

Sitecore on Sitecore

[Learn more](#)

NAVIGATING YOUR CONTENT MIGRATION

Author:	Derek Hunziker, Director, Digital Technology
Date:	October 9, 2024

A Guide to Migrating Content from Sitecore XM™ to XM Cloud™

On its face, migrating content from Sitecore XM™ or Sitecore XP™ to XM Cloud™ can appear to be a complex process, but with the right approach, it becomes much more manageable. Whether you're aiming to take advantage of XM Cloud's headless architecture, improve performance, or streamline content delivery, this guide will help you migrate your valuable content into XM Cloud using the best migration strategy for your needs.

This guide focuses on three primary migration approaches: **Sitecore Packages**, **Sitecore PowerShell Extensions (SPE)**, and the **Sitecore XM Cloud Migration Assistant**. Each method has its strengths and limitations, depending on your project's scale, customization requirements, and timeline. We'll explore how each approach works, provide practical examples, and highlight the pros and cons to help you make an informed decision.

Contents:

Sitecore Packages.....	2
Sitecore PowerShell Extensions (SPE).....	3
Sitecore XM Cloud Migration Assistant.....	6
Conclusion.....	7

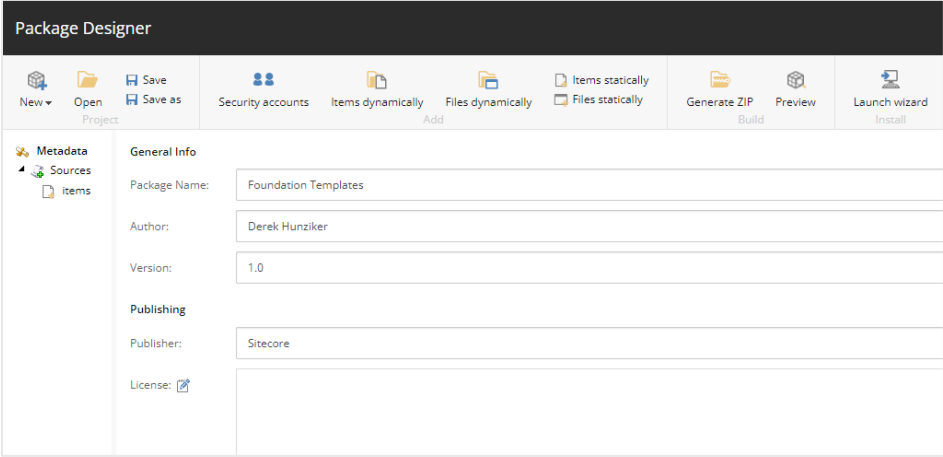
Sitecore Packages

As of the release of this guide, the Sitecore Package Designer remains supported in XM Cloud. This type of package is essentially a .zip file containing serialized items that are created using the Package Designer utility and installed via the package Installation Wizard. While newer serialization and packaging options exist in the form of [Sitecore Content Serialization \(SCS\) Packages](#), this guide focuses on Package Designer for its simplicity, ease of use, and compatibility with older Sitecore versions.


Steps

1. Access your XM environment.
2. From the Desktop view, access the Package Designer under the Development Tools menu.
3. Use the Package Designer utility to package up items, starting from base templates (e.g., Foundation layers).
4. Work your way up the dependency chain to include content items (e.g., pages).
5. Install the packages via the Installation Wizard in the target XM Cloud environment.

Example



The screenshot shows the Sitecore Package Designer interface. The title bar reads "Package Designer". The top navigation bar includes icons for "New", "Open", "Save", "Save as", "Security accounts", "Items dynamically", "Files dynamically", "Items statically", "Files statically", "Generate ZIP", "Preview", and "Launch wizard". Below the navigation bar, there is a sidebar on the left with "Metadata", "Sources", and "Items". The main area is titled "General Info" and contains the following fields:

Package Name:	Foundation Templates
Author:	Derek Hunziker
Version:	1.0
Publishing	
Publisher:	Sitecore
License:	

Create separate packages for Foundation, Feature, and Project layers, ensuring that all dependencies are properly packaged. Once all dependencies are migrated, package and install page items.

Pros

- **Simplicity:** Easy to use graphical interface for cherry-picking items.
- **Incremental:** Packages can be saved and updated over time, allowing for gradual migration.

Cons

- **Manual effort:** Requires manual packaging, which may lead to missed items or human error.
- **No modification:** Does not allow for refactoring during migration; it merely transfers items.
- **Slow:** Manually creating packages and installing them is a time-consuming process.

Sitecore PowerShell Extensions (SPE)

Sitecore PowerShell Extensions (SPE) provide access to Sitecore APIs for manipulating items programmatically. By surfacing the XM master database in XM Cloud, you can perform Create, Read, Update, and Delete (CRUD) operations between both databases, allowing for more flexible migration scenarios. This is particularly useful in scenarios where template schemas and field names differ between XM and XM Cloud.

Steps

1. Attach the XM master database to XM Cloud using a configuration patch file (see example below)
2. Use SPE scripts to perform CRUD operations, such as reading or iterating content from the old master database while creating items in the new database.
3. Migrate content between environments while performing any necessary transformations to the content. For example, a typical migration script may contain conditional logic to read from one or more fields from the XM master database. It's also common to traverse child items and/or datasources to extract the necessary content from XM before migrating it into XM Cloud.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:env="http://www.sitecore.net/xmlconfig/env">
  <sitecore env:require="!Development">
    <eventing defaultProvider="sitecore">
      <eventQueueProvider>
        <eventQueue name="oldmaster" patch:after="eventQueue[@name='web']"
type="Sitecore.Data.Eventing.$(database)EventQueue, Sitecore.Kernel">
          <param ref="dataApis/dataApi[@name='$(database)']" param1="$(name)" />
          <param hint="" ref="PropertyStoreProvider/store[@name='$(name)']" />
        </eventQueue>
      </eventQueueProvider>
    </eventing>
    <PropertyStoreProvider>
      <store name="oldmaster" patch:after="store[@name='web']" prefix="preview"
getValueWithoutPrefix="true" singleInstance="true"
type="Sitecore.Data.Properties.$(database)PropertyStore, Sitecore.Kernel">
        <param ref="dataApis/dataApi[@name='$(database)']" param1="$(name)" />
        <param resolve="true" type="Sitecore.Abstractions.BaseEventManager,
Sitecore.Kernel" />
        <param resolve="true" type="Sitecore.Abstractions.BaseCacheManager,
Sitecore.Kernel" />
      </store>
    </PropertyStoreProvider>
    <databases>
      <!-- Old Master-->
      <database id="oldmaster" patch:after="database[@id='web']"
singleInstance="true" type="Sitecore.Data.DefaultDatabase, Sitecore.Kernel">
        <param desc="name">$(id)</param>
        <icon>Images/database_web.png</icon>
        <securityEnabled>true</securityEnabled>
        <dataProviders hint="list:AddDataProvider">
          <dataProvider ref="dataProviders/main" param1="$(id)">
            <disableGroup>publishing</disableGroup>
            <prefetch hint="raw:AddPrefetch">
              <sc.include file="/App_Config/Prefetch/Common.config" />
              <sc.include file="/App_Config/Prefetch/Webdb.config" />
            </prefetch>
          </dataProvider>
        </dataProviders>
        <PropertyStore ref="PropertyStoreProvider/store[@name='$(id)']" />
        <remoteEvents.EventQueue>
```

```
<obj ref="eventing/eventQueueProvider/eventQueue[@name='${id}']" />
</remoteEvents.EventQueue>
<archives hint="raw:AddArchive">
  <archive name="archive" />
  <archive name="recyclebin" />
</archives>
<cacheSizes hint="setting">
  <data>100MB</data>
  <items>50MB</items>
  <paths>2500KB</paths>
  <itempaths>50MB</itempaths>
  <standardValues>2500KB</standardValues>
</cacheSizes>
</database>
</databases>
</sitecore>
</configuration>
```

Configuration patch file for XM master database.

```
Get-ChildItem -Path "oldmaster:/sitecore/content/MyTenant/MySite/Home" -Recurse
-Language "en" | ForEach-Object {

  Write-Host "Old item path:" $_.Paths.ContentPath

  # NOTE: Apply conditions or filtering as needed

  # Create items in XM Cloud "master" database using ID from XM
  # $NewItem = New-Item -Path "master:" -Name $_.Name -ItemType [...] -ForceId
  $_.ID -Parent [...]
  # $NewItem.Editing.BeginEdit()
  # $NewItem.Fields["NewField"].Value = $_["OldField"]
  # $NewItem.Editing.EndEdit()
}
```

Creating new items based on old items.

Pros

- **Flexibility:** Allows content modification (e.g., field cleanup, merging content) during migration.
- **Granular control:** Migrate specific items, fields, or languages.

Cons

- **Advanced:** Requires in-depth knowledge of PowerShell and Sitecore APIs.
- **Not officially supported:** May not be viable for future XM Cloud versions and should be tested thoroughly in non-production environments. Although intended to be temporary for the purposes of migrating content only, modifications to XM Cloud are generally discouraged.

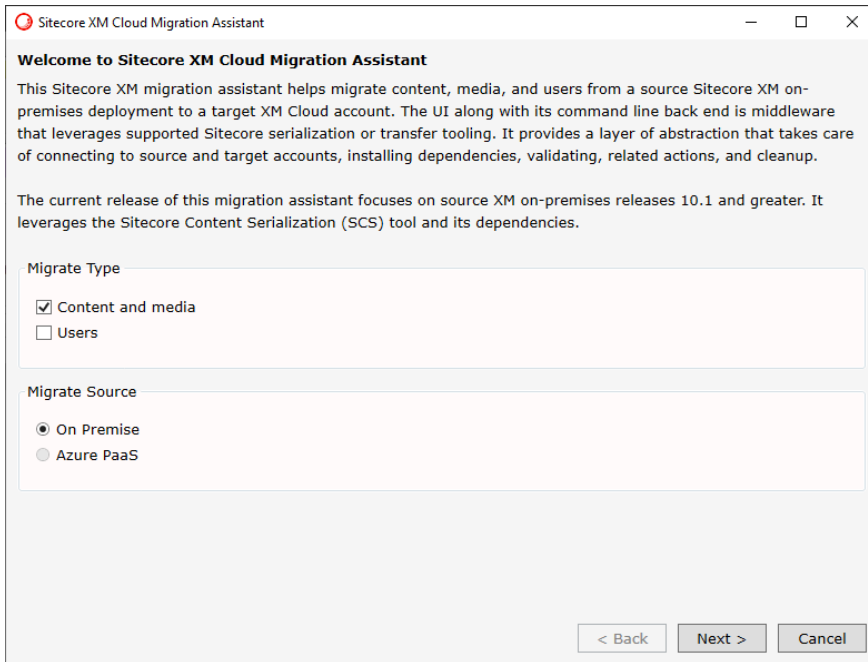
Sitecore XM Cloud Migration Assistant

The [Sitecore XM Cloud Migration Assistant](#) is based on the Sitecore CLI and Sitecore Content Serialization (SCS), while future versions aim to take advantage of Protocol Buffers for even faster performance. In its initial 1.x version, the tool supports migrating content, media, and user data from Sitecore XM releases 10.1 and later to XM Cloud, with both CLI and GUI options available.

Steps

1. Download the CLI or GUI version (linked above).
2. Establish connections between the source XM environment and XM Cloud.
3. Use either the CLI or GUI to migrate content starting with the base templates.
4. Continue migrating all content and higher-level items such as pages.

Example



Pros

- **Official tool:** Supported by Sitecore with thorough testing.
- **End-to-end:** Can migrate content, media, and user data comprehensively.
- **Speed:** Quick at getting items from XM into XM Cloud.

Cons

- **No content modification:** The initial version focuses on content transfer without the ability to modify during migration.
- **Version constraints:** Limited to migrating from XM versions 10.1 and later.

Conclusion

Each of these approaches offers distinct advantages depending on the complexity and requirements of the migration. Depending on your project's needs—whether you need the simplicity of packages, the flexibility of PowerShell scripts, or the speed and official support of the XM Cloud Migration Assistant—one of these methods will be a better fit. Be sure to test thoroughly and document each step to ensure a smooth transition to XM Cloud.