.NET Core | Azure | DevOps | Xbox | HTML5 | SharePoint/Office 365 | Angular 2

S&S

windows .developer

windows .developer

2.2017

Deutschland 9,80 €
Österreich 10,80 €
Schweiz 19,50 sFr

www.windowsdeveloper.de

Einblicke mit Azure

Der App-Checker

Mit Azure Application Insights die eigene App im Blick behalten

Mit Azure zur IoT-Infrastruktur

Vom Gerät zum Live-Dashboard

©iStockphoto.com/annuker ©S&S Media

Das .NET-Kern-Update
.NET Core 1.1, ASP.NET Core 1.1 und Entity Framework Core 1.1
▶ 22

Aus Alt mach Neu
Migration gewachsener Anwendungen mit CefSharp
▶ 60

Der Produktivitätsturbo
Mehr Produktivität mit SharePoint 2016 in Unternehmen
▶ 90

Automatisierung per SharePoint/Office 365 Developer PnP

Das PnP-Framework mit PowerShell

Seit Längerem will Microsoft in SharePoint fehleranfälligen und unsicheren serverseitigen Code durch Apps und anderen clientseitigen Code ersetzen. Hierfür stehen Webdienste bereit, wodurch der „alte“ Ansatz mit WSP-Dateien nicht länger möglich ist. Gerade aber das Provisionieren von Vorlagen für Site Collections, Sites, Listen und Webseiten wurde bisher über solche WSP-Dateien gelöst. Deshalb ist man nun insbesondere beim zusätzlichen Provisionieren von Artefakten in einer hybriden SharePoint-Umgebung einigen Herausforderungen ausgesetzt.

von Sebastian Schütze und Stefan Mumalo

CSOM (Client-side Object Model) bietet einen Ansatz für diese Art der Provisionierung, die jedoch eher funktionsorientiert ist. Die Office-Developer-PnP-Community hat mit dem PNP-Framework einen aufgabenorientierten Wrapper um das CSOM erstellt. Die PnP-PowerShell bietet hierbei eine interessante Möglichkeit der Automatisierung. Dieser Artikel stellt diese Möglichkeiten exemplarisch vor.

Ein neuer Ansatz: das PnP-Provisioning-Framework

Mit der Einführung von SharePoint 2013 (und nicht zuletzt auch SharePoint Online) wurde ein neues Paradigma vorgeschlagen. Das sollte dazu führen, dass SharePoint sicherer und performanter mit eigens geschriebenen Applikationen umgehen kann. In der Vergangenheit wurde SSOM (Server-side Object Model) ausgiebig in SharePoint-Lösungen verwendet, um Anpassungen vorzunehmen. Das führte zu großen Problemen, was sich meist in der Wartbarkeit und Sicherheit einer SharePoint-Farm zeigte. Mit den neuen SharePoint-Apps gehört das der Vergangenheit an, denn damit ist kein serverseitiger Code möglich. Eines konnte (und kann) jedoch immer weiterverwendet werden: Das Verteilen von SharePoint-Artefakten, wie Listen oder Webtemplates mit XML- und Caml-Code. Das führte bei vielen Kunden jedoch zu Problemen beim Lifecycle-Management des Deployments und auf Farm- bzw. Tenant-Level. Also empfiehlt Microsoft die Verschiebung von XML-gebundener Verteilung der SharePoint-Artefakte auf codegebundene Verteilung. Was heißt das? Inhalte wie Templates, Listen, Inhaltstypen usw. sollten mithilfe von eigenem Code erstellt werden. Das ist allerdings – gelinde gesagt – ein ganz schöner Aufwand! Warum? Nun, weil CSOM (Client-side Object Model) und JSOM (JavaScript Object Model) für das Remote Provisioning verwendet werden müssen und beide sehr funktionsorientiert aufgebaut sind. Das heißt, dass man zum Anlegen einer Liste bis zur Erstellung und zum Hinzufügen eines Inhaltstyps zur Liste leicht auf 50-100 Zeilen Code kommt. Das stellt natürlich einen enormen Aufwand dar.

Ein Ansatz, der genau das erleichtern soll, ist das PnP-Provisioning-Framework. Es hat den Vorteil, dass es aufgabenorientiert arbeitet. Wenn eine Liste erstellt

```
<pnp:Provisioning xmlns:pnp="http://schemas.dev.office.com/PnP/2015/12/ProvisioningSchema">
  <pnp:Preferences Generator="OfficeDevPnP.Co">...</pnp:Preferences>
  <pnp:Templates ID="MeineContainerID">
    <pnp:ProvisioningTemplate ID="MeineTemplateID" Version="1">
      <pnp:WebSettings SiteLogo="{site}/SiteAssets/logo.png" />
      <pnp:Lists>...</pnp:Lists>
      <pnp:Files>...</pnp:Files>
      <pnp:Pages>...</pnp:Pages>
    </pnp:ProvisioningTemplate>
  </pnp:Templates>
</pnp:Provisioning>
```

Abb. 1: Beispiel einer Provisioning-XML-Datei (Quelle: Axians IT Solutions GmbH)

wird, dann ist das mit wenigen Zeilen Code schnell erledigt. Das Framework, das es für C# und JavaScript gibt, ist ein Wrapper um CSOM bzw. JSOM und konzentriert sich auf die effektive Erstellung von Inhalten mit nur wenigen Zeilen Code. Die dazugehörige PnP-Provisioning-Engine bietet sogar die Möglichkeit, bereits gebaute Umgebungen in SharePoint in Form einer XML-Notation zu exportieren. Diese Inhalte können dann leicht auf andere Seiten übertragen werden. Das sind aber nur die offensichtlichsten Vorteile.

Vorteile, die auf der Hand liegen

Die folgende Aufzählung erhebt keinen Anspruch auf Vollständigkeit und soll lediglich die Hauptvorteile von PnP darstellen.

- **Systemunabhängigkeit:** Das Provisioning-Framework funktioniert für die SharePoint-Versionen 2013, 2016 und SharePoint Online. Es ist egal, wo der Deployment-Prozess durchgeführt wird: Die Provisioning Engine erlaubt ein lückenloses Verwenden des geschriebenen Codes (egal ob C#, PowerShell oder im XML Schema).
- **Erweiterbarkeit:** Die Engine kann leicht selbst durch eigene Funktionalität erweitert werden. Das Framework bietet Schnittstellen, um sich an den bestehenden Code mit REST oder CSOM anzudocken. Somit muss nicht unbedingt gewartet werden, bis eine neue Version erscheint.
- **Konfliktfreie Deltaprovisionierung:** Das bedeutet, dass z. B. ältere bestehende Sites durch vollständige XML-Templates ersetzt werden können. Es werden in der Regel auch keine Meldungen ausgegeben, dass ein Element bereits existiert. Alte Daten und Strukturen werden dabei erweitert, existierende Definitionen nicht gelöscht.
- **Wiederverwendbarkeit:** Erstellte Inhalte können mithilfe eines Exports als XML-Vorlage persistiert und somit wiederverwendet werden. Das dient einerseits als Site-Template zum Erstellen einer neuen Site mit einer definierten Vorlage, andererseits kann es auch für Multi-Staging-Systeme verwendet werden. Entwickelte Inhalte können dank des serialisierten XML vom Entwicklungsserver auf das Test- oder Produktivsystem umgezogen werden. Das XML kann sogar mit Parametern versehen werden, die dynamisch während der Provisionierung ersetzt werden können (Namen, GUIDs etc.). Somit ist es hochflexibel und in parametrisierter Form sogar sehr stark wiederverwendbar.

PowerShell und das PnP-Framework

Unser Hauptaugenmerk soll aber auf dem PowerShell-Ansatz liegen. Im Kern sind die PowerShell-Module ebenfalls Wrapper, die das Provisioning-Framework nutzen. Diese Module sind in C# geschrieben und bilden mit PowerShell ein mächtiges Werkzeug, um besonders hybride SharePoint-Umgebungen zu erweitern bzw. zu automatisieren.

PowerShell hat den großen Vorteil, dass sich die Commandlets sehr gut mit C#-Objekten verstehen. Man kann ohne viel Overhead (Visual Studio IDE, Debugger etc.) eine Automatisierung aufsetzen, die nicht den Anspruch hat, dass der gesamte Prozess sofort klar ist. Die Wartung ist ebenfalls leicht in der Handhabung, da es keine Neukompilierung benötigt. Es ist wichtig zu wissen, dass die Cmdlets für SharePoint Online von Microsoft nicht durch die PnP-PowerShell-Cmdlets ersetzt werden, sie erweitern diese vielmehr. Während mit den Cmdlets von Microsoft eher administrative Aufgaben durchgeführt werden, konzentrieren sich die Cmdlets von PnP auf die Verteilung von Artefakten und Inhalten in SharePoint.

Obwohl die Onlinehilfe der PnP-PowerShell-Seite selbst eine Anleitung [1] bereitstellt, um alles Nötige aufzusetzen, hatten wir die Anforderung, in Kundenprojekten Skripte einzusetzen, bei denen erst zur Laufzeit geklärt wurde, ob sie auf SharePoint OnPremise oder in der Cloud laufen sollten. Zum Glück liefert die Community dazu selbst schnell Anwendungsbeispiele, wie dies vollzogen werden kann [2]; eine Möglichkeit zeigt Listing 1.

Im Prinzip dürfen nach der Installation der Frameworkversionen für On-Premise und Online diese nicht automatisch in die Umgebung geladen werden. Das wird wie oben beschrieben erreicht, indem entsprechende Verweise in der *Path*-Variable der Umgebungsvariablen entfernt werden. Stattdessen baut man sich ein eigenes Unterscheidungsmerkmal in das Skript ein (z. B. eine Boolesche Variable) und lädt entsprechend eine der bei-

```
<pnp:ProvisioningTemplate ID="MeineTemplateID" Version="1">
  <pnp:WebSettings SiteLogo="{site}/SiteAssets/logo.png" />
  <pnp:Lists>...</pnp:Lists>
  <pnp:Files>
    <pnp:File Src="MeinBild.png" Folder="{site}/SiteAssets" />
  </pnp:Files>
  <pnp:Pages>...</pnp:Pages>
</pnp:ProvisioningTemplate>
```

Abb. 2: Dateien für ein PnP-Paket werden im Provisioning-Template mit angegeben (Quelle: Axians IT Solutions GmbH)

den DLLs in die Umgebung. Wichtig dabei ist, dass nur eine der beiden DLLs aktiv ist, denn sonst gerät man in einen Konflikt. Nachdem dies getan ist, kann ein Skript für verschiedene Umgebungen genutzt werden.

PnP-Provisioning-Schema und PnP-Pakete

Das Verteilen von Artefakten darf nicht allein durch eine Aneinanderreihung von Skriptbefehlen geschehen, sondern über XML. Das dazu definierte Provisioning-Schema beschreibt, wie das auszusehen hat: Dem XML liegt ein Schema zugrunde [3], das die Verwendung sehr umfangreich definiert. Dabei wird jedem Schema eine Version zugeordnet und es wird zusammen mit der Engine in seiner Funktionalität schnell erweitert. Das Schema erlaubt es, in einer Datei mehrere Templates mit eigenen IDs zu definieren und sie einer eigenen Versionierung unterzuordnen. Die Templates wiederum werden thematisch je nach Artefakten (Inhalten) gruppiert (Listen, Inhaltstypen, Security usw.). Teilweise erinnert das XML an die klassische Schemanotation in SharePoint. In **Abbildung 1** findet sich ein Beispiel für ein solches Provisioning-Template.

Aber wie kommt man zu diesem XML? Es muss und sollte nicht alles selbst zusammengebaut werden. Der beste Weg ist es, sich das Ganze in der SharePoint-Oberfläche zusammenzubauen und dann das XML zu exportieren. Danach kann es angepasst werden. Wenn Sie sich, wie oben beschrieben, mit der korrekten Seite verbunden haben, können Sie sich das Template wie folgt holen:

```
Get-PnPProvisioningTemplate -Out 'D:\MyPnPTemplate.xml'
```

Das reicht schon aus, um die Maschinerie zu starten. Umgekehrt lässt sich das auch von einem Template auf eine Site übertragen. Aber Vorsicht, bitte probieren Sie das erst auf einer Testsite aus, damit nichts ungewollt überschrieben wird.

```
Apply-PnPProvisioningTemplate -Path 'D:\MyPnPTemplate.xml'
```

Beide Cmdlets lassen sich noch nach so genannten Handlern einschränken. Sind diese nicht angegeben, werden alle unterstützten Artefakte exportiert. Geben Sie als Parameter *Handler*, also z. B. Content Types an, dann werden nur diese geholt bzw. gesetzt – egal, welche Inhalte in der Site oder im XML-Template enthalten sind. Die Templates haben nicht nur die Möglichkeit, einzelne Artefakte zu verteilen, sondern auch Designdateien wie Displaytemplates, JavaScript- und CSS-Dateien. Damit diese jedoch für Deployment-Prozesse nicht lose in irgendwelchen Ordnern liegen, können sie in *.pnp*-Dateien gepackt werden, was folgender Befehl erledigt:

```
Convert-PnPFolderToProvisioningTemplate -Folder 'D:\MyTmp' -Out 'D:\
                                                    MyPnPTemplate.pnp'
```

Hierbei muss nur aufgepasst werden, dass der Name der XML-Datei dem Namen der PnP-Datei entspricht. Die XML-Datei und alle anderen mit zu verpackenden Dateien müssen im angegebenen Pfad stehen. Natürlich müssen alle Dateien auch im XML-Template referenziert werden. **Abbildung 2** zeigt, wie das aussehen kann.

Natürlich haben Sie auch immer die Möglichkeit, sämtliche Cmdlets der PnP-PowerShell zu nutzen, statt dies durch XML durchzuführen. Das bietet Ihnen einen höheren Grad an Dynamik, besonders wenn Sie die Templatedateien selbst inhaltlich aufsplitten und nach

Listing 1

```
#eventuelles SP Online Modul entfernen
Remove-Module -Name Microsoft.Online.SharePoint.PowerShell-
                                        ErrorAction SilentlyContinue

$modulePath = '#OnPremisePnpPfad#\SharePointPnP.PowerShell.2016.
                                                    Commands.dll'
$modulePath = '#SPO365PnPfad#\SharePointPnP.PowerShell.Online.
                                                    Commands.dll'
$credentials = New-Object -TypeName System.Management.
                    Automation.PSCredential -argumentlist $UserName,
                                                    $PasswordSecureString
$siteUrl = 'https://Url.zum.SharePoint'
Import-Module $modulePath -DisableNameChecking
Connect-PnPOnline -Url $siteUrl -Credentials $credentials
```

Wie jedes Open-Source-Projekt lebt auch das PnP-Framework von einer Community und ist darauf angewiesen, dass wir als „Benutzer“ alles, was uns auffällt, auch mitteilen.

eigenen Merkmalen laden. Sie können also entweder die Skripte allein durch parametrisierte Provisioning-Templates laufen lassen oder Cmdlets mit einbringen.

Nicht alles, was glänzt, ist Gold

Der Entwickler darf vor allem nicht vergessen, dass das PnP-Framework ein Open-Source-Projekt ist, das kein gigantisches Entwicklungsteam im Rücken hat. Also wird es noch etwas dauern, bis das Framework uns wirklich alle Projektaufgaben erleichtern kann – aber vieles ist jetzt schon einfacher und schneller. Allerdings ist beispielsweise ist der Umgang mit Publishing Pages noch nicht enthalten (Stand: November 2016).Wenn Provisioning-Templates aus einer nicht-englischsprachigen Seite erstellt werden, ist die generierte Vorlage nicht ohne Anpassungen benutzbar. Anders als im Englischen werden die Systemlisten und Spalten mit exportiert, was beim Anwenden des Templates zu Fehlern führt. Es müssen also per Hand und mit viel Know-how die Dinge entfernt werden, die nicht gebraucht werden. Besonders bei PnP-PowerShell braucht es teilweise Geduld, bis neu hinzugefügte Features im PnP-Sites-Core auch als Cmdlet nutzbar sind.

Wer hier eine Toolsammlung erwartet, die einem die Arbeit größtenteils abnimmt, erwartet leider (noch) zu viel. Das wird sich aber vermutlich in den nächsten Monaten ändern. Wie jedes Open-Source-Projekt lebt auch das PnP-Framework von einer Community und ist darauf angewiesen, dass wir als „Benutzer“ alles, was uns an Fehlern, Erweiterungsmöglichkeiten und Unnützem auffällt, auch mitteilen. Üblicherweise wird sich des Problems sehr schnell angenommen und man erhält Feedback. Nicht umsonst hat sich diese Community den Slogan „Sharing is Caring“ auf die Fahne geschrieben.

Häufig fallen Fehler auf, die in der neuesten Version schon behoben sind. Es lohnt sich also immer, auf die neueste Version zu aktualisieren. Doch Vorsicht: Da Interferenzen nicht ausgeschlossen werden können, sollte vor einem Produktiveinsatz immer getestet werden. Die Authentifizierung an ADFS-Umgebungen funktioniert momentan nur eingeschränkt: Hier muss, sofern die Skripte automatisiert laufen sollen, auf eine erweiterte Webapplikation mit NTLM ausgewichen werden.

Rosige Aussichten

Mit der Veröffentlichung und Weiterentwicklung dieses Projekts ist allen geholfen, die Inhalte unkompliziert und auf schnellem, aber sicherem Weg auf SharePoint verteilen möchten. PnP ist Open Source, ganz im Trend der Zeit. Jeder kann sich beteiligen, es wird von Microsoft tatkräftig unterstützt, und nicht zuletzt durch die Community und die beteiligten MVPs entwickelt sich dieses Framework sehr schnell weiter. Auch wenn wir hier noch von einem frühen Entwicklungsstadium reden, bietet die aktuelle Version schon ein extrem großes Spektrum an Möglichkeiten. Und zusammen mit der kürzlich veröffentlichten JavaScript-Bibliothek der PnP Provisioning Engine wird die Entwicklung von Apps auf Basis des remote Provisioning-Patterns noch wesentlich leichter.

Sebastian Schütze ist Diplominformatiker, seit dreieinhalb Jahren SharePoint-Berater der Axians IT Solutions GmbH und seit mehr als sieben Jahren in Projekten tätig, u. a. als Webentwickler.

sebastian.schuetze@razorspoint.com

Stefan Mumalo ist SharePoint-Entwickler bei der Axians IT Solutions und beschäftigt sich seit zehn Jahren mit der SharePoint- und Webentwicklung.

stefan.mumalo@axians.de

Links & Literatur

[1] https://github.com/OfficeDev/PnP-PowerShell

[2] https://www.spcaf.com/blog/using-the-officedev-pnp-powershell-cmdlets-for-both-on-prem-and-in-the-cloud/

[3] https://github.com/OfficeDev/PnP-Provisioning-Schema