Rev: August 1, 2018

Sitecore Experience Commerce DevOps Guide

Sitecore Experience Commerce 9.0

A reference for deploying and administering the Sitecore Experience Commerce solution

sitecore®
Own the experience™

Table of Contents

Chapter 1 Introduction 3
1.1 Sitecore XC solution components 4
1.2 Sitecore XC architecture entities 5
Chapter 2 Installation 7
2.1 Software packages overview 8
2.2 Firewall ports 12
Chapter 3 Configuration 13
3.1 Security 14
3.2 User roles and permissions 17
3.3 Sitecore XC SDKs 18
3.4 Bootstrap the Commerce Engine 21
3.5 Clean and initialize the environment 22
Chapter 4 Administration 23
4.1 Commerce Service API 24
4.2 Commerce Control Panel 27
4.3 Caching 28
4.4 Fault injection 29
Chapter 5 Production deployments 30
5.1 Reference architecture 31

Chapter 1 Introduction

This document provides reference information for the production deployment, configuration, and administration of the Sitecore Experience Commerce™ (XC). It is focused primarily on the development operations (DevOps) role. It also serves as a reference for those developing and integrating solutions based on Sitecore XC.

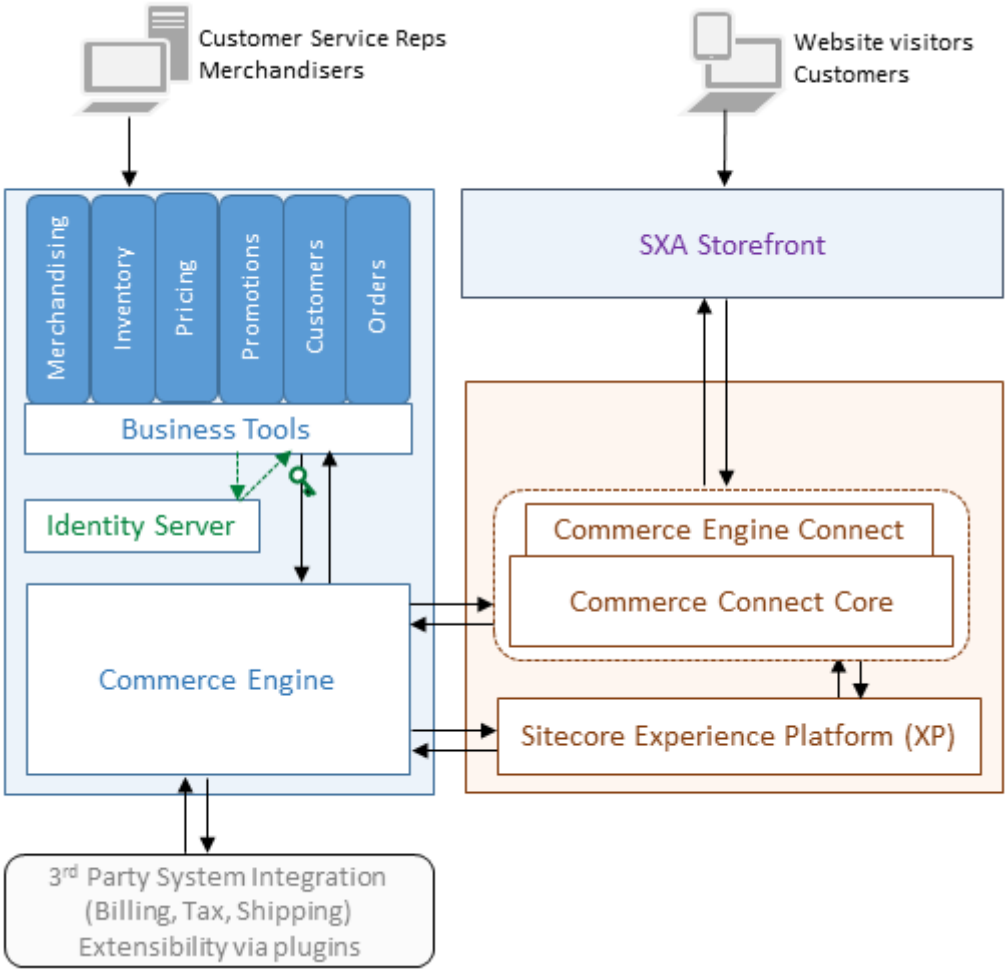
For instructions on deploying the out-of-box Sitecore XC solution on a single on-premises server, refer to the *Sitecore Experience Commerce Installation Guide for On-Premise Solutions*.

For instructions on deploying the out-of-box Sitecore XC solution to the cloud, refer to the *Sitecore Experience Commerce Installation Guide for Azure App Service*.

1.1 Sitecore XC solution components

Sitecore Experience Commerce (XC) is an e-commerce solution, built on the Sitecore Experience Platform (XP). Whether deployed on-premises or in the Azure App Service, the Sitecore XC solution enables marketers and merchandisers to fully personalize the end-to-end shopping experience.

The following figure shows a logical representation of the Sitecore XC solution components.



Commerce Connect Core — the Sitecore XP commerce API(s) for storefront developers. This is the integration layer between a front-end webshop solution and a back-end commerce system.

Commerce Engine (CE) Connect — a connector that enables integration between the Commerce Connect Core layer and Commerce Engine.

Commerce Engine — a thin ASP.NET core application that hosts commerce services such as catalogs, inventory, pricing, promotions, customers, and orders.

Business Tools — a set of tools for merchandisers and customer service representatives to access and manage commerce services. The Business Tools communicate directly with the Commerce Engine, after successful authentication by the Identity Server.

Identity Server — Sitecore's federated authentication service provider, for authenticating to the Business Tools and Commerce Engine.

SXA Storefront — a sample storefront website for customers and shoppers, and displays all commerce-related information on the site. The storefront communicates with the Commerce Engine through Commerce Engine Connect.

1.2 Sitecore XC architecture entities

The Sitecore XC solution uses a combination of architectural entities to achieve configuration, scalability, and extensibility.

1.2.1 Policies

All commerce functionality in the Sitecore XC solution is delivered by the Commerce Engine. Configuration of the Commerce Engine is defined through a set of policies.

A Sitecore XC policy is a group of settings that affect the functionality of a specific feature in the Commerce Engine. All policies are stored in a centralized policy store. Policies are heavily cached and rarely change their values outside of deployment scenario.

You can change configuration by editing the appropriate policy file.

1.2.2 Environments

A Commerce environment is a collection of policies that affect how a call to the Commerce Engine is executed.

Environments allow you to maintain separately configurable pools of data and service functionality hosted in a single service instance. This means that the same call to the Engine can behave differently, depending on what environment variable specified.

Environments can share the same persistent store as other environments or be separated into their own exclusive persistent store.

The Sitecore XC solution provides two environments by default: an Authoring environment and a Shops environment. The primary difference in the two environments is the level of caching.

- **Authoring environment**: caching is minimal. This means merchandisers or customer service representatives using the Business Tools always see the latest data because the information is retrieved directly from the database (and not cached values).
- **Shops environment**: caching is enabled for improved performance. This means that website visitors and shoppers can view product details, for example, more quickly because the data is read from cache (as opposed to retrieving the information directly from the database).

You can also create your own environment and customize the configuration as required.

Bootstrapping

Bootstrapping is the process of loading the policy and environment data from JSON files on disk created during deployment, into the global policy store so that other Commerce Engine instances can access it.

After bootstrapping, the environment JSON files are no longer needed. The Commerce Engine retrieves any subsequent environment configuration directly from the database during normal runtime operations.

1.2.3 Roles

The Sitecore XC solution implements the concept of Commerce Engine roles to support scalability.

In a production environment, traffic is usually split up among multiple installed instances of the Commerce Engine, which are usually physically located close to their traffic sources. These instances are referred to as Engine Roles. This distinction is purely logical Engine roles are defined by where the traffic they serve originates from.

Service roles can exist on the same or separate servers, and can be scaled independently as required. Each service role represents an instance of the Commerce Engine, and has different load characteristics

Each deployed role can have different policies and behaviors, which can be specified using a specific Commerce environment for that role. When a call is made to the Commerce Engine, the call's header specifies an environment. This environment is used by the engine to control policies and behavior for that call.

Specifying a particular environment allows explicit independent control (per role) of functions, such as caching, data storage, integration connections, and so on.

The Sitecore XC solution defines four Commerce Engine roles:

- **Authoring role**

  The Authoring role is the instance of the Commerce Engine that serves traffic from the Business Tools. Since this role serves lighter traffic (because ecommerce solutions have relatively few business users compared to the number of shoppers), scaling requirements are normally relatively low.

- **Shops role**

  The Shops role is the instance of the Commerce Engine that serves traffic from one or more storefronts. This role can scale to support demand, and is usually installed in close proximity to the Sitecore XP instances that generate the traffic.

- **Minions role**

  The Minions role is an instance of the Commerce Engine that runs independently and supports asynchronous processing (including any post-order capture processing as well as any cleanup. The Minions role is usually set up as a worker role (not a web role) and doesn't receive any web traffic. This role contains a series of minions, each one with a specific task (for example, a pending order minion that is used to process the pending orders queue).

- **DevOps role**

  The DevOps role is an instance of the Commerce Engine that is internal and only available to DevOps personnel. This role can be assigned higher privileges allowing DevOps personnel to perform maintenance tasks that are not permitted to other roles (for example, bootstrapping and environment initialization functions).

1.2.4 Plugins

The Commerce Engine includes a pluggable framework for extending or modifying existing functionality. The Sitecore XC plugin architecture provides extensibility points for custom functionality, without compromising upgradeability.

This plugin architecture allows for opt-in complexity and progressive enhancement of your Sitecore XC solution. You can write plugins for connectors to integrate third party systems. You can also write plugins to extend business logic, or to extend the Business Tools.

1.2.5 Databases

For Sitecore XC, there are two databases: the shared environments database and the global database.

The shared environments database is the main data store. This database stores all of the commerce data used on the site (for example, catalog data, customer records, pricing information, configured promotions), as well as the generic entities and lists that power the functionality of the various installed plugins.

The global database stores all the global configuration data that governs how the engine roles function. The global database stores all of the environment and policy data when you execute the bootstrap function (that is, the configuration stored in the JSON files).

Chapter 2 Installation

This chapter describes the system requirements and the software packages necessary for a Sitecore XC deployment.

Instructions for on installing Sitecore XC in a single-server on-premises deployment are available in the *Sitecore Experience Commerce Installation Guide for On-Premise Solutions*.

Instructions for on installing Sitecore XC in a cloud-based deployment are available in the *Sitecore Experience Commerce Installation Guide for the Azure App Service*.

For a list of system requirements for each Sitecore XC 9.0.x release, please see the *Sitecore Experience Commerce compatibility* Knowledge Base article.

2.1 Software packages overview

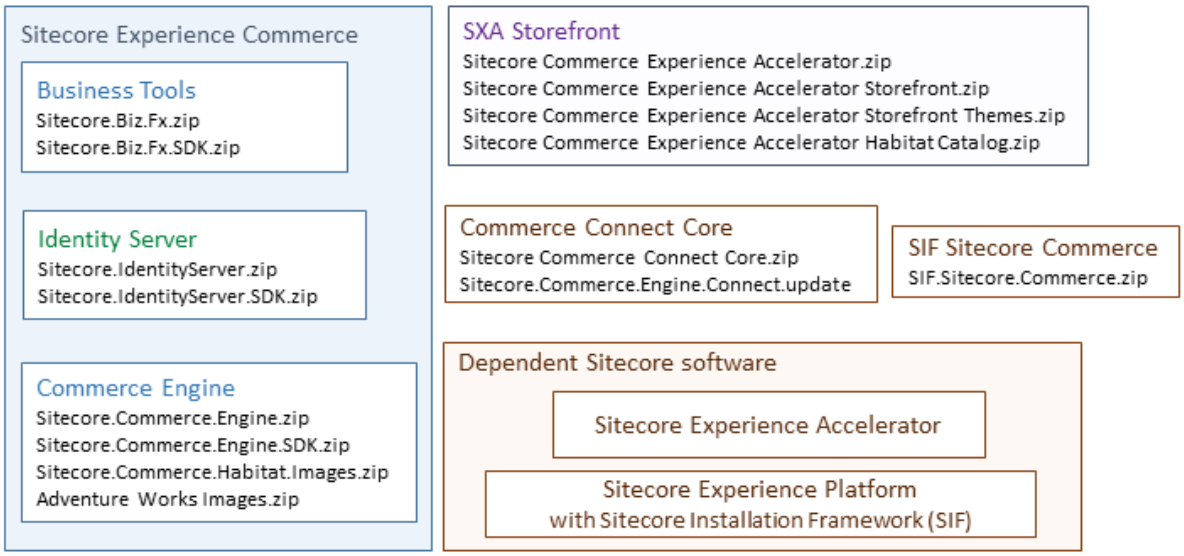
The Sitecore XC installation includes packages for Sitecore XC-specific components, as well as SXA Storefront packages.

There are separate release packages for a Sitecore XC on-premises deployment, and a Sitecore XC cloud-based deployment.

2.1.1 Commerce packages (on-premises installation)

The Sitecore XC release package does not include any Sitecore XP software. All Sitecore XP pre-requisite software must be installed first.

The following figure provides a logical view of the Sitecore XC packages for an on-premises deployment.



The following tables lists the software packages provided with your Sitecore XC release package for an on-premises installation. Each package has a unique version number and a `.zip` extension.

Package	Description
SIF.Sitecore.Commerce	Contains Commerce-specific extensions to the Sitecore Installation Framework (SIF), including the master deployment script for Commerce packages, and Commerce configuration.
Sitecore Commerce Connect Core	Contains a middleware integration layer between the Sitecore XC back-end and the front-end Storefront, and with the Sitecore Experience Platform. Contains generic APIs that are not specific to the Sitecore XC solution, enabling this connection layer to also be used when integrating other third party commerce systems with the Sitecore Experience Platform.
Sitecore.Commerce.Engine	Contains the binary for the Commerce Engine, a lightweight, micro-service based framework for the development of commerce solutions.
Sitecore.Commerce.Engine.SDK	Contains the development kit for compiling the Commerce Engine. Compiling the SDK programmatically fetches Commerce Engine plugins from Sitecore's public NuGet feed. This NuGet feed hosts plugins that are released and supported with the Sitecore XC 9.0 release.

Sitecore.Commerce.Engine.Connect	Contains a thin integration layer for integrating the Sitecore Commerce Engine with Sitecore Commerce Connect Core. The package file format is a Team Development for Sitecore (TDS) .update package
Sitecore.Biz.FX	Contains an integrated suite of Sitecore XC Business Tools, built on the Angular application platform version 4.
Sitecore.Biz.FX.SDK	Contains the development kit for compiling the Sitecore XC business tools.
Sitecore.IdentityServer	Contains the binary for the Identity Server web application, used by the Business Tools to authenticate to the Sitecore database for authorization in accessing the Commerce Engine.
Sitecore.IdentityServer.SDK	Contains the development kit for building the Identity Server web application.
Sitecore Commerce Experience Accelerator	Contains Commerce-specific extensions to the Sitecore Experience Accelerator's (SXA) templated UX layouts (for example, UI renderings used to display a catalog in the Storefront).
Sitecore Commerce Experience Accelerator Storefront	Contains the sample and starter storefront as an integrated part of the Sitecore XC solution, and built using Sitecore's new SXA UX layouts.
Sitecore Commerce Experience Accelerator Storefront Themes	Contains the themes used for the SXA Storefront site.
Sitecore Commerce Experience Accelerator Habitat Catalog	Contains the Habitat sample catalog for the Storefront site.
Sitecore.Commerce.Habitat.Images	Contains images for the Habitat sample catalog.
Adventure Works Images	Contains images for the Adventure Works sample catalog.

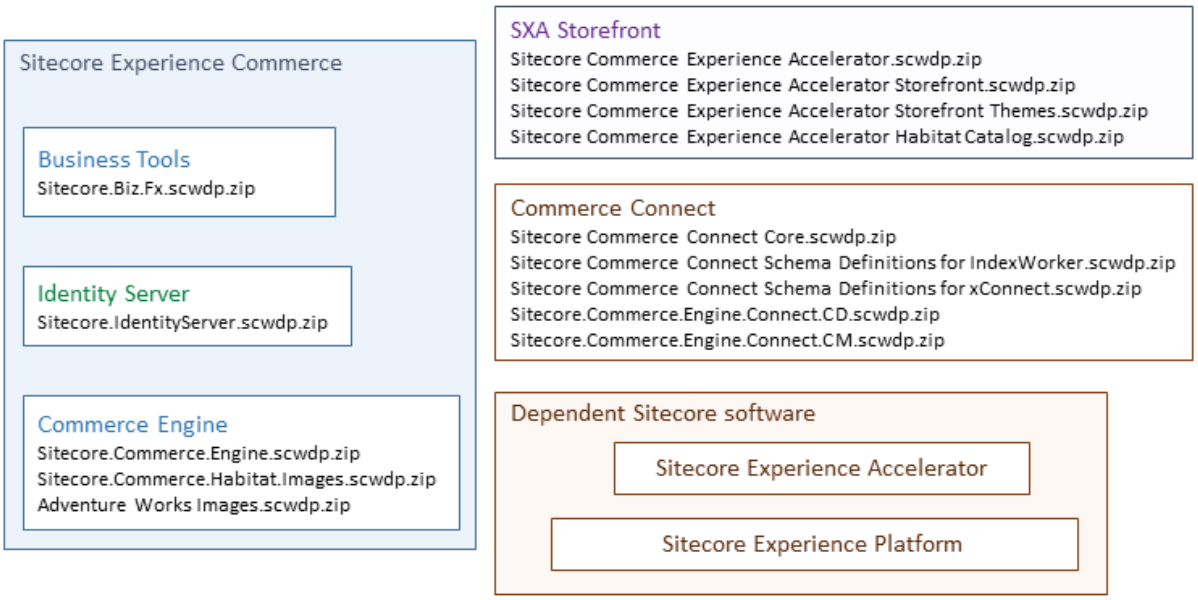
2.1.2 Commerce packages (Azure App Service installation)

The Sitecore XC Azure release package contains Web Deploy Packages (WDPs) for Sitecore XC-specific components. The Azure release package does not include any Sitecore XP software.

The Sitecore XP and Sitecore Experience Accelerator (SXA) WDPs are available separately. You can download Sitecore XP WDPs from the Sitecore Experience Platform Download page.

You can download Sitecore Experience Accelerator (SXA) WDPs from the Sitecore Experience Accelerator Download page.

The following figure provides a logical view of the Sitecore XC packages for a cloud-based deployment.



The following tables lists the Web Deploy Packages (WDPs) required for your Sitecore XC cloud installation. Each package has a unique version number and a `.scwdp.zip` extension.

Package	Description
Sitecore.BizFx	Contains an integrated suite of Sitecore XC business tools, built on the Angular application platform version 4.
Sitecore.IdentityServer	Contains the binary for the Identity Server, used by the Business Tools to authenticate to the Sitecore database for authorization in accessing the Commerce Engine.
Sitecore.Commerce.Engine	Contains the binary for the Commerce Engine, a lightweight, micro-service based framework for the development of commerce solutions.
Sitecore.Commerce.Habitat.Images	Contains images for the Habitat sample catalog.
Adventure Works Images	Contains images for the Adventure Works sample catalog.
Sitecore Commerce Connect Core	Contains a middleware integration layer between the Sitecore XC back-end and the front-end Storefront, and with the Sitecore XP.
Sitecore Commerce Connect Schema Definitions for IndexWorker	Contains schema definitions for the Commerce Connect IndexWorker.
Sitecore Commerce Connect Definitions for xConnect	Contains definitions for xConnect.
Sitecore.Commerce.Engine.Connect.CD	Contains a thin integration layer for integrating the Commerce Engine with Sitecore Commerce Connect Core on the Content Delivery instance.
Sitecore.Commerce.Engine.Connect.CM	Contains a thin integration layer for integrating the Commerce Engine with Sitecore Commerce Connect Core on the Content Management instance.

Sitecore Commerce Experience Accelerator	Contains Commerce-specific extensions to the Sitecore Experience Accelerator's (SXA) templated UX layouts (for example, UI renderings used to display a catalog in the Storefront).
Sitecore Commerce Experience Accelerator Storefront	Contains the sample and starter storefront as an integrated part of the Sitecore XC solution, and built using Sitecore's new SXA UX layouts.
Sitecore Commerce Experience Accelerator Habitat Catalog	Contains the Habitat sample catalog for the Storefront site.
Sitecore Commerce Experience Accelerator Storefront Themes	Contains the themes used for the SXA Storefront site.

2.2 Firewall ports

The following firewall ports are used by default:

Functionality	Port Range	Comment
SQL Server	1433, 1024-5000	Default server port is 1433. Client ports are assigned randomly between 1024 and 5000. Details are on microsoft.com.
Solr	8983	Reference: Running Solr

Chapter 3

Configuration

3.1 Security

Security in the Sitecore XC solution is based on certificates or on specific authenticated identities. Security is enforced at the controller level, and is based on a user's Sitecore credentials. Every user must be authenticated to be able to call any controller from the Commerce Engine.

There are two ways to authenticate:

- certificate authentication
- bearer token authentication

3.1.1 Certificate authentication

Certificate authentication is used for systems going through Commerce Engine (CE) Connect, like the SXA Storefront.

The caller must provide a header named X-ARR-ClientCert in the request headers with valid certificate information. The expected certificate information (i.e., issuer, thumbprint) is stored in the Commerce Engine config.json file. The same thumbprint must be stored in the CE Connect configuration file - `Sitecore.Commerce.Engine.config`.

The following is a sample of the certificate section in the `Sitecore.Commerce.Engine` configuration file:

```
"Certificates": {
                 "Certificates": [
                {
                 "Subject": "CN=storefront.engine",
                 "IssuerCN": "CN=storefront.engine",
                 "Thumbprint": "F1D8349D784BF672B99103C1C204A57556DD263A"
                }
                ]
          }
```

3.1.2 Bearer token authentication

Bearer token authentication is used for systems calling the Commerce Engine directly, without going through CE Connect. The caller must connect to the Sitecore Identity Server, using Sitecore credentials, to obtain a token. That token is used as an authorization bearer in request headers.

The URL of the Sitecore Identity Server must be specified in the Commerce Engine's `config.json` file (in the `SitecoreIdentityServerUrl` parameter).

The Sitecore Identity Server provides two endpoints for obtaining a token:

- `GetToken` (`http://{{SitecoreIdServerHost}}/connect/token`)

  A silent mode that allows you to get a token without having to log in through the UI. Used by Postman, Console, and Deployment scripts.

- `Authorize` (`http://{{SitecoreIdServerHost}}/connect/authorize`):

  Loads the Sitecore Identity Server login page. Once the user logs in successfully with Sitecore credentials, the user is returned to their website. Used by the Commerce Business Tools.

The identity of any system calling the Commerce Engine (e.g., Postman, Commerce Business Tools) must be stored in Sitecore Identity Server's configuration file.

The following is a sample of the client configuration in the Sitecore Identity Server's `appSettings.json` file for the Postman client:

```
"Clients": [
      {
        "ClientId": "postman-api",
        "ClientName": "Postman API",
        "AccessTokenType": 0,
```

```
            "AccessTokenLifetimeInSeconds": 3600,
            "IdentityTokenLifetimeInSeconds": 3600,
            "AllowAccessTokensViaBrowser": true,
            "RequireConsent": false,
            "RequireClientSecret": false,
            "AllowedGrantTypes": [
              "password"
            ],
            "RedirectUris": [
              "https://www.getpostman.com/oauth2/callback"
            ],
            "PostLogoutRedirectUris": [
              "https://www.getpostman.com"
            ],
            "AllowedCorsOrigins": [
              "https://www.getpostman.com"
            ],
            "AllowedScopes": [
              "openid",
              "dataEventRecords",
              "dataeventrecordsscope",
              "securedFiles",
              "securedfilesscope",
              "role",
              "EngineAPI",
              "postman api"
            ]
          }
        ]
```

The following is a sample of the client configuration in the Sitecore Identity Server's `appSettings.json` file for the Commerce Business Tools:

```
    "Clients": [
          {
            "ClientId": "CommerceBusinessTools",
            "ClientName": "CommerceBusinessTools",
            "AccessTokenType": 0,
            "AccessTokenLifetimeInSeconds": 3600,
            "IdentityTokenLifetimeInSeconds": 3600,
            "AllowAccessTokensViaBrowser": true,
            "RequireConsent": false,
            "RequireClientSecret": false,
            "AllowedGrantTypes": [
              "implicit"
            ],
            "RedirectUris": [
              "http://localhost:4200",
              "http://localhost:4200/?"
            ],
            "PostLogoutRedirectUris": [
              "http://localhost:4200",
              "http://localhost:4200/?"
            ],
            "AllowedCorsOrigins": [
              "http://localhost:4200/",
              "http://localhost:4200"
            ],
            "AllowedScopes": [
              "openid",
              "dataEventRecords",
              "dataeventrecordsscope",
              "securedFiles",
              "securedfilesscope",
              "role",
              "EngineAPI"
            ]
```

Additional security highlights include:

- HTTPS:// and SSL support
- No credit card storage option
- PCI Level 1 DSS 2.0 Certified Tokenization

- Strong password enforcement
- 90-day forced administrator password changes
- Back office geographical and proximity real-time validations
- Back office IP restriction access

3.2 User roles and permissions

Access and permissions to the Sitecore XC Business Tools are controlled by roles. You create users and assign roles using the **User Manager** tool on the **Sitecore Launchpad**, as described on the Documentation Portal.

Note
Every Sitecore XC user who needs access to the Commerce Business Tools must be assigned to the *Commerce Business User* role, at a minimum.

The following table lists the pre-defined roles and associated permissions for the Sitecore XC Business Tools.

User role	Business Tools permissions
Commerce Business User	This role is required to access the Business Tools dashboard and is assigned to all Commerce users by default. However, if this is the only role a user is assigned, they cannot access any of the Business Tools.
Commerce Administrator	Has permissions for all the Business Tools and can perform all Business Tools – related actions.
Customer Service Representative	Has permissions for **Customers** and **Orders**. A Customer Service Representative can edit and deactivate customer accounts and add, delete, or edit items, put on hold, add additional payments, or cancel customer orders.
Customer Service Representative Administrator	Includes all the permissions that are available with the Customer Service Representative role, with additional permissions for approving refunds on orders and Return to Merchandise Authorizations (RMAs).
Merchandiser	Has permissions for **Merchandising**, **Inventory**, and **Relationship Definitions**. A Merchandiser can create and edit catalogs, categories, sellable items, relationships between them, and manage inventory information.
Pricer	Has permissions for **Pricing**. A Pricer can create and edit price books, price cards, and pricing snapshots.
Pricer Manager	Includes all the permissions that are available with the Pricer role, with additional permissions for approving snapshots.
Promotioner	Has permissions for **Promotions**. A Promotioner can create promotions, including setting qualifications and benefits, and creating coupon codes to be used for marketing campaigns.
Promotioner Manager	Includes all the permissions that are available with the Promotioner role, with additional permissions to approve or reject promotions.
Relationship Administrator	Has permissions for **Relationship Definitions**. A Relationship Administrator can create cross-sell and upsell relationships between catalogs, categories, and sellable items.

In a single server deployment, you can create accounts and user groups on the machine where Sitecore XC is installed. In a multi-server deployment, you create these accounts and groups on the domain controller.

For more information about role ranagement, see the Create and set up a role and Assigning access rights topics.

3.3 Sitecore XC SDKs

The Sitecore XC solution includes SDKs for the following:

- Commerce Engine
- Identity Server

3.3.1 Compile the Commerce Engine SDK Engine

You can customize the Sitecore XC Engine, using the SDK package provided in the release package.

When you make changes to the Sitecore XC Engine (using the SDK), you must re-compile the Commerce Engine.

To compile and package the Sitecore XC Engine using the command line method (recommended):

1. Start a command prompt as administrator and go to the Sitecore Commerce Engine SDK root directory (that is, where you extracted the SDK zip file).
2. Run the following command:

   ```
   dotnet.exe restore Customer.Sample.Solution.sln
   ```

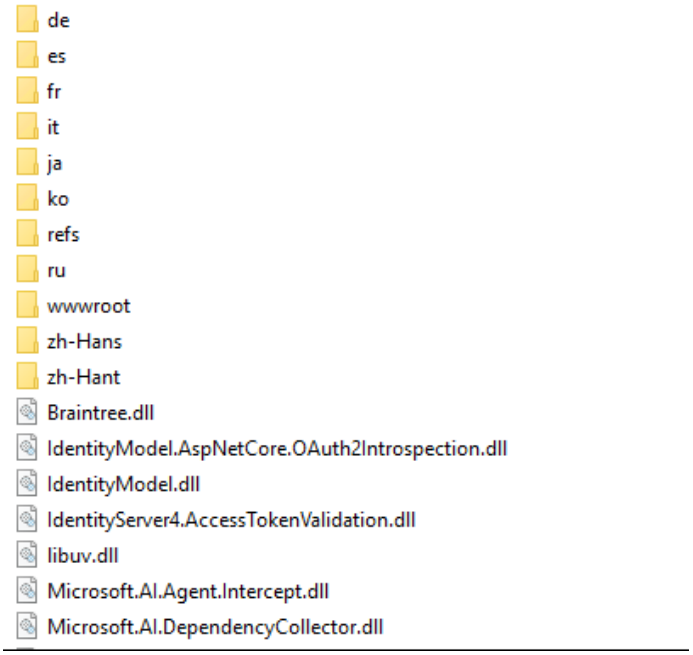
   Note
   The default `dotnet` path is "`C:\Program Files\dotnet\dotnet.exe`". You must specify the solution name because there are two .sln files under the Engine SDK root directory.

3. Run the following command:

   ```
   dotnet.exe publish Customer.Sample.Solution.sln -o <output-dir>
   ```

   Note
   Make sure that the installed version of the .NET Framework is the DevPack/SDK edition, not the Runtime edition. The `dotnet publish` command fails if only the Runtime edition is installed.

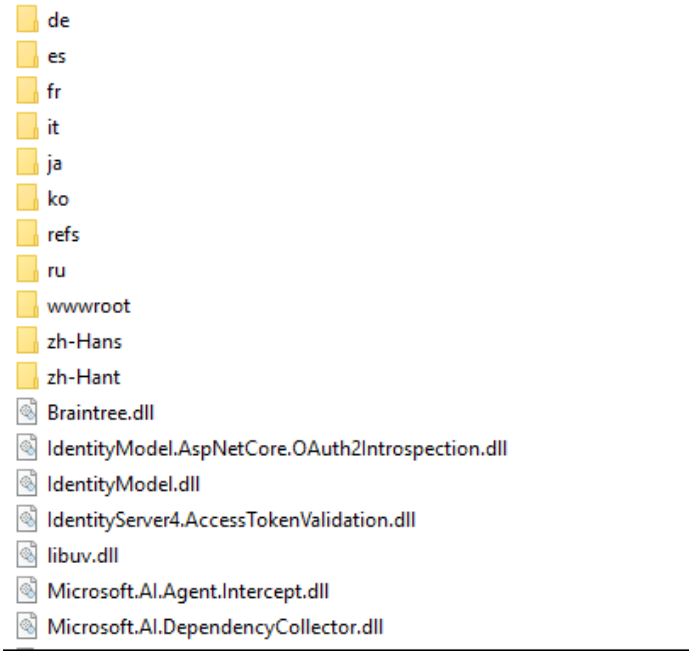
4. Zip the contents of the output folder (where you published the `Customer.Sample.Solution`. Make sure that the root of the .zip file contains the compiled DLLs and folders:

5. Copy the zipped `Sitecore.Experience.Commerce.Engine.*.zip` file to your deployment root.

To compile and package the Sitecore XC Engine using Visual Studio:

1. Launch Visual Studio.
2. From the *Sitecore Commerce Engine SDK* folder, open the `Customer.Sample.Solution.sln` file.
3. To build the solution, on the menu bar, click **Build**.
4. To publish the `Sitecore.Commerce.Engine` project to your file system, click **Publish**.
5. Zip the contents of the output folder (where you published the `Customer.Sample.Solution`. Make sure that the root of the .zip file contains the compiled DLLs and folders:



6. Copy the zipped `Sitecore.Experience.Commerce.Engine.*.zip` file to your deployment root.

3.3.2 Compile the Sitecore Identity Server SDK

You can customize the Sitecore Identity Server, using the SDK package provided in the release package.

You can compile the Sitecore XC Identity Server from a command line (recommended) or from Visual Studio 2017.

To compile and package the Sitecore Identity Server using a command line (recommended):

1. Start a command prompt as administrator and go to the *IdentityServer SDK* root directory (that is, where you extracted the Identity Server SDK zip file).
2. Run the following command:

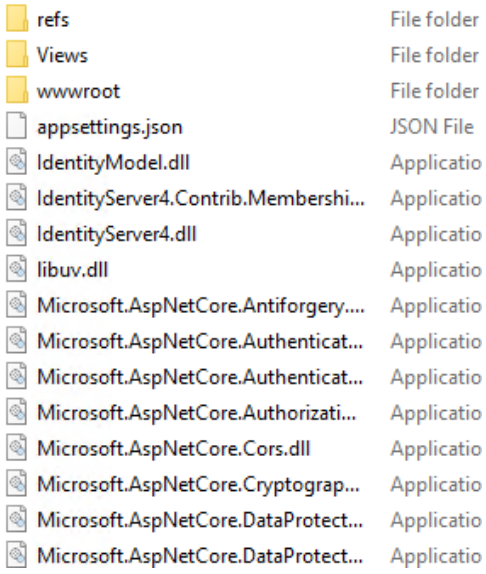
   ```
   dotnet.exe restore
   ```

3. Run the following command:

   ```
   dotnet.exe publish -o <output-dir>
   ```

   Note
   Make sure that the installed version of the .NET framework is the DevPack/SDK edition, and not the Runtime edition. The `dotnet publish` command fails if only the Runtime edition is installed.

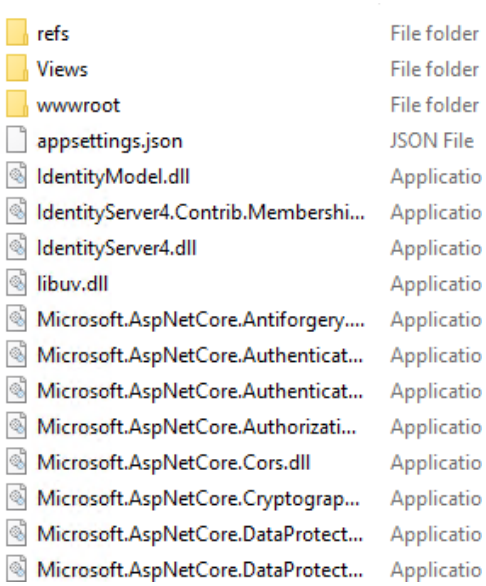
4. Zip the contents of the folder where you published the `Sitecore.IdentityServcer.sln`. Make sure that the root of the `.zip` file contains the compiled DLLs and folders, as shown in the following:



5. Copy the zipped `Sitecore.IdentityServer.*.zip` file to your deployment root.

To compilie and package the Sitecore Identity Server using Visual Studio:

1. Launch Visual Studio.
2. From the *Sitecore IdentityServer SDK* folder, open the `Sitecore.IdentityServer.sln` file.
3. To build the solution, on the menu bar, click **Build**.
4. To publish the project to your file system, click **Publish**.
5. Zip the contents of the folder where you published the `Sitecore.IdentityServcer.sln` file. Make sure that the root of the `.zip` file contains the compiled DLLs and folders, as shown in the following:



6. Copy the zipped `Sitecore.IdentityServer.*.zip` file to your deployment root.

3.4 Bootstrap the Commerce Engine

Whenever you make changes to any environment configuration files, you must run the Bootstrap operation on the Commerce Engine to ensure that your changes are propagated to the global database.

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

Note
When you place a call to the Commerce Engine API from outside the Commerce Business Tools (for example, using Postman), you must disable the antiforgery protection setting in the `wwwroot\config.json` file.

To run the bootstrap operation:

1. Install Postman and launch the application.
2. Navigate to the *Sitecore Commerce SDK* folder and open the *Postman* folder.
3. Import the contents of the *Postman* foder into Postman.

   Note
   Make any necessary changes to the global variables in the environment you are using (such as ServiceHost) under the **Settings** > **Manage Environments** menu in Postman.

4. In the **Collections** pane in Postman, navigate to the *Authentication* folder.
5. Open the *Sitecore* folder and execute the *GetToken* call.

   When Postman displays an access token in the **Body** pane, authentication is successful.
6. In the **Collections** pane, navigate to the *SitecoreCommerce_DevOps* folder.
7. Open the *1 Environment Bootstrap* folder, and execute the *Bootstrap Sitecore Commerce* call.

3.5 Clean and initialize the environment

You can clean your Commerce Engine environment using the Postman samples provided in the Commerce Engine SDK.

After you have run the Postman samples to clean and initialize your environment, you must perform additional steps in the Sitecore Content Editor.

3.5.1 Clean the environment

The following instructions assume that you are using Postman to exercise the Commerce Server API, and that you have imported the Postman samples provided in the Commerce Engine SDK.

To clean the Commerce Engine environment:

1. Launch the Postman application.
2. In the **Collections** pane in Postman, navigate to the *Authentication* folder.
3. Open the *SitecoreCommerce_DevOps* folder.
4. Open the *2 Clean Environment* folder, and execute the *Clean Environment* call.
5. Open the *3 Environment Initialize* folder, and execute the *Ensure\Sync default content paths* call.
6. In the same *3 Environment Initialize* folder, execute the *Initialize Environment* call.
7. Open the Sitecore Launchpad in a browser and click on **Content Editor**.
8. In the left pane, navigate to the **Commerce** > **Catalog Management** > **Catalogs** window.
9. In the right window, uncheck the **Selected Catalog** box and save your changes.
10. Re-check the **Selected Catalog** box and save your changes again.

3.5.2 Republish the site and rebuild the search indexes

You must re-publish the site re-index the master and web indexes for the Sitecore XC site. You can perform both of these operations from the **Control panel** on the **Sitecore Launchpad**.

To re-publish and re-index the site:

1. In the **Content Editor**, click on the **Publish** tab.
2. Click **Publish** and select **Publish site** from the drop-down list.
3. In the **Publish Site** window, select **Republish – publish everything** and click **Publish**.
4. Return to the **Sitecore Launchpad** and click on **Control Panel**.
5. In the Control Panel, click on **Indexing Manager**.
6. In the Indexing Manager window, select the following indexes:
   - sitecore_master_index
   - sitecore_web_index
7. Click on **Rebuild** at the bottom of the window to initiate the re-indexing process.

Chapter 4

Administration

This chapter contains the following sections:

- Commerce Service API
- Commerce Control Panel
- Caching
- Fault injection

4.1 Commerce Service API

The Service API provides the ability for external entities to execute functions within Sitecore XC. The API is RESTful based on Odata. Odata provides built-in benefits, such as supporting annotations on metadata, which enables clients to pre-validate and support enumerable properties.

Service API commands can be short or long-running. Commands are request/response oriented.

The API can be extended by adding commands via a Commerce plugin.

The Sitecore Commerce Service API is made up of three role-oriented APIs, each targeting specific logically separated business needs:

- CommerceOps API – for the DevOps role in managing a Sitecore XC implementation, including methods to create and manage environments and global policies.
- CommerceShops API – for supporting the online shopping experience such as a web storefront. For example, when adding to a cart in the Storefront, the related processor in Commerce Connect makes an API call to the Commerce Engine to perform the cart activity and return a result.
- CommerceAuthoring API – for business users. It introduces the Entity Views concept, which is a mechanism for mapping potentially complex Sitecore Commerce entities into a flattened structure that can be dynamically represented in business tools.

4.1.1 API headers

Sitecore XC uses headers passed in via the Service APIs to establish context for the call. These headers are available from within the CommerceContext, which is passed around on all calls. There are standard known headers that are supported upon installation, and additional headers can be established for specific plugins. Plugins may reference these headers when performing actions.

Some headers are only relevant in certain actions. However, it is not harmful to include a header even if it is not used in a particular call. Supported headers include:

Name	Usage
ShopName	The name of the current Shop that you are accessing.
ShopperId	A unique ID for a Shopper. This is normally a GUID, represented as a string.
CustomerId	If the Shopper is registered, then an additional ID is passed in, representing the unique ID for a registered customer. This is normally a GUID, represented as a string
Currency	The currency that is desired in the response. For example, if you request a SellableItem, you will get pricing in the desired currency (if available).
Language	The language desired in the response. For example, if you request a SellableItem, you will get localized strings in the desired language (if available).
EffectiveDate	The effective date to use during any date-based calculations. This allows time-travel scenarios where you wish to see results as if the interactions occurred at dates and times not in the present. If an EffectiveDate is not passed in, then DateTime.UtcNow is used.
Latitude	A string representing the current latitude of the Shopper.
Longitude	A string representing the current longitude of the Shopper.
IpAddress	The IP address of the Shopper.

Name	Usage
Roles	A "|" delimited list of roles for the caller. This may influence what actions are allowed, what information is returned in queries, or if the query can be performed. For security, when you are authenticated via Federated Authentication, these roles are ignored and the roles are derived directly from the authentication token.
IsRegistered	A boolean represented as a string that indicates the current caller is registered.

4.1.2 CommerceOPS API actions

CommerceOps API actions include:

Action	Description
Bootstrap	Completely reloads metadata for environments from a source directory (wwwroot/data/environments). For security, this should be done from an internal instance of the Commerce Engine (CommerceOps). Metadata includes global and environment specific policies and policy sets. Note that this does not add or remove any artifacts to the environment. Bootstrapping only updates metadata. Sample: `http://{{ServiceHost}}/{{OpsApi}}/Bootstrap()`
CleanEnvironment	Removes all artifacts from a Commerce environment. The environment to clean is passed as a header called "Environment". Passing in an empty environment completely clears the global environment, which is where metadata for all the environments is stored. Sample: `http://{{ServiceHost}}/{{OpsApi}}/CleanEnvironment()`
InitializeEnvironment	Initializes a specified Commerce environment from an empty EntityStore, and adds needed artifacts. Sample: `http://{{ServiceHost}}/{{OpsApi}}/InitializeEnvironment(environment='{{Environment}}')`
GetEnvironment	Retrieves a single named Commerce environment. Sample: `http://{{ServiceHost}}/{{OpsApi}}/Environments('Entity-CommerceEnvironment-Habitat')`
ImportEnvironment	Imports a Commerce environment from a raw exported environment JSON string. This imports and initializes the metadata for the specified environment only. Sample: `http://{{ServiceHost}}/{{OpsApi}}/ImportEnvironment()`
ExportEnvironment	Exports a Commerce environment to JSON. Sample: `http://{{ServiceHost}}/{{OpsApi}}/ExportEnvironment(environmentName='{{Environment}}')`
ListEnvironments	Retrieves a list of all known Commerce environments. Sample: `http://{{ServiceHost}}/{{OpsApi}}/Environments`

Action	Description
GetApiRoutes	Retrieves a list of the current API routes. Sample: `http://{{ServiceHost}}/{{OpsApi}}/GetApiRoutes`

Actions pertaining to the CommerceShops and CommerceAuthoring APIs are described in the Sitecore Experience Commerce Developer's Guide.

4.2 Commerce Control Panel

The Commerce Control Panel is the central location in the Sitecore XP tool suite for configuring Sitecore Commerce settings for the Engine, business tools, storefronts, and system messages.

You can edit the Commerce Control Panel content in the Content Editor. Click the **Content Editor** icon on the Sitecore Launchpad, and then expand the *Commerce* folder in left navigation pane.

The **Commerce Control Panel** has three major sections.

- **Shared Settings** – defines the options for a number of configuration settings applicable across Commerce entities including the Engine, business tools, and storefronts.
- **Commerce Engine Settings** – defines Commerce Terms applicable to the Commerce Engine and its Business Tools.
- **Storefront Settings** – defines settings applicable to one or more Storefront instances; the settings are selected from the options defined in the Shared Settings folder.

For more information see the *Commerce Engine Settings* and *Storefront Configuration Settings* topics on doc.sitecore.net.

4.3 Caching

Sitecore XC uses the core libraries of the Sitecore caching framework and Entity Memory Caching.

A cache store may be defined for each Commerce Environment. This allows caching to be cleared for a specific environment without affecting others. Cache stores are instantiated when the environment starts up. The cache store for an environment includes the following caches:

- `CommerceTerms` – caches CommerceTerms from the Control Panel.
- `Customer.All` – caches loaded Customers, if this cache is enabled.
- `EntityCache` – caches loaded CommerceEntities.
- `FulfillmentOptions` – caches the list of Fulfillment Options from the Control Panel.
- `ItemModel` – caches loading of the ItemModel.
- `LocalizedMessages` – caches the loading of localized messaging from Sitecore Items in the Control Panel.
- `PaymentOptions` – caches the list of Payment Options from the Control Panel.
- `PolicySet` – caches any PolicySets that are loaded.
- `Promotions.All` – caches loaded Promotions, if this cache is enabled.
- `SitecoreItemsByPath` – caches a call to Sitecore to retrieve all the items in a particular path.
    - `ViewTerms` – caches the ViewTerms from the Control Panel.

Caching is administered using the CommerceOps API. API actions call `GetCacheStoresCommand`, which in turn calls `GetCacheStoresPipeline`.

The API actions on cache stores include:

- `Get Cache Stores` – retrieves a list of all instantiated cache stores.

  For example (GET): `http://{{ServiceHost}}/{{OpsApi}}/GetCacheStores()`

- `Get Cache Store` – retrieves a specified cache store.

  For example (GET):

  `http://{{ServiceHost}}/{{OpsApi}}/GetCacheStore(name='{{CacheStoreName}}')`

- `Clear Cache Store` – clears a specified cache store.

  For example (PUT):

```
http://{{ServiceHost}}/{{OpsApi}}/ClearCacheStore()
Body:
{
  "cacheStoreName": "{{CacheStoreName}}"
}
```

Entity Memory Caching adds the ability to cache entities in memory. For any given entity in the system, caching policies define if an entity can be cached in memory, its caching priority, and its caching expiration.

Property	Type	Description	Default	Required
EntityFullName	string	The fully qualified name of the entity.	""	Y
AllowCaching	bool	Whether the entity may be cached in memory.	True	N
Priority	string	The priority for the entity in the cache (Normal/High).	Normal	N
Expiration	long	Time until the entity expires from the cache, in milliseconds.	60000	N

4.4 Fault injection

Fault injection provides the ability to deliberately cause a service to fail, in order to test failure in a complex distributed solution. Sitecore XC has a single fault injection pattern, which covers the concurrency fault in the persistence of a Commerce Entity. Perform the following tasks to inject the defined concurrency fault:

- Modify the project.json file for the Sitecore.Commerce.Engine project by adding a reference to the Sitecore.Commerce.Plugin.FaultInjection.
- Add a Header to the request named `FaultInjection`.
- Assign a value of `ConcurrencyFault` to the Header you just added.
    - Invoke any API method that persists a Commerce Entity.

At this point, you should have a concurrency fault raised.

The following is a sample result from calling the `AddCartLine` using Postman with the fault injection header inserted:

Headers raw view:

```
Content-Type:application/json
ShopName:{{ShopName}}
ShopperId:{{ShopperId}}
Language:{{Language}}
Currency:{{Currency}}
Environment:{{Environment}}
IpAddress:{{IpAddress}}
CustomerId:{{CustomerId}}
Roles:{{Roles}}
FaultInjection:ConcurencyFault
```

Response body:

```
"ResponseCode": "Error",
 "Messages": [
 {
  "MessageDate": "2016-09-01T12:12:45.0287079-05:00",
  "Code": "Error",
  "Text": "SQL.UpdateEntity.Fail:Id=Cart01|Try=1|Environment=Entity-CommerceEnvironment-
   AdventureWorks|Message='Concurrency error: The Entity version supplied (1) is lower or
equal to the current version
   (1).'|Number=50000|Procedure='sp CommerceEntitiesUpdate'|Line=22",
  "CommerceTermKey": "EntityPersistException"
 }
```

Chapter 5 Production deployments

This chapter contains information on production deployments for on-premises solutions and includes the following sections:

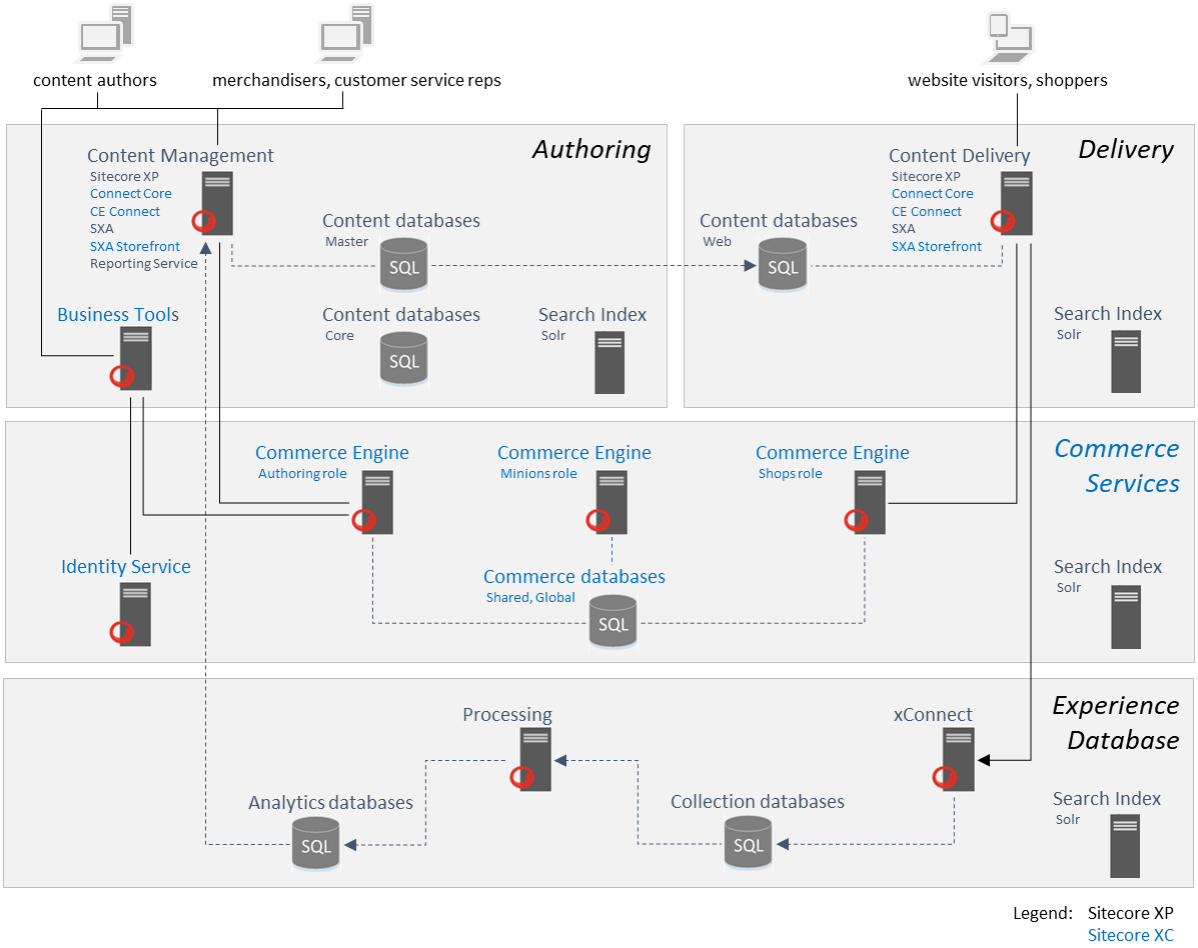
- Reference architecture

5.1 Reference architecture

The following figure shows the Sitecore XC distributed production deployment reference architecture, applicable for an on-premises deployment or on hosted virtual machines.

The diagram shows the main Sitecore XP and Sitecore XC services and some of the main connections between them. In the context of a Commerce deployment, "Sitecore XP" is the entire Sitecore plaform including Experience Mansagement, Experience Platform, xDB, and xConnect.

The diagram identifies the services that are typically bundled together on the same physical resources versus those that are physically separated. This distribution of physical resources is considered the minimum in a typical production deployment.



The reference architecture for the Sitecore XC solution comprises four broad logical partitions:

- **Authoring**: comprises Sitecore XP's core services for creating, managing and publishing content. In a Commerce context, this includes hosting the Storefront and Business Tools for internal access.
- **Delivery**: comprises Sitecore XP's core services for displaying web content to visitors, and executing in-session personalization rules. In a Commerce context, this includes hosting the Storefront for visitor access.
- **Commerce Services**: comprises Sitecore XC's Engine processing roles.
- **Experience Database**: comprises Sitecore XP's services and storage roles that collect and store experience data.

Service	Description
Content Management (CM)	A Sitecore XP internal-facing web server that hosts client applications used by internal content authors for creating, managing and previewing content. In a Commerce production deployment context, the CM hosts: - an instance of the Storefront and the dependent SXA framework for internal users to access external shopper data and other internal information. - internal connectors including Commerce Connect Core, and Commerce Engine Connect (CE Connect), which enable the Commerce Engine to connect with Content Management and the Storefront. The reporting service is also deployed on the CM, which displays analytics data to internal users.
Content Delivery (CD)	A Sitecore XP external-facing web server for displaying web content to visitors, and for executing in-session personalization rules. In a Commerce production deployment context, the CD hosts: - instance(s) of the externally-facing Storefront. - the Conect Core and CE Connect internal connectors.
Content databases	Sitecore XP databases including: - Master: store unpublished content managed via the CM; these belong in the authoring network. - Web: store the latest version of published Sitecore content; as part of the delivery network it serves its data to the CD. - Core: store configuration information about Sitecore user interfaces and accounts; they serve both the CM and CD.
Commerce Engine	Deployed instances of core Commerce processing, each serving distinct logical roles: - Shops: serves external requests from externally-facing Storefront(s). - Authoring: serves internal requests from the Business Tools. - Minions: a worker role that performs asynchronous background processing. - DevOps: a maintenance role for internal DevOps users (not shown in diagram).
Commerce databases	Two logically separated storage databases including: - Shared Environments: stores the Commerce data used on the Storefront(s). - Global: stores the configuration data that govern how the Engine roles function.
Business Tools (BizFX)	Business tools interface used by internal users.They communicate directly with the Commerce engine services via a dedicated Commerce EntityView core API.
Identity Server	Sitecore's federated authentication service provider, for authenticating to the Business Tools and Commerce Engine.
xConnect	Collects visitor contact and interaction data from the CD.
Collection databases	Sitecore XP databases that store visitor contact and interaction data, for example: contacts, devices, locations, actions, engagement automation states.
Processing	Processes the real-time customer interaction data from Collection databases; aggregates and stores them in the Analytics databases for internal reporting.
Analytics databases	Stores analytics data from the Processing server.
Search indexes	Allow internal users to perform searches (for example, catalog items, orders, and customers).