.

- The XC1 topology is for a scaled deployment that *excludes* modules supporting the SXA Storefront. You use this topology in a scaled, Commerce Engine only deployment, that does not use the SXA Storefront.

NOTE

The Sitecore Commerce Container SDK does not include all Docker compose YAML files required to deploy the container images of the XC1 topology.

The resources required to run XC in a non-production deployment can be significant but are required to mimic the exact configuration that is used in production. In non-production deployments, we recommend that you run XC with the [workstation hardware requirements](#).

1.1.3. Roles included in XC0 and XC1-CXA topologies

The XC0 and XC1-CXA topologies include container images for Sitecore roles and for external services.

The following table lists the Sitecore roles that are included in the XC0 and XC1-CXA topologies, respectively.

NOTE

Sitecore XP and Sitecore XC asset images are provided as samples and are only suitable for non-production deployment.

You must create your own set of container images from your customized Commerce solution.

Sitecore roles	XC0 topology	XC1-CXA topology
Sitecore Identity Server	✓	✓
Sitecore Commerce Business Tools (BizFX)	✓	✓
Sitecore Commerce Authoring	✓	✓
Sitecore Commerce Shops	✓	✓
Sitecore Commerce Minions	✓	✓
XDB Search Worker	✓	✓
Marketing Automation Engine	✓	✓
Sitecore Cortex Processing Engine	✓	✓
xConnect Server (Standalone)	✓	
Content Management (Standalone CM/CD)	✓	
xConnect Server		✓
Content Management		✓
Content Delivery		✓
XDB Processing		✓
XDB Reporting service		✓
XDB Collection service		✓
XDB Search service		✓
XDB Reference Data service		✓
Marketing Automation Reporting		✓

In addition to the Sitecore roles, the XC0 and XC1-CXA topologies include sample container images of external services.

NOTE

The external services are provided as non-production container images to facilitate the deployment of the Sitecore XC sample solution in a development environment. They are not intended nor suitable for production deployment.

- Microsoft SQL Server
- Apache Solr
- RedisLabs Redis Server
- Traefik Reverse Proxy

1.2. Sitecore XC Docker Compose files

Overview of how Docker Compose files are used in a Sitecore XC deployment and the information they contain about containers and configurations.

To deploy the containers, Sitecore Docker Compose requires the following files:

- The `docker-compose.yml` file
- The `.env` file

NOTE

The Docker Compose files for each Sitecore XC topology are included in the Sitecore Commerce Container SDK.

1.2.1. Docker Compose file

The [Docker Compose configuration file](#) is a text file (`docker-compose.yml`) that contains information about the different containers and configuration of each Sitecore role in the deployment topology. The Docker Compose file is the main configuration file that the `docker-compose` command uses.

1.2.2. Environment variables file

The [environment variable configuration file](#) is a text file (`.env`) that stores the configuration information for the environment you want to deploy. The [sample .env file](#) provides default values for any environment variables referenced in the Docker Compose file. You can edit this file outside the main Docker Compose configuration, that is, using a text editor without using `docker-compose` command line utility.

The environment variables are the preferred mechanism for passing configuration settings into containers.

The following example shows how, for example, the `mssql` service role in the `compose.yml` file uses an environment variable defined in the `.env` file to configure the SQL Server SA password (`SA_PASSWORD`):

```
... mssql:
  isolation: ${ISOLATION}
  image: ${XP_SITECORE_DOCKER_REGISTRY}nonproduction/mssql-developer:2017-${XP_SITECORE_TAG}
  restart: unless-stopped
  environment:
    SA_PASSWORD: ${SQL_SA_PASSWORD}
    ACCEPT_EULA: "Y"
  ports:
    - "14330:1433"
  volumes:
    -type: bind
      source: c:\containers\mssql-data
      target: c:\data
...
```

1.3. Software Requirements

The following are software requirements for installing Sitecore Experience Commerce 10.2 with containers on your developer workstation:

- Operating system:
 - Windows 10 1809 or later
 - Windows Server 1809 or later.

NOTE

For more information about Windows and containers compatibility for running containers in Hyper-V isolation and process isolation mode, refer to [Microsoft's documentation](#).

- Microsoft PowerShell 5.1
- [Docker Desktop for Windows](#)
- [Sitecore Commerce Container SDK](#)

NOTE

You must extract the Docker Compose configuration folder for the desired topology.

See this [article](#) for additional details about Sitecore XC software compatibility for this release.

1.4. Hardware requirements

The following are hardware requirements for installing Sitecore Experience Commerce with containers on your developer workstation:

- RAM
We recommend that a developer workstation has a minimum of 32GB of RAM.
- CPU
We recommend a quad core or higher.
- Disk
Sitecore container images require approximately 25 GB of free disk space. We recommend the use of solid-state drive (SSD) disks for optimal performance when downloading and running Docker containers.

NOTE

The type of disks used for SQL Server and Solr can have a significant impact on performance.

2. Prepare for Commerce containers deployment

Before you can deploy the Sitecore XC containers, there are some concepts you need to be familiar with, and some procedures you need perform.

To prepare for Commerce container deployment:

- [Read about production and non-production container images](#)
- Familiarize yourself with [environment variables](#).
- [Download](#) the `Sitecore.Commerce.Container.SDK.*.*.*.ZIP` package and familiarize yourself with its content. For a description of the SDK, see this [topic](#).
- Familiarize yourself with the [ComposeInit script](#).

2.1. About production and non-production containers images

Sitecore XC Docker images that are provided as samples are not suitable for a production environment.

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

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

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.

2.2. Use a script to automate preparation for deployment

The [Sitecore.Commerce.Container.SDK](#) provides a script that automates some of the preparation tasks for Sitecore deployment to a developer machine. The path to the script file is as follows:
`Sitecore.Commerce.Container.SDK/scripts/ComposeInit.ps1`.

The `ComposeInit.ps1` script lets you set the mandatory variable values that are required to deploy a minimally customized Sitecore XC developer environment, in the XC1-CXA configuration topology.

The `ComposeInit.ps1` script performs the following actions:

- Compresses the Sitecore license file.
- Creates the Identity Server token signing certificate.
- Populates environment variables in the `.env` file.

NOTE

For a brief description of all variables contained in the `.env` file and their default values, see [the environment variables list](#).

- Updates Traefik configuration if custom host names are used.
- Updates Windows hosts names.

To execute the `ComposeInit.ps1` script:

1. Make sure to have values for the following variables at hand:

NOTE

These are the values required in a minimally customized environment. All other parameters use default values (paths) based on the structure of the `Sitecore.Commerce.Container.SDK`.

- `LicenseXmlPath`
- `BraintreeEnvironment`
- `BraintreeMerchantId`
- `BraintreePublicKey`
- `BraintreePrivateKey`

NOTE

By default, the script deploys the XC1-CXA topology.

2. Open a PowerShell window, replace the parameterized variable values with your own, and then run the sample script:

```
/ComposeInit.ps1 -LicenseXmlPath <pathToSitecoreLicense> -BraintreeEnvironment  
<BraintreeEnvironment> -BraintreeMerchantId <BraintreeMerchantId>-BraintreePublicKey  
<BraintreePublicKey> -BraintreePrivateKey <BraintreePrivateKey>
```

The script creates the following Docker container volumes under the default path `c:\containers`:

- `cm\domains-shared`
- `cd\domains-shared`
- `mssql-data`
- `solr-data`

NOTE

If you change any paths in the script, the script will not update the updated paths where referenced in the Docker `.yaml` files.

Using custom environment variables parameters

The `ComposeInit.ps1` script supports some parameters that let you overwrite default values with your own environment variable values.

The following table lists parameters that you can set to change hostnames and default passwords:

Parameter	Environment variable
<code>SitecoreAdminPassword</code>	<code>SITECORE_ADMIN_PASSWORD</code>
<code>SqlSaPassword</code>	<code>SQL_SA_PASSWORD</code>
<code>SqlServer</code>	<code>SQL_SERVER</code>
<code>SqlUserName</code>	<code>SQL_SA_LOGIN</code>

Parameter	Environment variable
IdHost	ID_HOST
CdHost	CD_HOST
CmHost	CM_HOST
BizFxHost	BIZFX_HOST
AuthoringHost	AUTHORING_HOST
ShopsHost	SHOPS_HOST
MinionsHost	MINIONS_HOST

When you set custom hostnames, the `ComposeInit.ps1` script generates the TLS certificates required for those hosts, updates the Traefic configuration, modifies the `.env` file, and updates the hosts file accordingly.

NOTE

To further customize the deployment, update the environment variables and modify the `ComposeInit.ps1` script to suit your needs.

2.3. The environment variables list

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

The environment variables for Sitecore Docker Compose are stored in an environment variable configuration file (`.env`). Docker Compose loads these variables automatically during startup.

IMPORTANT

All the environment variables must fit inside a single 32,700 character block in the `.env` file. If the total size of your own variables combined with the Sitecore variables exceeds this size, you will be unable to set new values, and the system will **not deploy successfully**.

The following table lists and briefly describes the environment variables contained in the `.env` file for each Sitecore topology.

NOTE

Variables that do not have a default value are set when you run the `ComposeInit.ps1` script.

Environment variables for XC0, XC1, and XC1-CXA topologies

Variable name	Topology	Default value	Description
BASE_SITECORE_DOCKER_REGISTRY	XC0, XC1-CXA	src.sitecore.com/base/	The base Sitecore container registry.
XP_SITECORE_DOCKER_REGISTRY	XC0, XC1, XC1-CXA	scr.sitecore.com/sxp/	Sitecore container registry.
XP_SITECORE_TAG	XC0, XC1, XC1-CXA	10.2-1tsc2019	Image tag with the version to be pulled from the container registry.
<p>NOTE Refer to the Sitecore.Commerce.Container.SDK for the latest value.</p>			
MODULES_SITECORE_DOCKER_REGISTRY	XC1, XC1-CXA	scr.sitecore.com/sxp/modules/	The Sitecore modules container registry.
SPE_SITECORE_TAG	XC1-CXA	6.3-1809	The container image tag for the Sitecore Powershell Extension module.
<p>NOTE Refer to the Sitecore.Commerce.Container.SDK for the latest value.</p>			
SXA_SITECORE_TAG	XC1-CXA	10.2-1809	The container image tag for the Sitecore SXA module.
<p>NOTE Refer to the Sitecore.Commerce.Container.SDK for the latest value.</p>			
XC_NONPRODUCTION_SITECORE_DOCKER_REGISTRY	XC0, XC1-CXA	scr.sitecore.com/sxc/nonproduction	The Sitecore Commerce container registry for non-production container image.
XC_SITECORE_DOCKER_REGISTRY	XC0, XC1-CXA	scr.sitecore.com/sxc/	Sitecore Commerce container registry.
XC_PACKAGES_TAG	XC0, XC1, XC1-CXA	10.2-1tsc2019	Image tag with the version to be pulled from the container registry.
TRAEFIK_IMAGE	XC0, XC1-CXA	traefik:v2.2.0-windowsservercore-1809	The Traefik image tag.
TRAEFIK_ISOLATION	XC0, XC1-CXA	hyperv	Override for Docker isolation level for traefik. Possible values: default, hyperv, process.

Variable name	Topology	Default value	Description
ISOLATION	XC0, XC1-CXA	default	Override for Docker isolation modes . Possible values: default, hyperv, process.
CD_HOST	XC1-CXA	xc1-cxacd.localhost	CD host name.
CM_HOST	XC0, XC1-CXA	For XC0: xc0cm.localhost For XC1-CXA: xc1-cxacm.localhost	CM host name.
ID_HOST	XC0, XC1-CXA	For XC0: xc0id.localhost For XC1-CXA: xc1-cxaid.localhost	ID Server host name.
AUTHORING_HOST	XC0, XC1-CXA	authoring.localhost	Authoring service host name.
SHOPS_HOST	XC0, XC1-CXA	shops.localhost	Shops service host name.
MINIONS_HOST	XC0, XC1-CXA	minions.localhost	Minions service host name.
BIZFX_HOST	XC0, XC1-CXA	bizfx.localhost	The Business Tools host name.
SITECORE_ADMIN_PASSWORD	XC0, XC1-CXA,	Password12345	The Sitecore application administrator password.
SQL_SERVER	XC0, XC1-CXA,	mssql	The SQL server hostname
SQL_SA_LOGIN	XC0, XC1-CSX,	sa	The SQL server SA login name
SQL_SA_PASSWORD	XC0, XC1-CXA	Password12345	The SQL Server administrator password.
SITECORE_MASTER_DB	XC0, XC1, XC1-CXA	Sitecore.Master	The Sitecore master db name.
SITECORE_CORE_DB	XC0, XC1, XC1-CXA	Sitecore.Core	The Sitecore core db name.
XC_GLOBAL_DB	XC0, XC1, XC1-CXA	SitecoreCommerce_Global	The Sitecore Commerce global db name.
XC_GLOBAL_DB_TRUSTED_CONNECTION	XC0, XC1-CXA	false	Whether the connection is trusted or not.
XC_SHARED_DB	XC0, XC1 XC1-CXA	SitecoreCommerce_SharedEnvironments	The Sitecore Commerce shared db name.
XC_SHARED_DB_TRUSTED_CONNECTION	XC0, XC1-CXA	false	Whether the connection is trusted or not.
XC_SHARED_ARCHIVE_DB	XC0, XC1-CXA	SitecoreCommerce_ArchiveSharedEnvironments	The name of the database where archived Commerce entities are stored.

Variable name	Topology	Default value	Description
XC_ENGINE_PORT	XC0, XC1-CXA	5000	The port used to configure the internal Commerce Engine service URLs.
XC_ENGINE_BRAINTREEENVIRONMENT	XC0, XC1-CXA	<i>no default</i>	The Braintree environment name. Set when running the <code>ComposeInit.ps1</code> script.
XC_ENGINE_BRAINTREEMERCHANTID	XC0, XC1-CXA	<i>no default</i>	Your merchant ID for the Braintree payment provider . Set when running the <code>ComposeInit.ps1</code> script.
XC_ENGINE_BRAINTREEPUBLICKEY	XC0, XC1-CXA	<i>no default</i>	The public key associated to your Braintree account. Set when running the <code>ComposeInit.ps1</code> script.
XC_ENGINE_BRAINTREEPRIVATEKEY	XC0, XC1-CXA	<i>no default</i>	The private key associated to your Braintree account . Set when the running the <code>ComposeInit.ps1</code> script.
XC_BIZFX_DEFAULT_LANGUAGE	XC0, XC1-CXA	en	The default language used in the Business Tools user interface.
XC_BIZFX_DEFAULT_CURRENCY	XC0, XC1-CXA	USD	The default currency to use by the shop.
XC_BIZFX_DEFAULT_SHOPNAME	XC0, XC1-CXA	CommerceEngineDefaultStorefront	The default shop name.
XC_ENGINE_CONNECT_CLIENTID	XC0, XC1-CXA	CommerceEngineConnect	The client ID assigned to Commerce Engine Connect for Sitecore Identity. This ID is used to identify the Commerce Engine Connect with Commerce Engine.
XC_IDENTITY_COMMERCEENGINECONNECTCLIENT_CLIENTSECRET1	XC0, XC1-CXA	<i>no default</i>	Shared secret between the Identity Server and Commerce Engine Connect client roles. Length: 64 characters Set by the running the <code>ComposeInit.ps1</code> script.
REPORTING_API_KEY	XC0, XC1-CXA	<i>no default</i>	Symmetric key used to access the Sitecore XDB Reporting WebAPI. Length: 32 characters Set by the running the <code>ComposeInit.ps1</code> script.
TELERIK_ENCRYPTION_KEY	XC0, XC1-CXA	<i>no default</i>	Symmetric key used by the Telerik web controls. Length: 64-128 characters Set by the running the <code>ComposeInit.ps1</code> script.

Variable name	Topology	Default value	Description
SITECORE_IDSECRET	XC0, XC1-CXA	<i>no default</i>	Shared secret between the Identity Server and client roles. Length: 64 characters Set by running the <code>ComposeInit.ps1</code> script.
SITECORE_ID_CERTIFICATE	XC0, XC1-CXA	<i>no default</i>	The Identity Server certificate used to encrypt data. Set by the running the <code>ComposeInit.ps1</code> script.
SITECORE_ID_CERTIFICATE_PASSWORD	XC0, XC1-CXA	<i>no default</i>	The password required to open the Identity Server certificate. Set by the running the <code>ComposeInit.ps1</code> script.
SITECORE_LICENSE	XC0, XC1-CXA	<i>no default</i>	The path to the Sitecore license file. Set by the running the <code>ComposeInit.ps1</code> script.
SOLR_CORE_PREFIX_NAME	XC0, XC1-CXA	<code>sitecore</code>	The prefix to use in Sitecore Solr core names (as in <code>sitecore_master_index</code> , for example.)
SOLR_COMMERCE_PREFIX_NAME	XC0, XC1-CXA	<code>commerce</code>	The prefix to use in Commerce Solr core names (as in <code>commerce_CatalogItemsScope</code> , for example.)
MEDIA_REQUEST_PROTECTION_SHARE_D_SECRET	XC0, XC1-CXA	<i>no default</i>	Shared secret for media request protection. Length: 64 characters Generated and set by the running the <code>ComposeInit.ps1</code> script.

3. Deploy a Sitecore XC developer workstation

You use Docker for Windows to deploy the Sitecore XC container packages.

To deploy a Sitecore XC developer workstation using containers:

1. In Docker for Windows, [switch to Windows container mode](#). (The Docker for Windows tray icon has an option to switch between Linux and Windows container development).
2. Download and extract the `Sitecore.Commerce.Container.SDK.*.*.zip` package from the [Sitecore Downloads](#) page and store it on your local workstation.
3. In Windows Explorer, go to the folder where you extracted the Sitecore Commerce Container SDK, and open the Docker Compose folder for the topology that you want to deploy.
4. Open the `/xc1-cxa/docker-compose.yml` file, and review the content to get a better understanding of the containers and connection strings between the different roles.
5. In the Windows console, go to the folder that contains the `docker-compose.yml` file, and run the following Docker Compose command:

```
docker-compose up --detach
```

NOTE

Docker Compose pulls all the required images from the Sitecore Container Registry, creates the required Docker network configuration, and deploys all the containers to the local environment. When the deployment is successfully completed, the Docker Compose command exits.

IMPORTANT

Before running an image, ensure that your host operating system version is greater than or equal to the OS version of the container image. Otherwise, the following error may occur: "ERROR: manifest not found: manifest unknown: manifest unknown" error."

Also, ensure the host OS version patch level matches or is greater than the container OS image patch level.

6. To check the Docker container status, run the following command:

```
docker container list
```

NOTE

This command generates a list of all the containers and their current status.

7. When the status of all the containers is healthy, validate that you can access the Commerce Authoring environment. Open a browser and enter the URL for the instance of the Commerce Engine running the Commerce Authoring service. The default host name for the Commerce Authoring is: `https://authoring.localhost/commerceops/$metadata`. (The default host names for the other topologies are listed [here](#).)

NOTE

Commerce services run on the following ports by default: 443, 8079 and 8080. To avoid errors, ensure to stop all services running on these ports.

8. When deployment is done, you complete the [post-deployment steps](#).

4. 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](#)
- [Perform full rebuild of Commerce indexes](#)
- [Populate Solr managed schema](#)
- [Manually rebuild Sitecore XP indexes](#)
- [Create an SXA Storefront tenant and site \(for XC1-CXA topology only\)](#)
- [Clean up a workstation environment](#)

4.1. Bootstrap and initialize the Commerce Engine

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

The 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 run 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 samples](#) deployed. The Postman samples are included as part of the [Sitecore Commerce Engine SDK](#), available for [download](#) in *Sitecore XC Packages for On Premise WDP*.

4.1.1. Setup the environment in Postman

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

Within the Sitecore Container SDK, the *postman* folder contains predefined sets of sample environment files (for the Habitat environment and for the Adventure Works environment). You can import either of these sample environments into Postman or create your own.

NOTE

With a new installation of Postman, you must disable SSL certificate verification in order to get a response back from the Commerce Engine. To do this, in Postman, click **File, Settings** and then set `SSL certificate verification` to `OFF`.

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

1. In the top right corner of Postman, click the **Manage Environments** icon.
2. In the **Manage Environments** dialog, click `Import`, click `Choose file`, and then browse to the `Sitecore Commerce Container SDK/postman` folder and select the appropriate environment file for your deployment.
3. Click **Add**.
The following shows an example of the predefined Habitat environment variables for XC container images running in Docker:

Variable (key)	Default value
Environment	HabitatAuthoring
ShopName	CommerceEngineDefaultStorefront
ShopperId	ShopperId
Language	en-US
Currency	USD
ServiceHost	https://authoring.localhost
ShopsApi	api

Variable (key)	Default value
OpsApi	commerceops
AuthoringHost	https://authoring.localhost
MinionsHost	https://minions.localhost
ShopsHost	https://shops.localhost
SitecoreIdServerHost	https://{{Topology}}id.localhost

NOTE

The `{{Topology}}` variable is set based on the `Topology` key value. The `Topology` value should be set to `xc0` or `xc1-cxa`, based on your deployment. For example, if the `Topology` key is set to `xc1-cxa`, then `https://{{Topology}}id.localhost` resolves to `https://xc1-cxaid.localhost`.

SitecoreIdServerUserName	sitecore\admin
SitecoreIdServerPassword	The admin user password used to authenticate the request. (default: Password12345)
GeoLocation	IpAddress=1.0.0.0
MinionsEnvironment	HabitatMinions
ShopsEnvironment	HabitatShops
Protocol	https
Topology	xc0 Possible values: xc0 or xc1-cxa.

4.1.2. 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. Open 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 that 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 the above steps for other environments if applicable (for example, for the AdventureWorks environment).

4.2. Validate the deployment

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

4.2.1. 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.localhost`.
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 instructions](#) and, in the **Link** field, enter the URL `https://bizfx.localhost/` (instead of `https://localhost:4200`).

4.2.2. 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:
 - In a XC0 topology: `https://xc0cm.localhost`
 - In a XC1-CXA topology: `https://xc1-cxacm.localhost`
2. Validate that you can login to Sitecore and access the Sitecore Launchpad.

4.3. 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.

4.4. 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://xc1-cxacm.localhost/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).

4.5. 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 :

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*

4.6. Populate Solr managed schema

To populate Solr managed schema:

1. Login to Sitecore
2. Open Sitecore Control Panel and, in the **Indexing** section, click **Populate Solr Managed Schema**.
3. In the **Schema Populate** dialog, select the Sitecore index for which to populate schema. At a minimum, you must select the Sitecore Web index (**sitecore_web_index**) and Sitecore Master index (**sitecore_master_index**).
4. Click **Populate**.

4.7. Manually rebuild Sitecore XP indexes

There are situations where you need to completely rebuild Sitecore search indexes, for example, when you deploy to a production environment, when you have new or changed content to re-index, when indexes are out of date, or when indexes have been corrupted.

You can manually trigger a complete rebuild of the Sitecore master and web indexes to ensure that they reflect the latest changes in your Commerce data. The following procedure shows how to rebuild Sitecore XP indexes using the Sitecore Control Panel, but there are other ways to perform a complete rebuild of Sitecore search indexes.

Alternatively, you can perform a partial update of the Sitecore master and web indexes, by only re-indexing Commerce content within those indexes, [using Postman API requests](#).

NOTE

It is best practice to rebuild each index separately. It does not matter which one you choose to build first.

To rebuild a Sitecore search index :

1. From the **Sitecore Launchpad**, open the **Control Panel**.
2. In the **Indexing** section, click **Indexing Manager**.
3. In the **Indexing Manager** dialog box, select **sitecore_master_index** and click **Rebuild**.
4. When the Sitecore master index is rebuilt, repeat the steps above to rebuild the **sitecore_web_index**.

4.8. Create an SXA Storefront tenant and site

If your deployment topology includes the SXA Storefront and you 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.

4.9. Clean up a workstation environment

You can stop or completely remove workstation environment.

To stop a Docker Compose environment without removing its contents, run:

```
docker-compose stop
```

To resume a previously stopped Docker Compose environment, run:

```
docker-compose start
```

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

```
docker-compose down
```

5. Appendix A: Using existing Solr and SQL services in a Commerce Docker deployment

How to configure a Commerce Docker deployment to use existing instances of Solr and SQL hosted services.

The `Sitecore.Commerce.Container.SDK` package includes sample container images for SQL and Solr services. These sample services allow you to deploy a complete, new working sample solution. However, there are scenarios where you might want to convert an existing, non-containerized Commerce deployment to a containerized Commerce Docker deployment, and having this deployment use your existing instance of Solr and SQL services. The following information provides basic, common instructions for deploying the Sitecore XC sample containerized solution with Docker, but using an existing instance of Solr and SQL services.

NOTE

This information is provided as guidance and, depending on your customizations or other circumstances specific to your Commerce deployment, additional configuration could be required.

5.1. Configure a Commerce Docker container deployment to use an existing SQL service

To configure your Commerce container deployment to use an existing SQL service instance:

1. In the `Sitecore.Commerce.Container.SDK` package, open the `<topology>/ .env` file, and update the environment variables to match the configuration of your existing SQL environment (if these variables are being used in your own deployment). For example, ensure that:
 - The value of the `SQL_SA_PASSWORD` and the `SQL_SERVER`, `_SQL_SA_LOGIN` variables matches the value in your existing (non-container) SQL environment.
 - The value of the `SITECORE_ADMIN_PASSWORD` matches the value in your existing (non-container) deployment.
 - All `*_DB` variable values match the database names in your existing SQL environment.
2. Open the `docker-compose.yml` file, and remove both the `mssql:` and the `mssql-init:` service sections from the file to prevent them from deploying the sample service.
3. In the `docker-compose.yml` file, other services depend on the `mssql:` or the `mssql-init:` service being either *started* or *healthy*. Remove these dependencies on the MSSQL and MSSQL-INIT services to avoid the following type of errors during deployment: `ERROR: Service '<service name>' depends on service 'mssql' which is undefined.`
For example, the following shows a dependency that must be removed, where the `cd` service depends on the health of the MSQL service:

```
cd:
  isolation: ${ISOLATION}
  image: ${XC_NONPRODUCTION_SITECORE_DOCKER_REGISTRY}sitecore-xc1-cd:${XC_PACKAGES_TAG}
  depends_on:
    mssql-init:
      condition: service_healthy
```

4. In the `docker-compose.yml` file, search for all instances of `Sitecore_ConnectionStrings` variables, and update the connection string value with the existing SQL service instance. For example, in the section that contains the `cd:` configuration, update the following connection strings with the connection string information from your existing non-containerized deployment:
 - `Sitecore_ConnectionStrings_Security`
 - `Sitecore_ConnectionStrings_Web`
 - `Sitecore_ConnectionStrings_Messaging`
 - `Sitecore_ConnectionStrings_ExperienceForms`
 - `Sitecore_ConnectionStrings_Exm.Master`
5. In the `docker-compose.yml` file, for each Commerce Engine service environment section, for example `COMMERCEENGINE_AppSettings_EnvironmentName: HabitatShops`, `COMMERCEENGINE_AppSettings_EnvironmentName: HabitatMinions`, `COMMERCEENGINE_AppSettings_EnvironmentName: HabitatAuthoring`, update the value of following parameters to your SQL server host name:
 - `COMMERCEENGINE_GlobalDatabaseServer: <YOUR SQL SERVER HOST NAME>`
 - `COMMERCEENGINE_SharedDatabaseServer: <YOUR SQL SERVER HOST NAME>`
6. In the `docker-compose.yml` file, search for any other references to MSSQL or other configuration related to MSSQL, and update the values to match the configuration of your existing, non-containerized MSSQL service.

5.2. Configure a Commerce Docker container deployment to use an existing Solr service

To configure a Commerce container deployment to use your existing Solr service instance:

1. In the `Sitecore.Commerce.Container.SDK` package, open the `<topology>/ .env` file, and update the environment variables to match the configuration of your existing Solr environment (if these variables are being used in your own deployment). For example, ensure that:
 - The value of the `SOLR_CORE_PREFIX_NAME` is configured to the correct Solr core prefix.
 - The value of the `SOLR_COMMERCE_PREFIX_NAME` is configured to the correct Solr core prefix.

2. Open the `docker-compose.yml` file, and completely remove the `solr:` and the `solr-init:` configuration sections from the file, to prevent the deployment of the sample solr service.
3. In the `docker-compose.yml` file, other services have dependencies on the condition `solr` or `solr-init` service, either being *started* or *healthy*. Remove these dependencies on the Solr service, otherwise the following error occurs during deployment: `ERROR: Service '<service name>' depends on service 'Solr' which is undefined.`
The following shows a dependency example, where the `xdbsearch` service depends on the condition of the Solr service:

```
xdbsearch:
  isolation: ${ISOLATION}
  image: ${XC_NONPRODUCTION_SITECORE_DOCKER_REGISTRY}sitecore-xc1-xdbsearch:${XC_PACKAGES_TAG}
  depends_on:
    xdbcollection:
      condition: service_started
    solr:
      condition: service_started
```

4. In the `docker-compose.yml` file, update all instances of the `Sitecore_ConnectionStrings_Solr.Search:` parameter to match the value of the Solr connection string used in your deployment.
5. Search for any other references or configuration related to SOLR, and update their value to match the configuration of your existing, non-containerized Solr deployment.

5.3. Prepare for deployment

After you have updated the `.env` and the `docker-compose` files to match the configuration of your own instance of SSQL and Solr services, [prepare the required files and certificates](#).

5.4. Additional consideration when deploying the containerized solution

To deploy Sitecore XC developer workstation using containers, follow [these instructions](#).

The following are considerations when checking the health status of Docker containers:

- If any container has an unhealthy status, verify the connection string to ensure they are correct.
- If the status of any container related to Xconnect is unhealthy, and you have confirmed that the connection strings are valid, ensure that in the `Xdb.Collection.ShardMapManager` database, the `_ShardManagement.ShardsGlobal` table specifies the correct `Servername` value.

5.5. Post-deployment steps when using an existing SQL service instance

The post-deployment requirements when using an existing SQL service are different than in a new deployment. When you deploy using existing services, you only need to perform the following post-deployment tasks:

1. Setup the Postman environment to use the appropriate environment variables, and then bootstrap the Commerce Engine. This step is required to update the Global database with your own environments.

NOTE

You can follow [these steps](#) to setup the environment using your own values, and to bootstrap the Commerce Engine. You do not need to initialize the environment.

2. Restart the Commerce Engine containers to ensure the environment updates are applied.

NOTE

For a deployment with a storefront, the required `domains.config` configuration is missing from the container. You must update the `\containers\cm\domains-shared\domains.config` file (and the same file under `\containers\cd\` if applicable) to include configuration specific to your storefront.

6. Appendix B: the Windows hosts file

The `ComposeInit.ps1` script updates the Windows host files with the names used in the default deployment. The reverse proxy container uses the host names to access the Sitecore application from a browser.

The default host names differ depending on the topology you decide to deploy. All the host names should point to the loopback IP address 127.0.0.1.

The following shows an example of the content of Windows host file definition with the default hostnames for the XC0 topology:

```
127.0.0.1 xc0cm.localhost
127.0.0.1 xc0id.localhost
127.0.0.1 bizfx.localhost
127.0.0.1 authoring.localhost
127.0.0.1 shops.localhost
```

The following table lists the default hostnames for each topology:

Topology	IP address	Hostnames
XC Workstation (XC0)	127.0.0.1	xc0cm.localhost
	127.0.0.1	xc0id.localhost
	127.0.0.1	bizfx.localhost
	127.0.0.1	authoring.localhost
	127.0.0.1	shops.localhost
XC Scaled (xc1-cxa)	127.0.0.1	xc1-cxacm.localhost
	127.0.0.1	xc1-cxacd.localhost
	127.0.0.1	xc1-cxaid.localhost
	127.0.0.1	bizfx.localhost
	127.0.0.1	authoring.localhost
	127.0.0.1	shops.localhost
	127.0.0.1	minions.localhost

NOTE

For more information, see the documentation for [Traefik Docker configuration discovery](#).