Architektur | Agile Methoden | Angular | Azure | IoT | SharePoint

S&S

windows .developer

windows .developer

4.2018

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

www.windowsdeveloper.de

Eine Frage der Architektur

Dependency Injection in Aktion
Clean Code durch Design Patterns

Design to Cost
SaaS und Cloud kosteneffizient planen

IoT to the Rescue
Microsofts Internet-of-Things-Ökosystem

4 195809 809803 04

Ordnung im Unternehmen
Struktur für große Angular-Anwendungen
▶ 58

Das SharePoint-ALM-API
SharePoint-Online-Lösungen provisionieren
▶ 86

Erfahrungsbasierte Aufwandsschätzung
Kalkulation mit der Abakus-Methode
▶ 44

SharePoint-Online-Lösungen mit dem ALM-API provisionieren

Das SharePoint-ALM-API

Im November 2017 wurde von Microsoft für SharePoint (SP) Online das ALM-API veröffentlicht. Dieses API schließt eine große Lücke innerhalb der automatischen Provisionierung von Lösungen in SP Online. Zusammen mit dem Site Collection App Catalog existieren nun weitreichende Möglichkeiten, Lösungen zu verteilen, ohne auf manuelle Schritte oder Workarounds ausweichen zu müssen.

von Sebastian Schütze

Halt! Bevor wir richtig in das Thema hineinspringen, müssen wir sicherstellen, dass wir alle ALM nicht mit grünen Wiesen und lila Kühen verwechseln und uns überlegen, was das mit SharePoint zu tun hat. ALM bedeutet nichts anderes als Application Lifecycle Management und beschreibt den Prozess von der Entwicklung der ersten Zeile Code bis hin zur Installation der fertigen Software auf dem Produktivsystem.

Dieser Prozess soll sicherstellen, dass nichts vergessen wird, was zur Qualität des fertigen Produkts beiträgt. Dazu gehören natürlich Planung, Tests, Deployment und Maintenance. Da die Maxime Qualität sein soll, reicht es nun auch nicht aus, dass Entwickler ihr Produkt entwickeln und Operations das ganze Schritt für Schritt auf dem System installiert. Am Ende sollte es nämlich so aussehen, dass Operations sich nur noch um Probleme der Plattform kümmern muss, auf der die Software installiert wird, nicht aber um die Installation selbst. Und das funktioniert nur mit Automatisierung.

ALM und SharePoint: ein mühsamer Aufstieg

Wenn es um automatische Provisionierung ging, waren ALM und SP noch nie die besten Freunde. Jedenfalls nicht mehr seit der Version 2013. Mit ihr wurden die Add-ins eingeführt, die damals allerdings noch Apps hießen. Die klassischen serverseitigen Lösungen ließen sich zuvor zu 100 Prozent automatisiert provisionieren und installieren. Ab SharePoint 2013 gab es jedoch folgendes Problem: Es gab PowerShell Cmdlets, die eine automatisierte, skriptgesteuerte Verteilung von SP-Add-ins zuließen. Jedoch wurde nur die Hälfte des Weges gegangen. Erstens konnte man nur unter Missachtung von Best Practices Add-in-Instanzen in Produktivumgebungen einem Web hinzufügen. Zum anderen musste bei jeder App-Instanz noch die Berechtigung des Add-ins manuell erfolgen. Dies ist natürlich bei einer App, die vorinstalliert auf vielen Webs vorhanden sein soll, eine große Erschwernis. Um das zu umgehen, konnte man wieder nur auf instabile Methoden wie das Web Scraping zurückgreifen, bei dem man praktisch mit PowerShell die Website öffnete, dann im Internet Explorer den HTML-Button anklickte und hoffen musste, dass alles fehlerfrei funktionierte. Nun sind diese Zeiten aber vorbei. Na ja, fast, zumindest für SharePoint Online. Wann und ob dies überhaupt auf SharePoint 2016 implementiert wird, war zum Zeitpunkt des Schreibens dieses Artikels noch nicht bekannt.

Architektur des ALM-API

Die Architektur, die nun eine offizielle und direkte Verbindung mit dem App Catalog hat, vereinfacht das Hinzufügen, Installieren, Aktualisieren, Deinstallieren und Löschen von Lösungen (**Abb. 1**).

Wenn es um automatische Provisionierung ging, waren ALM und SP noch nie die besten Freunde. Jedenfalls nicht mehr seit der Version 2013, mit der die Add-ins eingeführt wurden.

Zu **Abbildung 1** ist zu sagen, dass die Bezeichnungen der Befehle wie *add* oder *deploy* dem echten Wording in den einzelnen APIs entspricht. Soll von einer Bibliothek z. B. durch einen Administrator eine Lösung zu einem Katalog hinzugefügt werden, wird dieser Vorgang immer das Schlüsselwort *add* beinhalten. Gleiches gilt für die anderen Befehle in der Abbildung. Mithilfe dieses API können Add-ins der Form **.app* und SPFx-(SharePoint-Framework-)Applikationen mit der Dateiendung **.sppkg* verteilt werden. Die Befehle haben folgende Bedeutungen:

- *add* lädt eine angegebene Lösung zum App Catalog hoch. Diese ist dann noch nicht für die Installation freigegeben.
- *deploy* bewirkt das Bereitstellen der Lösung, damit sie auf einzelnen Sites installiert werden kann.
- *retract* ist das Gegenstück zu *deploy* und deaktiviert diese Lösung. Bereits installierte Instanzen funktionieren dann Tenant-weit nicht mehr
- *remove* ist das Gegenstück zu *add* und bewirkt das Entfernen der Lösung aus dem App Catalog. Ist die Lösung zu dem Zeitpunkt aktiviert, wird vorher der Befehl *retract* automatisch ausgeführt.
- *install* installiert eine Instanz der aktivierten Lösung aus dem App Catalog in eine angegebene Site.
- *upgrade* aktualisiert eine Instanz einer Lösung auf einer angegebenen Site, falls die Lösung im App Catalog bereits eine höhere Version besitzt.
- *uninstall* bewirkt die Deinstallation der Lösungsinstanz auf einer angegebenen Site. Im Gegensatz zum Deinstallieren einer Lösung über die Weboberfläche wird die Instanz hierbei nicht in den Papierkorb verschoben.

Teile der Befehle werden an unterschiedlichen Orten ausgeführt. Während Operationen wie *add*, *deploy*, *retract* und *remove* im App Catalog ausgeführt werden, werden Befehle wie *install*, *upgrade* und *uninstall* auf der jeweiligen Site ausgeführt, auf der die Lösungsinstanz existieren soll.

Es werden noch weitere Befehle unterstützt, die dem Nutzer Informationen zu den einzelnen Lösungen im Katalog geben. Es kann auch erwartet werden, dass das API noch um weitere Befehle erweitert wird.

API-Zugriffsmöglichkeiten

Der Zugriff auf das API erfolgt in jedem Fall immer mit REST im Hintergrund. Es gibt fünf verschiedene Möglichkeiten, auf das API zuzugreifen:

- REST-API: Für einen direkten Zugriff mit client- oder serverseitigen Skripten (JavaScript, Node.js, C#, Java etc.) [1]
- C Sharp: mit der PnP Sites Core Component Library der Office-365-PnP-Initiative [2]
- PowerShell: Mithilfe der PnP PowerShell Cmdlets [1]
- JavaScript: Über die offizielle PnP-JS-Core-JavaScript-Bibliothek (einfacher als direkt über das REST-API zu gehen) [3]
- Office 365 CLI: Ein Kommandozeilenwerkzeug, das plattformunabhängig ist (für Linux-/MacOS-Benutzer ohne PowerShell) [4]

Vorweg: Das Schönste ist, dass die letzten vier Optionen alle auf GitHub als Open-Source-Projekte verfügbar sind. Das heißt, man kann den gesamten Code einsehen, Verbesserungen vorschlagen oder Bugs mit beseitigen. Wichtig ist natürlich, dass alle Bibliotheken vom REST-API abhängig sind und somit niemals mehr unterstützen können als das API in SharePoint Online selbst. Jedoch vereinfachen sie den Zugriff sehr stark, sodass teilweise einzeiliger Code ausreicht. In den folgenden Abschnitten sollen Beispiele für solche Zugriffe gegeben werden, die sich am besten dazu eignen, sie später in einer Releasepipeline in VSTS zu integrieren.

Zugriff mit PowerShell

Wenn es um Automatisierung geht, ist PowerShell wahrscheinlich die Nummer eins. Im Vergleich zu C Sharp oder JavaScript gibt es kaum Overhead bei der Programmierung. Auf GitHub bekommt man das PowerShell Cmdlet, das Voraussetzung zur Nutzung

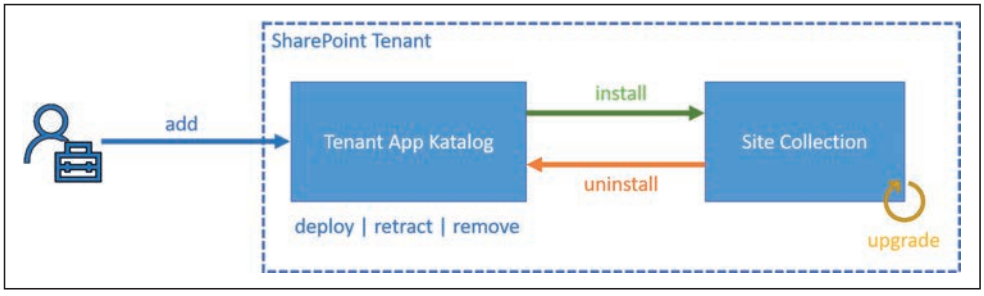

Abb. 1: Die ALM-API-Architektur

des ALM-API ist [5]. Dabei ist drauf zu achten, dass dies nur für SP Online gilt und entsprechend nur dieses PowerShell-Modul dafür genutzt werden kann. Ist das PowerShell-Modul für SP Online installiert, kann man sich mit dem Tenant direkt verbinden. Der folgende Code zeigt ein Beispiel zur Verbindung und Verteilung einer Lösung.

```
Connect-PnPOnline -Url https://mytenant.sharepoint.com/sites/AppCatalog/
$myTulevaApp = Add-PnPApp -Path ./MyTulevaApp.sppkg
                                        #wahlweise auch mit *.app
Connect-PnPOnline -Url https://mytenant.sharepoint.com/sites/
                                                    mysitecollection/
Install-PnPApp -Identity $myTulevaApp.Id
```

Hier kann man sehen, dass man sich für das Hinzufügen erst mit dem Katalog und beim Installieren dann mit der Zielseite verbinden muss. Die Berechtigungen werden hierbei automatisch übernommen. In den beiden folgenden Zeilen kann man dann der Vollständigkeit halber die Deinstallation aus der Site Collection und die Entfernung der App aus dem Tenant sehen.

```
Uninstall-PnPApp -Identity $myTulevaApp.Id
Remove-PnPApp -Identity $myTulevaApp.Id
```

Zugriff mit dem Office 365 CLI

Dieses Kommandozeilenwerkzeug ist noch sehr neu, wurde im vierten Quartal 2017 vorgestellt und bietet eine Möglichkeit, auf Office-365-APIs zuzugreifen. Das Office 365 CLI wurde vor dem Hintergrund der neuen Philosophie von Microsoft entwickelt, Multiplattformkompatibilität und Open Source zu unterstützen. Beide Kriterien erfüllt das Werkzeug. Das CLI ist nicht nur Open Source, sondern kann auch auf iOS-, Linux- und Windows-basierten Betriebssystemen verwendet werden. Dadurch kann es zur Automatisierung verwendet werden, falls die Verteilung nicht durch Agenten auf Windows-basierten Systemen geschieht, sondern durch andere Betriebssysteme. Das Werkzeug kann unter [3] heruntergeladen und installiert werden. Es handelt sich zwar noch um eine frühe Version, funktioniert jedoch schon zuverlässig. Das CLI wurde mit TypeScript geschrieben und nutzt JavaScript mit NodeJS als zugrunde liegende Technologie. Die Installation erfolgt mithilfe von npm: > *npm i -g @pnp/office365-cli*.

Nach der Installation kann im Command Prompt mit dem Befehl *office365* das Office 365 CLI gestartet werden. Im folgenden Code können wir ein Beispiel für die Verbindung und Installation einer Lösung sehen, wenn das CLI-Tool offen und bereit ist.

```
> spo connect https://mytenant.sharepoint.com
> spo app add -p MyTulevaApp.sppkg  #wahlweise auch mit *.app
> spo app install -i #MeineAppGuideHier# -s https://mytenant.sharepoint.
                                                    com/sites/mysitecollection/
```

Zu sehen ist hier, dass der Benutzer sich nicht erst mit dem Katalog verbinden muss, sondern lediglich mit dem Tenant selbst. Das Werkzeug kümmert sich selbst um das Finden des Tenant-Katalogs. Beim Installieren jedoch muss dann die Ziel-Site-Collection angegeben werden. Die Deinstallation und Entfernung verlaufen analog zur Installation:

```
> spo connect https://mytenant.sharepoint.com
> spo app uninstall -i #MeineAppGuideHier# -s https://mytenant.sharepoint.
                                                    com/sites/mysitecollection/
> spo app remove --id #MeineAppGuideHier#
```

Wenn in den Beispielen ein App Guide erwähnt wird, ist damit die Liste gemeint, die nach der Installation mit dem Befehl *spo app get* zurückgegeben wird.

Zusammenfassung

Der Automatisierung steht nichts mehr im Weg. Das vorgestellte Tool ist jedoch nur ein Schritt auf dem Weg zu einer kompletten Releasepipline und stellt lediglich die Möglichkeit bereit, speziell für SharePoint Lösungen zu veröffentlichen. Die Entwicklung des API bietet erstmals die Möglichkeit, ohne unsaubere Workarounds den Prozess ordentlich auf die Beine zu stellen.

Sebastian Schütze ist seit zehn Jahren Webentwickler. Er hat sich seit fünf Jahren auf SharePoint spezialisiert und in Folge der Cloud-Technologien in Azure die Möglichkeiten von DevOps für sich entdeckt. Er arbeitet aktuell als Freelancer für verschiedene Firmen.

Links & Literatur

[1] https://docs.microsoft.com/en-us/sharepoint/dev/apis/alm-api-for-spfx-add-ins/

[2] https://github.com/SharePoint/PnP-Sites-Core/

[3] https://github.com/SharePoint/office365-cli/

[4] https://github.com/SharePoint/PnP-JS-Core/wiki/Working-With:-ALM-API/

[5] https://github.com/SharePoint/PnP-PowerShell/