Sitecore Azure Toolkit 1.1.0

All the official Sitecore documentation.

sitecore®
Own the experience™

Configure the Bootloader module for a Sitecore deployment

The Bootloader module is a tiny module that facilitates the installation of supported Sitecore modules. You must always add the Bootloader module to the `modules` parameter of your Sitecore App Service deployment when installing other modules.

To configure the Bootloader module:

1. Locate the WebDeploy package (WDP), which is the Sitecore.Cloud.Integration.Bootload.wdp.zip) of the Bootloader module in `resources\<version to be deployed>\addons` folder of your Sitecore Azure Toolkit installation.
2. Upload the WDP to the storage account and note the URL of the package because you will need to add it to the snippet in step 4. If you are running Sitecore 8.2.7, then in step four change the parameter value to: `bootloaderMsDeployPackageUrl`

Note

Sitecore version 8.2.7 uses an ARM template that has already configured the bootloader by default, so if you are running Sitecore 8.2.7 ignore the following steps, (3, 4, and 5),

1. On Github, in the addons folder of your Sitecore version and environment configuration, locate the bootloader.json template.
2. Add the following snippet to the `modules` parameter of your `azuredeploy.parameters.json` file:

```
{
"modules": {
"value": {
"items": [
{
"name": "bootloader",
"templateLink" : "<link to the Bootloader template>",
"parameters": {
"msDeployPackageUrl" : "<link to the Bootloader WDP>"
}
}
]
}
}
}
```

3. Populate the parameters for the Bootloader module:
   - For `templateLink`, go to Github and use the URL of the bootloader.json template for your particular topology. You can also upload the template to your storage account.
   - For `msDeployPackageUrl`, use the URL of the WDP package for the Bootloader module.

Deploy a new Sitecore environment to the Azure App Service

With the Sitecore Azure Toolkit, you can deploy a new Sitecore environment to the Microsoft Azure App Service®.

This topic describes how to:

Plan your environment

The Sitecore Azure Toolkit supports the following four Sitecore configurations by default, to suit different needs:

Configuration	Description
XP0	The Sitecore Experience Platform, running as a single WebApp instance. Use this configuration for development and testing. For security and scalability reasons, it is best practice to use the XM1 or XP1 configuration in production environments.
XM	The Sitecore Experience Management configuration running both the *Content Delivery* and *Content Management* roles. Use this environment when you are not planning to use the *Analytics* and *Marketing* features of the Sitecore Experience Platform (that is, in CMS-only mode).
XP	The Sitecore Experience Platform configuration running four roles: *Content Delivery*, *Content Management*, *Processing*, and *Reporting*. Use this environment when you are planning a fully featured Sitecore Experience Platform installation.

XDB	The Sitecore Experience Database configuration running both the *Processing* and *Reporting* roles. Use this environment in combination with an on-premise Sitecore XM installation to provide the Experience Database features.

Prepare WebDeploy packages

The Azure Resource Manager (ARM) requires that WebDeploy packages (WDPs) containing the application code and resources are available for download over the Internet. One option to host WDPs is to create a Microsoft Azure® storage account.

To prepare the WDPs:

1. Ensure you have access to a Microsoft Azure subscription to deploy a Sitecore environment.
2. Create a Microsoft Azure storage account.
3. Go to the Sitecore Experience Platform Download page and download the prebuilt WDPs.
4. Use Azure Storage Explorer to connect to a Microsoft Azure storage account and upload Sitecore WDPs for your configuration. Ensure you use a *blob* type storage container.

   Note

   WDPs have the extension `.scwdp.zip` and contain the role name as part of the file name, for example, `Sitecore 8.2 rev. 161115_cm.scwdp.zip`.

5. In Azure Storage Explorer, copy the URLs for your WDPs.
6. In Azure Storage Explorer, create a Shared Access Signature (SAS) token for the storage container, and then append the token to the WDP URLs. Make a note of the package URLs for later use in ARM templates.

Obtain MongoDB connection strings

To obtain the necessary connection strings when deploying XP0 and XP1 configurations, go to your MongoDB vendor.

The strings are for the analytics database and for three tracking databases. You can choose to host MongoDB on Azure Virtual Machines or sign up for the MLab MongoDB hosting service. When the MongoDB cluster is ready, note the connection strings for the databases as you will use them later in ARM templates.

Download and configure an environment template

To download and configure the environment template for your selected Sitecore configuration:

1. Go to the Github repository for your selected Sitecore configuration.
2. Locate the environment template (*azuredeploy.json*) and download the corresponding parameters file (`azuredeploy.parameters.json`). Make a note of the URL of *azuredeploy.json*.

   Alternatively, you can download the *azuredeploy.json* template and related templates in addons and nested subfolders and upload to your Azure storage account.

3. Open the `azuredeploy.parameters.json` file in the editor and fill in the following parameters

Parameter name	Configuration	Description	Value restrictions
`deploymentId`	All	The unique ID of the deployment. This value prepends the names of all resources.	It must contain only alphanumeric characters and hyphens. The maximum length is 60 characters.
`cmMsDeployPackageUrl cdMsDeployPackageUrl`	XM and XP	Links to the WDPs for the roles.	The URLs must be accessible from Microsoft Azure data centers. Use SAS tokens, or an equivalent, to provide secure access to the resources.
`prcMsDeployPackageUrl repMsDeployPackageUrl`	XP and XDB only	Links to the WDPs for the roles.	The URLs must be accessible from Microsoft Azure data centers. Use SAS tokens or an equivalent to provide secure access to the resources.
`singleMsDeployPackageUrl`	XP0 only	Links to the WDPs for the roles.	The URLs must be accessible from Microsoft Azure data centers. Use SAS tokens or an equivalent to provide secure access to the resources.
`sitecoreAdminPassword`	All	The password for the*sitecore\admin* user when Sitecore is deployed.	The minimum length is 8 characters.
`sqlServerLogin`	All	The name of the administrator user for the virtual SQL Server for core, master, web, and reporting SQL Azure databases.	-
`sqlServerPassword`	All	The password for the administrator user for the virtual SQL Server for core, master, web, and reporting SQL Azure databases.	The minimum length is 8 characters.

`repAuthenticationApiKey`	XP and XDB only	Shared authentication key for XDB Reporting service.	The minimum length is 32 characters, for example, a GUID.

`analyticsMongoDbConnectionString`
`trackingLiveMongoDbConnectionString`
`trackingHistoryMongoDbConnectionString`
`trackingContactMongoDbConnectionString`
	XP0, XP and XDB	MongoDB connection strings for analytics and tracking databases.	-

`applicationInsightsLocation`	All	Optional location for Application Insights telemetry data storage. East US by default.	The Microsoft Azure Service Availability by Region lists the locations supporting Application Insights.
`sitecoreSKU`	XM, XP, XDB	Optional size of the Sitecore configuration.	XP1-XP5 for XP, XM1-XM5 for XM and XDB1-XDB5 for XDB.

Add modules

To add modules to your deployment:

1. Add the `modules` parameter to your `azuredeploy.parameters.json` file using the following snippet:

```
modules: {
value: {
items: [
/* add module entries here */
]
}
}
```

2. Configure the Bootloader module.
3. Configure WFFM for deployment on the Azure App Service.

Invoke the deployment command

To invoke the PowerShell command and initiate provisioning:

1. In PowerShell, go to the *Azure Toolkit* folder and load the Azure Toolkit module: *Import-Module .\tools\Sitecore.Cloud.Cmdlets.psm1 -Verbose*.
2. Add an Azure account to your PowerShell session: `Add-AzureRMAccount`.
3. If there is access to multiple subscriptions, select the subscription that you want to deploy into: `Set-AzureRMContext -SubscriptionName "<name of the subscription>"`.
4. Start provisioning using the `Start-SitecoreAzureDeployment` commandlet:

```
Start-SitecoreAzureDeployment [-location] <String> [-Name] <String> [-ArmTemplateUrl] <String> [-ArmParametersPath] <String> [-Li
```

The `Start-SitecoreAzureDeployment` commandlet accepts the following parameters:

Parameter	Description
`Location`	The name of the Azure datacenter where you want the resources deployed. Refer to the Azure datacenter compatibility table for the list of Microsoft Azure datacenters that the Sitecore Experience Platform supports deployment to.
`Name`	The name of the resource group for the new environment. It can refer to a new or an existing resource group. This is usually the same as the deployment ID.
`ArmTemplateUrl`	The URL of the ARM template file for the environment configuration that you want to deploy.
`ArmParametersPath`	The path to the populated `parameters.json` file for the template that you selected.
`LicenseXmlPath`	The path to the Sitecore license file that you want to deploy to the environment.
`SetKeyValue`	(Optional) Use this parameter in a script to deploy several environments by using a common set of default parameter values in the `azuredeploy.parameters.json` file and supplying environment-specific parameter values in the command line. The value of this parameter is a hashtable that contains a subset of parameters declared in the `azuredeploy.parameters.json` file. Any values that you specify in the command line in this parameter override the values specified in the `azuredeploy.parameters.json` file For example, to set or override the value in the `deploymentId` parameter, enter the following:

```
-SetKeyValue @{ "deploymentId" = "<new value>" }.
```

Send feedback about the documentation to docsite@sitecore.net.

Getting started with the Sitecore Azure Toolkit

The Sitecore Azure Toolkit contains the tools and resources that are necessary to prepare and deploy Sitecore solutions to the Microsoft Azure App Service®. Azure is a cloud-computing platform that provides a rich variety of services to help you create and run scalable applications without high upfront infrastructure investments.

The Sitecore Azure Toolkit simplifies the task of preparing and deploying a Sitecore solution with:

- PowerShell commandlets to package a Sitecore instance into role-specific packages
- Out-of-the-box integration with Microsoft Azure services: Microsoft Azure SQL®, Microsoft Azure Redis Cache, Microsoft Application Insights®, Microsoft Azure Search®
- Prebuilt ARM templates for frequently used topologies: XM, XP, XPSingle
- Configuration tweaks to run Sitecore on the Azure App Service
- Security features: HTTPS, low-privileged SQL access, secure Sitecore password

This topic describes:

Deploying Sitecore onto Azure with the Sitecore Azure Toolkit

The following is a high-level overview of the process that takes place when you use the Sitecore Azure Toolkit (SAT) to deploy Sitecore to Azure:

- First, you can download or create Web Deploy packages (WDPs) with the Sitecore Azure Toolkit. WDPs contain the application code and resources required to deploy Sitecore as well as any associated modules (if required). During the packaging process, the content of the WDPs are transformed to ensure correct operation when they are deployed to Azure. These transformations are defined via Sitecore Cargo Payload (SCCPL) packages which SAT references during the packaging process.
- After obtaining the packages, you prepare the Azure Resource Manager templates (ARM templates). These templates instruct the Azure Resource Manager to deploy resources to Azure, this includes both infrastructure (such as Web Apps) and application (WDPs). Sitecore provides default templates, to prepare them means choosing the right templates and providing the right parameters to those templates to configure the required deployment correctly.
- When you have prepared the WDPs and templates, you can deploy all of the required infrastructure and Sitecore applications onto Azure with a single command from SAT.

Prerequisites

To use the Sitecore Azure Toolkit, ensure you have the following:

- (If you are deploying a version of Sitecore that is earlier than 9.0) A Cloud-hosted MongoDB cluster for the *xDB Collection* and *Tracking* databases that are used with XP and XP0. You can either host MongoDB with a PaaS service such as mLab, or you can choose to self-host MongoDB using Azure virtual machines.

  Note

  From Sitecore version 9.0 and later, this is not a pre-requisite.

- .NET Framework 4.6
- PowerShell 4.0
- Microsoft Azure PowerShell® 2.0.1 or later. To check your installed version, open PowerShell and run the following command:

  ```
  Get-Module -ListAvailable -Name Azure -Refresh
  ```

  You can install Microsoft Azure PowerShell® using the Microsoft® Web Platform Installer or Windows PowerShell commands.

- Microsoft SQL Server Data-Tier Application Framework (DacFX) for SQL server 2012 or later.

  Note

  You usually install this framework with Microsoft SQL Server® or Microsoft Visual Studio®. You can also download DAC Fx from the Microsoft Download Center. DacFX is also installed with Microsoft Web Deploy 3.6 using the Web Platform Installer. In some software configurations Sitecore Azure Toolkit might not be able to load a proper version of DacFx automatically.

- An Azure subscription for deployments. You can sign up for a free trial account on the Microsoft Azure website.
- Cloud-hosted storage for Sitecore WebDeploy packages. An Azure Storage account can be used for this purpose.

Downloading the Sitecore Azure Toolkit

You can download the Sitecore Azure Toolkit from dev.sitecore.net, where you can also find the tools and resources necessary to package and deploy Sitecore solutions to the Azure App Service.

Note

When you download the Sitecore Azure Toolkit, the zip file is marked as *blocked* by Microsoft Windows therefore you must unblock the zip file. In Windows Explorer, right-click the file and click Properties. Then in the Properties dialog, on the General tab, click Unblock.

The following resources are distributed separately:

- Prebuilt WebDeploy packages for Sitecore roles in different environments on the Sitecore Experience Platform download page.
- Azure Resource Manager (ARM) provisioning templates and related parameter files on Github.

Installing the Sitecore Azure Toolkit

Sitecore Azure Toolkit is a ZIP package that you can unpack into a folder of the hard drive. When unpacked, Sitecore Azure Toolkit package creates the following files and folders:

- *Tools* folder – Microsoft PowerShell commandlets and the necessary supplementary libraries.
  - Sitecore.Cloud.Cmdlets.psm1 – the main module containing the necessary commandlets.
- *Resources/9.0.0* – Version-specific resources for Sitecore 9.0.
  - *CargoPayloads* – role-specific and feature-specific transformation packages. They are also known as Sitecore Cargo Payload packages (SCCPL).
  - *MsDeployXmls* – parameters files for WebDeploy packages.
  - *Configs* – configuration files for packaging tasks.
- Top-level files and a *Copyrights* folder – README and licensing information.

To load the main module into a PowerShell session, run the following command in the folder where the toolkit is unpacked:

```
Import-Module .\tools\Sitecore.Cloud.Cmdlets.psm1 -Verbose
```

This command loads commandlets into the PowerShell session. Commandlets provide an interface into the functionality of the Sitecore Azure Toolkit, for example:

- `Start-SitecoreAzurePackaging` – packages a Sitecore solution into WDP(s) so they are ready for deployment.
- `Start-SitecoreAzureModulePackaging` – packages a Sitecore module into WDP(s) so they are ready for deployment.
- `Start-SitecoreAzureDeployment` – initiates a Sitecore deployment onto Azure that includes all of the required resource infrastructure.

Azure Resource Manager templates

ARM templates are located on Github and provide a description of Sitecore environments hosted on Microsoft Azure App Service. The templates include definitions of the following resources:

- Hosting plans for Sitecore instances.
- Microsoft Azure SQL databases for content and reporting.
- Redis Cache service for session state.
- Microsoft Application Insights® for diagnostics.
- Microsoft Azure Search for content search and contact segmentation index.

The templates are compatible with the WebDeploy packages available on dev.sitecore.net for the corresponding Sitecore version or those that are produced by the Sitecore Azure Toolkit. The structure and organization of the templates are nested. The main template references an infrastructure template and an application template. The deployment responsibilities are then split respectively between templates to setup resources and templates to install the Sitecore application.

Web Deployment Packages

Prebuilt WDPs are located on the Sitecore Experience Platform download page. The WDP(s) come in a single .zip file and are grouped per version and per topology. After downloading the WDPs you must extract them from the .zip file before the Sitecore Azure Toolkit can use them.

Each role of the topology sizes intended for production use is contained within its own WDP. This means, for example that there will be a WDP for the *Content Delivery* role, a WDP for the *Content Management* role, and so on. The only exception to this is for single topologies that are designed for development and testing purposes. For these topologies you can combine multiple roles into a single WDP for deployment on a single shared resource.

Sitecore creates WDPs from standard Sitecore deployment packages that have been modified to run on Azure during the packaging process. Use these WDPs for a standard original deployment of Sitecore.

Note

With appropriate configuration changes, the Sitecore Azure Toolkit also enables users to package their own custom Sitecore deployments into WDPs so they can run on Azure. Further details on how this process works are available within the packaging topics.

Using the Sitecore Azure Toolkit

Use Sitecore Azure Toolkit to perform the following tasks:

- Package a Sitecore solution for the Microsoft Azure App service
- Deploy a new Sitecore environment to the Microsoft Azure App Service

Send feedback about the documentation to docsite@sitecore.net.

Package a Sitecore solution for the Microsoft Azure App Service

This document describes how to use the Sitecore Azure Toolkit to package a local Sitecore instance for deployment to the Microsoft Azure App Service®.

Prerequisites

Before you start the packaging process, make sure you have:

- A local Sitecore instance that you use as input to the packaging process. The Sitecore Azure Toolkit supports packaging Sitecore instances that are installed with the installation program or from a ZIP file and use the default folder structure.
- The Microsoft Data Tier Application Framework (DAC Fx). This framework is usually installed with Microsoft SQL Server® or Microsoft Visual Studio®. You can also download DAC Fx from the Microsoft Download Center.
- The Sitecore Azure Toolkit must be installed according to the instructions described in Getting Started with Sitecore Azure Toolkit.

Invoke the packaging command

To start the packaging process, import the Sitecore Azure Toolkit PowerShell module and invoke the `Start-SitecoreAzurePackaging` commandlet.

Note

The following commands must be carried out in Microsoft Windows PowerShell® in the folder where the Sitecore Azure Toolkit is installed.

```
> Import-Module .\tools\Sitecore.Cloud.Cmdlets.psm1
> Start-SitecoreAzurePackaging [-sitecorePath] <String> [-destinationFolderPath] <String> [-cargoPayloadFolderPath] <String> [-commonC
```

The `Start-SitecoreAzurePackaging` commandlet has the following parameters:

Parameter	Description
`sitecorePath`	The path to either a Sitecore instance folder of the Sitecore installation to be packaged or a packaged Sitecore instance folder in a zip file. An example of a path to a website folder, `C:\inetpub\wwwroot\simple161221`
`destinationFolderPath`	The folder where the generated packages are to be stored.
`cargoPayloadFolderPath`	The path to the folder containing the Sitecore version-specific role and feature transformation files. For example, for Sitecore 8.2 Update-1, this path would go to the subfolder: *resources\8.2.1\cargopayloads* in the Sitecore Azure Toolkit installation.
`commonConfigPath`	The path to the file that contains a list of the transformations applied to all of the roles. For example, for Sitecore 8.2 Update-1, this file is `common.packaging.config.json` and is located in the Sitecore Azure Toolkit installation, in the *resources\8.2.1\configs* subfolder.
`skuConfigPath`	The path to the file that contains the lists of role-specific transformations for the selected Sitecore version and the Sitecore configuration on the Microsoft App Service. For example, for Sitecore 8.2 Update-1, these files are located in the Sitecore Azure Toolkit installation, in the resources\8.2.1\configs subfolder and have the following names: • `xm1.packaging.config.json` – for an XM1 configuration. • `xp1.packaging.config.json` – for an XP1 configuration. • `xp0.packaging.config.json` – for an XP0 configuration.
`archiveAndParameterXmlPath`	The path to the WebDeploy Sitecore version-specific archive manifest and parameter declaration files. For example, for Sitecore 8.2 Update-1, these files are located in the Sitecore Azure Toolkit installation, in the *resources\8.2.1\msdeployxmls* subfolder.
`fileVersion`	An optional parameter that allows you to embed a version marker in the generated WebDeploy packages. The marker is stored in the `Version.txt` file inside the generated packages.

When performed, the Start-SitecoreAzurePackaging commandlet produces a number of WebDeploy packages in the destination folder. All packages follow the same name pattern, which includes the role name, for example, `Sitecore 8.2 rev. 161115_cm.wdp.zip`.

Note

Sometimes the commandlet also generates an intermediate WDP package with no role-specific suffix in its name. Ignore this package during deployment.

Packaging Example

The following code sample shows how to use the Start-SitecoreAzurePackaging commandlet to prepare WDP packages for XP1 configuration. The sample assumes the following:

- Sitecore is installed locally in `C:\inetpub\example`, and the website folder is `C:\inetpub\example\Website`.
- Sitecore Azure Toolkit is installed in `C:\Tools\SitecoreAzureToolkit.`
- Generated packages are stored in `C:\Workspace\WDPs.`

```
$SAT="C:\Tools\SitecoreAzureToolkit"

# Import commandlets

Import-Module "$SAT\tools\Sitecore.Cloud.Cmdlets.psm1"

# Set the parameter variables

$SKU="xp1"
```

```
$Version="8.2.1"
$Resources="$SAT\resources\$Version"
$Website="C:\inetpub\example\Website"
$Output="C:\Workspace\WDPs"
# Create the output folder
mkdir $Output
# Start the packaging process
Start-SitecoreAzurePackaging -sitecorePath "$Website" -destinationFolderPath $Output -cargoPayloadFolderPath "$Resources\cargopay
```


The structure of an SCCPL transformation

A Sitecore Cargo Payload (SCCPL) package is the extension of a ZIP package with the following structure:

Command	Description
`CopyToWebsite/*`	Copies files to the *Website* folder in the target installation. You can use this to deploy new files or overwrite existing files, such as resource files, DLLs, and configuration files.
`CopyToRoot/*`	Copies files to the root of the Web Deployment Package (WDP). You usually use this, for example, to inject `.dacpac` or `.sql` files to perform database changes.
`Xdts/*`	XDT transformations for XML files are under the Website root. Use XDT transformations to tweak configuration files that the WDP deploys without adding new configuration files. The file name convention is: `{original file name}.xdt`. Note SCCPL transformations use the XDT syntax to transform XML files, not the Sitecore configuration patch syntax.
`IOActions/*.ioxml`	An XML file that describes the actions that are going to happen to the files in a WDP package. See the example in the following IOActions section.

This topic describes:

- The structure of a SCCPL package
- Basic SCCPL packages
- IOActions

The structure of a SCCPL package

The following example shows the structure of a SCCPL package.

Package	Description
`CopyToWebsite\App_Config\Include\Component.config`	Configuration file for a new component, integration into Sitecore.
`CopyToWebsite\bin\Component.dll`	DLL file for the component.
`CopyToWebsite/sitecore/shell/client/Applications/Component/page.cshtml`	Resource file - MVC page view.
`Xdts\App_Config\ConnectionStrings.config.xdt`	XDT transformation: adding connection string for the new component.
`IOActions\Component.ioxml`	Enable/disable configuration files.

Basic SCCPL packages

You can fine-tune your Sitecore solution to match your needs by using SCCPL packages. There are two different approaches to doing this:

- Adjust your current SCCPL.

Note

If you choose this option, you must branch and merge on updates.

- Create a new SCCPL based on your current SCCPL in the *Cargo* folder, (the *Cargo* folder includes SCCPLs). Then, include the relevant SKU file, (in `$SKU.packaging.config.json`, you can learn how to process a SCCPL).

The following table outlines the basic Sitecore SCCPL packages:

Package	Description
`Sitecore.Cloud.Search.Azure.sccpl`	Transforms the `web.config` file using the Azure Search provider. It is also a patch to add the `ConnectionStrings.config` file to the `cloud.search` connection string.
`Sitecore.Cloud.ApplicationInsights.sccpl`	Copies the Application Insights configuration files to both the */base* and the *App_Config/Sitecore/Azure/* folders, and the assemblies files to the */bin* folder. The `ConnectionStrings.config` file also transforms to add the `appinsights.instrumentationkey` connection and update the `web.config` file with the Application Insights configuration.
`Sitecore.Cloud.DisableAnalytics.sccpl`	Applies the current SCCPL for the XM WDPs. This disables the `Analytics` and `XConnect` connection strings in the `ConnectionStrings.config` file and turns off the `Xdb.Enabled` setting in the `Sitecore.Xdb.config` file.
`Sitecore.Cloud.DisableExm.sccpl`	Applies the current SCCPL for the XM WDPs. This disables the `EXM` connection strings in the `ConnectionStrings.config` file and turns off the `EXM.Enabled` setting in the `Sitecore.EmailExperience.Core.config` file.
`Sitecore.Cloud.Common.sccpl`	A common SCCPL for all WDPs. The main actions of this package include: • Setting the data folder to *D:\home\site\wwwroot\App_Data* • Preparing a placeholder for a Sitecore license • Disabling the `Sitecore.Diagnostics.config` file • Changing the value of the `customErrors mode` setting to `"Off"` (`customErrors mode="Off"`)
`Sitecore.Cloud.HttpsRedirection.sccpl`	A common SCCPL for all WDPs. This package transforms the `web.config` file to apply https redirection.
`Sitecore.Cloud.IPSecurity.sccpl`	Updates the `web.config` file with an example of IP filtering to limit access to the CM, Processing, and DDS roles.
`Sitecore.Cloud.Redis_CD.sccpl`	Applies the SCCPL for the CD WDP to the XM and XP topologies, and enables the Redis cache configuration and connection strings.
`Sitecore.Cloud.RoleSpecific_CD.sccpl`	Applies the SCCPL for the CD WDP to the XM and XP topologies. The main actions of this package include: • Setting the WDP role to `ContentDelivery` in the `web.config` file • Setting the correct connection strings list for the CD role in the `connectionstrings.config` file
`Sitecore.Cloud.RoleSpecific_CM.sccpl`	Applies the SCCPL for the CM WDP to the XM and XP topologies. The main actions of this package include: • Setting the WDP role to `ContentManagement` in the `web.config` file • Setting the correct connection strings list for the CM role in the `connectionstrings.config` file
`Sitecore.Cloud.RoleSpecific_DDS.sccpl`	Applies the SCCPL for the DDS role to the XP topology. The main action of this package is: • Setting the WDP role to `ContentManagement`, `DedicatedDispatch` in the `web.config` file.
`Sitecore.Cloud.RoleSpecific_PRC.sccpl`	Sets the correct connection strings list for the DDS role and applies the PRC WDP for the XP and xDB topologies. The main actions of this package include:

	• Setting the WDP role to `Processing` in the `web.config` file. • Setting the correct connection strings list for the PRC role in the `connectionstrings.config` file.
`Sitecore.Cloud.RoleSpecific_REP.sccpl`	Applies the SCCPL for the CM WDP to the XP and xDB topologies. The main actions of this package include: • Setting the WDP role to `Reporting` in the `web.config` file. • Setting the correct connection strings list for the REP role in the `connectionstrings.config` file.
`Sitecore.Cloud.RoleSpecific_XDBSingle.sccpl`	Applies the SCCPL for the Sitecore Single WDP to the xDB Single topology. The main actions of this package include: • Setting the WDP role to `Processing`, `Reporting` in the `web.config` file. • Setting the correct connection strings list for the Single (PRC+REP) role in the `connectionstrings.config` file.
`Sitecore.Cloud.RoleSpecific_XMSingle.sccpl`	Applies the SCCPL for the Sitecore Single WDP to the XM Single topology. The main actions of this package include: • Setting the WDP role to `ContentManagement`, `ContentDelivery` in the `web.config` file. • Setting the correct connection strings list for the Single (CM+CD) role in the `connectionstrings.config` file.
`Sitecore.Cloud.RoleSpecific_XPSingle.sccpl`	Applies the SCCPL for Sitecore Single WDP to the XP Single topology. The main action of this package is: • Setting the correct connection strings list for the Single (CM+CD+PRC+REP) role in the `connectionstrings.config` file.
`Sitecore.Cloud.Security.sccpl`	Copies database accounts creation scripts to the website root and transforms the `web.config` file with security rules.
`Sitecore.Cloud.Security_CD.sccpl`	Transforms the `web.config` file with security rules for the CD role.
`Sitecore.Cloud.SetCompatibilityLevel.sccpl`	Copies the SQL server compatibility level scripts to the website root.
`Sitecore.Cloud.Thundercracker.sccpl`	The main actions of this package include: • Removing the *UploadWatcher* folder functionality, (because it is not applicable for Azure). • Changing the location of the *Media.CacheFolder* folder. • Changing the `PageStateStore` and `ViewStateStore` settings to write to the DB.

IOActions

The IOActions are described in `.ioxml` files with `path` and `action` attributes; the `enable`, `disable`, and `delete` action values; and the following structure:

```
<IOActions>
<IOAction path="App_Config\Include\001.Sitecore.Speak.Important.config" action="disable" />
…
</IOActions>
```

Attributes	Description
`path`	The path to the file that you want to change, relative to the *Website* folder. Note The value must be the exact path to a single file, including the expected extension(s). Wildcards are not supported.
`action`	A file-level action (enable, disable, or delete) that is performed on the file.

IO action values	Description

`enable`	Enable a configuration file by renaming it from `*.config.disabled`, or `*.config.example`, to `*.config`.
`disable`	Disable a configuration file by renaming it from `*.config` to `*.config.disabled`.
`delete`	Delete the file from the package.

Send feedback about the documentation to docsite@sitecore.net.

The Web Deploy Packages for a module

The following sections describe references and commands you can use to create Web Deploy packages (WDPs) that you can deploy either on-premise or on Azure, using ARM templates:

- Generating an initial WDP
- Applying transformations
- Embedding dynamic transformations
- Embedding files into the package, (such as .dacpac or .sql)
- Adjusting parameters
- Invoking multiple commands in a row
- Testing your package

Before you start, you must install Sitecore Azure Toolkit and load the commandlets into a PowerShell session. The required commandlets are defined in the `tools\Sitecore.Cloud.Cmdlets.dll` file.

Generating an initial WDP

You can use a Sitecore module ZIP package or a Team Development for Sitecore .update package and convert it into an equivalent `.scwdp.zip` package that is compatible with MSDeploy using the following PowerShell call:

```
ConvertTo-SCModuleWebDeployPackage [-Path] <string> [[-Destination] <string>]
```

Parameters	Description
`Path`	The path to the original module ZIP package (or an `.update` package created by Team Development for Sitecore).
`Destination`	The path to the file or folder that you want to save the generated package into.

This creates a `.scwdp.zip` package with the following content:

Content	Description
`Content/Website/*`	The files that you want to install on the website.
`Content/Website/App_Data/Poststeps/*.poststep`	Post step of the original package, if it was defined.
`core.sql`	The SQL script that installs items into the core database. This file is only present when the package installs items in the core database.
`master.sql`	The SQL script that installs items into the master database. This file is only present when the package installs items in the master database.

The `.scwdp.zip` package accepts the following parameters:

Parameter	Description
`Application Path`	The path to the website where you will install the application (use an IIS application name or a physical folder path).
`Core Admin Connection String`	The connection string that connects to the core database during installation. This parameter is only added when the original package installs items onto the core database.
`Master Admin Connection String`	The connection string that connects to the master database during installation. This parameter is only added when the original package installs items onto the master database.

Advanced scenarios

Refer to the following sections (Applying transformations, Embedding dynamic transformations, Embedding files into the package, Adjusting parameters, Invoking multiple commands in a row, Testing your package) if you want to perform more advanced scenarios, such as:

- Changing the default module configuration when you install it on Azure, for example, if you want to enable App Insights or Azure Search integration.
- Generating multiple WDPs for different roles or configurations from a single Sitecore ZIP as a part of your build pipeline.
- Installing a new database together with the module.
- Updating a database (including adding new tables or users and roles), when installing a module.
- Adding parameters to make a package installation more configurable.

Applying transformations

You can apply transformations to configure your module in the following scenarios:

- If you want a non-standard installation of a module on Azure. For example, to enable Azure Search integration or to tweak performance parameters.
- You can also use transformations to create role-specific packages from a single source package.

Note

You can only apply transformations to the content of a module package, that is, the transformations can only affect the files that the module delivers.

Use the following as a reference:

```
Update-SCWebDeployPackage [-CargoPayloadPath] <string[]> [-Path <string>] [-WhatIf]:
```

Parameters	Description
`CargoPayloadPath`	An array of paths to the SCCPL files that apply to the WDP.
`Path`	The path to the WDP that you want to modify.
`WhatIf`	A switch that enables a dry run of the transformation and produces the actions that would be performed, without modifying the package.

Embedding dynamic transformations

Dynamic transformations allow a module to transform the target Sitecore installation at the time the module is installed. For example, you can add an HTTP handler to the `Web.config` file.

Note

Dynamic transformations are applied to the entire Sitecore installation when the module package is deployed and can affect any preexisting files. Improper use of dynamic transformations can make your Sitecore instance unusable.

Use the following reference:

```
Update-SCWebDeployPackage [-EmbedCargoPayloadPath] <string[]> [-Path <string>] [-WhatIf]
```

Parameters	Description
`EmbedCargoPayloadPath`	An array of paths to the SCCPL files that embed into the WDP. .Transformations are stored in the *Content/Website/App_Data/Transformations* folder and applied by Bootloader during the installation process.
`Path`	The path to the WDP that you want to modify.
`WhatIf`	A switch that enables a dry run of the transformation and produces the actions that would be performed, without having to modify the package.

Embedding files into the package

SCCPL transformations are convenient to add prebuilt sets of files and configuration changes into WDPs. However, you can also embed individual files, for example, dacpac files, for new databases into a WDP.

Use the following reference:

```
Update-SCWebDeployPackage [-SourcePath] <string> [-Path <string>]
```

Parameters	Description
`SourcePath`	The path to the file that you want to add.
`Path`	The path to the WDP that you want to modify.

Adjusting parameters

If you want to customize your deployment process, or if you need to deploy dacpac or SQL files with the module, you must create and embed a custom `parameters.xml` file into your deployment package. You can also define parameters by using the reference for IIS Web Application packages.

When you apply a `parameters.xml` file to your WDP, SAT automatically regenerates the package manifest to include the files you reference in your parameters. This means that your `parameters.xml` file becomes a kind of manifest, describing what is inside the package.

The following command adds parameters and regenerates the manifest:

```
Update-SCWebDeployPackage [-ParametersXmlPath] <string> [-Path <string>]
```

Parameters	Description
`ParametersXmlPath`	The path to the `parameters.xml` file that you want to embed.
`Path`	The path to the WDP that you want to update.

Invoking multiple commands in a row

It is also possible to pipe commands in a single command line:

```
$WDP = ConvertTo-SCModuleWebDeployPackage $ZIP or $Update | `
Update-SCWebDeployPackage -CargoPayloadPath @($SCCPL) | `
Update-SCWebDeployPackage -SourcePath $DACPAC | `
Update-SCWebDeployPackage -ParametersXmlPath $Parameters
```

When using .Net, replace `ConvertTo-SCModuleWebDeloyPackage` with `dotnet`, publish, and then manipulate the WDP as usual.

Testing your package

With Microsoft Web Deploy you can test your package installation on a local system with the following command line:

```
msdeploy -verb:sync -source:package=%CD%\test.scwdp.zip -dest:archiveDir=%CD%\output -setParam:X=TESTVALUE
```