

Implementierung eines Chatbots zur Ausführung von Prozessen

STUDIENARBEIT T2_3201

für die Prüfung zum

Bachelor of Science

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Simon Lay

Abgabedatum 1. Juni 2020

Bearbeitungszeitraum	6 Monate
Matrikelnummer	1536357
Kurs	TINF17B3
Ausbildungsfirma	Siemens AG Karlsruhe
Betreuer der DHBW	Aydin Mir Mohammadi

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit T2_3201 mit dem Thema: „Implementierung eines Chatbots zur Ausführung von Prozessen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum

Unterschrift

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Ort Datum

Unterschrift

Zusammenfassung

Die vorliegende Studienarbeit wurde im Rahmen des DHBW-Studiums Bachelor of Science Informationstechnik im fünften und sechsten Semester verfasst. Sie behandelt die Entwicklung eines Chatbots zum Orchestrieren von Anwendungen (fachlichen Prozessen). Genauer zum Starten und Beenden von virtuellen Maschinen innerhalb der Cloud-Computing-Plattform Microsoft Azure. Der Chatbot soll zum Beispiel in Microsoft Teams integriert werden können. Hier sollen die Benutzer des Chatbots Nachrichten senden und Empfangen, um dem Chatbot Aufträge wie: „Starte die Siemens virtuelle Maschine“ oder „Fahre die PKI VM Landschaft hoch“ mit zugeben. Hierbei muss der intelligente Chatbot, anhand der natürlich sprachlichen Konversation, selbst herausfinden, welche VMs genau gestartet oder beendet werden sollen. Beispielsweise, welche VMs immer zusammen gestartet werden.

Der Mehrwert eines solchen Bots ist es, das der Anwender in der Lage ist, einen technischen Prozess zu starten, ohne die technischen Bedienung tatsächlich zu kennen. Der Bot unterstützt ihn bei dem Starten des technischen Prozess und übernimmt die Bedienung. Somit muss der Anwender wenig bis gar kein Wissen über das Azure Portal besitzen, aber kann trotzdem die benötigten VMs starten oder wieder herunter fahren.

Der Chatbot ist mithilfe des Microsoft Bot Frameworks in C# erstellt worden. Für das natürliche Sprachverständnis nutzt er den Dienst LUIS von Microsoft. Zur Authentifizierung wird Azure Active Directory mit der Azure Service Management Berechtigung (User Impersonisation) verwendet und zur Steuerung der Azure Ressourcen, die Azure Management API für .NET.

Inhaltsverzeichnis

Abbildungsverzeichnis	VI
Abkürzungsverzeichnis	VII
1 Einleitung	1
1.1 Grundlagen	1
1.1.1 Azure	1
1.1.2 Azure Ressourcen	1
1.2 Problemstellung	3
1.3 Erwartetes Ergebnis	3
1.4 Vorgehensweise	4
2 Chatbots	5
2.1 Chatbots im Allgemeinen	5
2.2 Microsoft Bot Framework in C#	6
2.2.1 Allgemeines	6
2.2.2 Dialoge	7
2.3 Implementierung	9
2.3.1 Einzelne VMs starten oder beenden	10
2.3.2 VM-Landschaften starten	11
2.3.3 VM-Landschaften herunterfahren	12
2.3.4 Konfigurationsdialoge	12
3 Natural Language Understanding	14
3.1 Allgemeines über Natural Language Understanding	14
3.2 LUIS	15
3.3 Benutzer	16
3.4 Implementierung	16

Inhaltsverzeichnis

4 Funktionen des Chatbots	18
4.1 Eingrenzung der Funktionen	18
4.2 Implementierung	18
4.3 Authentifizierung	19
4.4 Identifikation der Landschaften	22
4.5 Datenspeicherung	23
5 Schluss	25
5.1 Veröffentlichung in Kanälen	25
5.1.1 Möglichkeiten	25
5.1.2 Teams	26
5.2 Fazit	26
5.3 Ausblick	27
6 Anhang	VIII
6.1 Link zum Repository	VIII
6.2 Beispielkonversationen	VIII
Literaturverzeichnis	XIII

Abbildungsverzeichnis

1.1	Azure (Quelle: [25])	2
2.1	CoreBot Template Projektstruktur (Quelle: Screenshot)	7
2.2	Choice Prompt (Quelle: Screenshot)	8
2.3	Dialog Klassen des Chatbots (Quelle: Screenshot)	10
3.1	LUIS: StartSingleVm Beispieläußerungen (Quelle: Screenshot)	17
4.1	Azure Service Management Berechtigungen (Quelle: Screenshot)	20
4.2	Authentifizierungs-Prompt (Quelle: Screenshot)	21
6.1	Teams-Integration: Begrüßung (Quelle: Screenshot)	IX
6.2	Teams-Integration: Starte VM (Quelle: Screenshot)	IX
6.3	Teams-Integration: Starte VM-Landschaft (Quelle: Screenshot)	X
6.4	Teams-Integration: Beende VM-Landschaft (Quelle: Screenshot)	X
6.5	Teams-Integration: Konfiguriere: VM zu Landschaft hinzufügen 1 (Quelle: Screenshot)	XI
6.6	Teams-Integration: Konfiguriere: VM zu Landschaft hinzufügen 2 (Quelle: Screenshot)	XI
6.7	Teams-Integration: Konfiguriere: Notwenige VMs einer Landschaft (Quelle: Screenshot)	XII

Abkürzungsverzeichnis

VM	Virtuelle Maschine
SDK	Software Development Kit
ARM	Azure Resource Manager
JSON	JavaScript Object Notation
XML	Extensible Markup Language
LUIS	Language Understanding Intelligent Service
JWT	JSON Web Token
NLU	Natural Language Understanding
NLP	Natural Language Processing
CLI	Command Line Interface

1 Einleitung

1.1 Grundlagen

Zum Betreiben von Softwaresystemen wird IT-Infrastruktur benötigt. Diese kann lokal verfügbar oder über das Internet verteilt liegen. In dieser Studienarbeit wird sich auf Infrastruktur der Plattform Azure konzentriert:

1.1.1 Azure

Microsoft Azure ist eine Cloud-Computing-Plattform und bietet eine Menge verschiedenster Ressourcen. Ressourcen sind zum Beispiel virtuelle Computer, Datenbanken, Speicher oder Netzwerke. Auf diese Ressourcen müssen unter Umständen mehrere Entwickler oder Stakeholder steuernden Zugriff haben. Cloud-Computing-Plattformen sind typischerweise über das Internet erreichbare IT-Infrastruktur. Diese stellen Speicherplatz, Rechenleistung und Anwendungssoftware zur Verfügung. Bei einer Cloud-Computing-Plattform muss der Speicherplatz, die Rechenleistung oder die Software nicht lokal zur Verfügung stehen, sondern kann in der Cloud gehostet werden. Dies bringt mehrere Vorteile mit sich, zum Beispiel Platzeinsparung für die Hardware, weniger Administrator Tätigkeiten, Skalierbarkeit und schneller verfügbare Infrastruktur. Es kann immer genau so viel der Ressourcen in Anspruch genommen werden, wie wirklich benötigt wird.

Die Infrastruktur kann über verschiedenste Schnittstellen und Protokolle erreicht werden. [8]

1.1.2 Azure Ressourcen

In Abbildung 2.1 ist eine Veranschaulichung der Struktur von Azure zu sehen. Eine Person kann ein Abonnement erwerben und darin mehrere Ressourcengruppen erstellen. Jeder Ressourcengruppe können eine oder mehrere Ressourcen zugeordnet werden. Jede Ressource kann sich maximal in einer Ressourcengruppe befinden. Ressourcen müssen sich in einer Gruppe

1 Einleitung

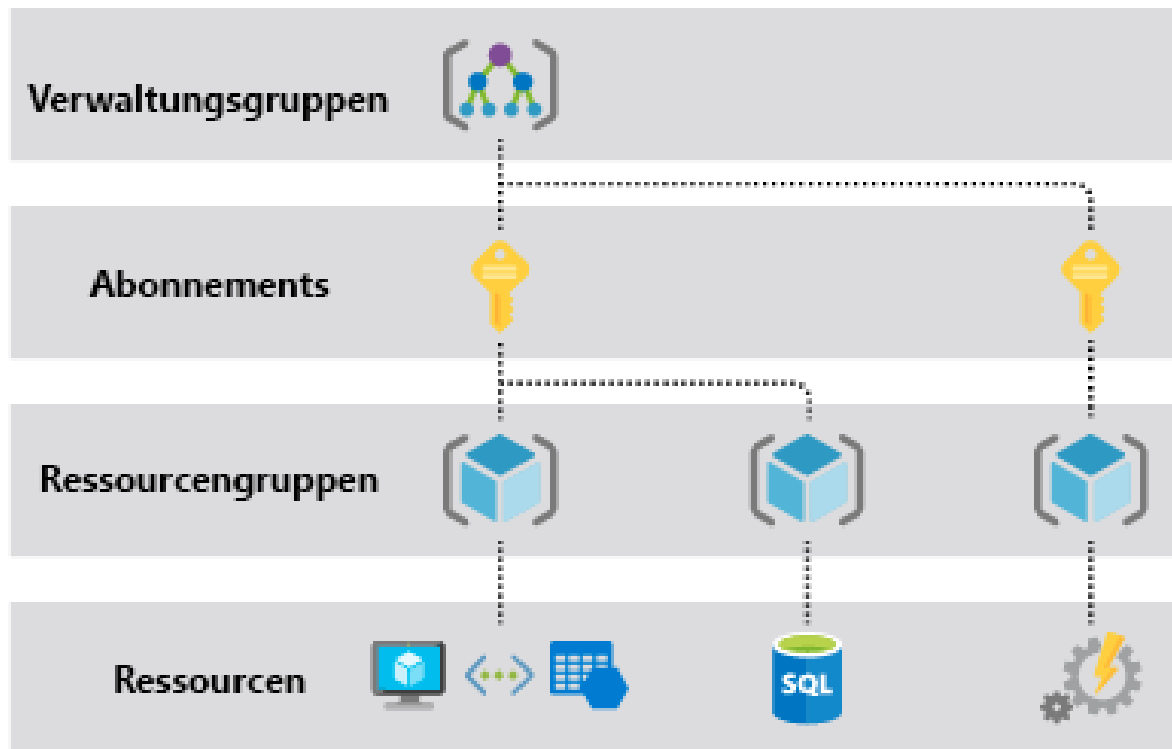


Abbildung 1.1: Azure (Quelle: [25])

befinden, können allerdings auch verschoben werden. [23] Verschiedene Ressourcen können über die Ressourcengruppen kategorisiert werden. Also Ressourcen die in irgend einer Art und Weise zusammen gehören, können in der selben Ressourcengruppe liegen. Es gibt zusätzlich aber auch die Möglichkeit einer Ressource Tags hinzuzufügen. Ein Tag in Azure besteht jeweils aus einem Schlüssel-Wert-Paar. Pro Ressource kann jeder Schlüssel nur einmal vorkommen. Es sind aber mehrere Tags mit unterschiedlichen Schlüsseln möglich. Somit lassen sich Ressourcen aus verschiedenen Ressourcengruppen, über den selben Tag trotzdem zueinander zuordnen. Zu Ressourcen zählen alle Dienste, die in einem Azure-Abonnement erstellt werden können. Hierzu zählen zum Beispiel eine Virtuelle Maschine (VM) oder Datenbanken. Microsoft bietet für den Zugriff auf Ressourcen und Ressourcengruppen in Azure mehrere Möglichkeiten: [5]

- Azure-Portal (Web-Oberfläche)
- Azure PowerShell
- Azure Command Line Interface (CLI)

1 Einleitung

- Azure SDKs (zum Beispiel .NET)
- REST-Schnittstelle

Hiermit kommen wir zur Problemstellung dieser Arbeit.

1.2 Problemstellung

Viele Prozesse müssen in ihrer jeweiligen Systemumgebung angestoßen werden. Hierzu bedarf es zum einen detaillierte Kenntnisse des Prozesses und seine Parameter, als auch einen Zugangspunkt zur Systemumgebung des Prozesses.

Am Beispiel der Systemumgebung Azure braucht der Benutzer zum einen einen Azure Account der Zugriff auf die Ressource innerhalb von Azure hat. Ebenfalls muss sich der Benutzer auch innerhalb des Azure Portals auskennen. Hierbei benötigt er Wissen zu den Ressourcengruppen. Er benötigt Wissen zu den einzelnen Ressourcen, wie zum Beispiel die virtuelle Maschine. Und er benötigt Wissen zu den einzelnen Operationen, wie zum Beispiel das Starten und Beenden. Des weiteren muss er Kenntnisse darüber haben, welche Ressourcen zusammen gehören und welche er genau für seine Absichten benötigt. Zusammengehörende Ressourcen werden im nachfolgenden „Landschaft“ genannt. Es kommt oft vor, dass Ressourcen für ihre Funktionalität auf weitere Ressourcen angewiesen sind. Also eine Landschaft, zum Beispiel genannt Siemens Landschaft, kann mehrere Ressourcen enthalten, wie zum Beispiel PKI VM, Main VM und UserDatabase. Nun will der Benutzer vielleicht nur die Main VM starten, vielleicht aber auch die gesamte Siemens Landschaft und weiß nicht welche Ressourcen hierbei alle benötigt werden, beziehungsweise überhaupt Teil der Landschaft sind. Möglich wäre, dass die PKI VM nur optional zu starten ist oder beide VMs zwingend notwendig sind. Hierbei sind viele Variationen möglich, in denen sich ein Benutzer, welcher wenig Kenntnisse über die Ressourcen besitzt, nur schwer zurecht findet. Dieses Wissen jedem Benutzer anzulernen, ist eine zwar meist notwendige, jedoch aufwendige Aufgabe, welche mithilfe des Chatbots umgangen werden soll.

1.3 Erwartetes Ergebnis

Die Idee hinter dieser Studienarbeit ist es, einen Chatbot zu entwickeln, der das Starten und Beenden der virtuellen Maschinen übernimmt. Dieser verwendet eine der weiteren Schnittstellen zum Starten und Beenden der VMs, abseits des Web-Portals, welches der Benutzer verwenden

1 Einleitung

würde und in welchem er sich auskennen müsste. Die Benutzer müssten dem Chatbot somit nur noch mitteilen, was sie gemacht haben wollen und der Chatbot erledigt es für sie. Der Chatbot muss zum einen Zugriff auf alle beteiligten Ressourcen in Azure haben. Zum anderen soll der Chatbot eine Intelligenz besitzen, mithilfe der er dem Benutzer helfen kann, sich in den Ressourcen-Landschaften innerhalb von Azure zurecht zu finden und die richtigen Prozesse anzustoßen. Genauso muss er Informationen über die einzelnen Ressourcen und Landschaften haben, um den Benutzer auch damit zu unterstützen. Dafür sollen dem Bot Daten zur Verfügung stehen, welche er von Benutzern oder Administratoren mit großem Wissen über die Ressourcen und Ressourcen-Landschaften erlangen kann und welches er zur Unterstützung für Nutzer mit weniger Wissen benutzen kann. Diese Daten bilden ein fachliches Modell über die Ressourcen und Landschaften ab. Vorerst konzentriert sich diese Studienarbeit nur auf das Starten und Beenden von virtuellen Maschinen innerhalb von Azure. Die Möglichkeit zur Erweiterung der Funktionen ist jedoch gegeben.

Das erwartete Ergebnis ist somit ein intelligenter Chatbot zum Starten und Beenden von VM-Landschaften oder einzelnen virtuellen Maschinen. Implementiert in C# .NET. Mit dem Chatbot soll von Microsoft Teams aus kommuniziert werden können.

1.4 Vorgehensweise

Zuerst wurde sich mit der Grundlagenaneignung in den Bereichen Chatbots im Allgemeinen, Natural Language Understanding und dem Microsoft Chatbot Framework in C# beschäftigt. Zu diesen Themen existieren jeweils Kapitel im Folgenden dieser Arbeit. Dann wurden die Funktionen, die der Chatbot haben soll, genau eingegrenzt und definiert. Jetzt wurde mit der Implementierung des Chatbots und dem Trainieren des Natural Language Understanding Dienstes begonnen. Dannach wurden die Funktionen des Chatbots implementiert und der Chatbot in eine Umgebung eingebunden, in der er später verwendet werden soll. Zuletzt wurde der Chatbot getestet, verbessert und über Erweiterungsmethoden nachgedacht.

2 Chatbots

2.1 Chatbots im Allgemeinen

Ein Bot ist eine Applikation, die automatisiert Aufgaben erledigt. [7] Genau dies soll der Chatbot in diesem Projekt tun. Er soll automatisiert Cloud-Dienste Starten und Beenden können.

Bei einem Chatbot ist diese Funktion noch um das Chatten erweitert. Er hat einen Bereich zur Texteingabe und einen zur Textausgabe, ähnlich einer Konsole oder eines Nachrichtendienstes. Mithilfe dieser kann der Benutzer mit einem technischen System kommunizieren. Merkmal eines Chatbots ist es, das man in natürlicher Sprache mit dem technischen System kommunizieren kann. Chatbots können in zwei Arten unterschieden werden. Zum einen Chatbots, dessen Funktionen sich auf die Kommunikation beschränken und zum anderen Chatbots, welche eine Schnittstelle zu Diensten außerhalb des Chats bieten. In diesem Projekt soll zweiteres der Fall sein.

Heutzutage kennt man Chatbots von Webseiten, welche bei Support-Anfragen erst einmal versuchen das Problem ohne einen Mitarbeiter zu lösen. Virtuelle persönliche Assistenten, wie Alexa oder Siri, greifen ebenfalls auf Chatbots zu. Chatbots können zudem in bereits bestehende Chat-Dienste integriert werden. Hier unterscheidet sich der Chat für den Benutzer nicht von einem Chat mit einer richtigen Person, jedoch auf der anderen Seite des Chats antwortet der Chatbot. [26] [10]

Ein Chatbot zerteilt den Anfragetext, interpretiert diesen nach gegebenen Mustern und antwortet ebenfalls mit vom Programmierer vordefinierten Antworten. Je nach Intention der Anfrage, ist die vordefinierte Antwort noch mit Daten gefüllt, die der Chatbot von zum Beispiel einem Dienst abfragt. Den Part des Verstehens einer Anfrage und der Intention dahinter, wird im Kapitel Natural Language Understanding näher erläutert.

2.2 Microsoft Bot Framework in C#

2.2.1 Allgemeines

Das Microsoft Bot Framework bietet mehrere Hilfestellungen, um einen Chatbot in .NET zu erstellen. Mit dem Bot Builder Software Development Kit (SDK) lässt sich Visual Studio um mehrere Templates erweitern, was die Entwicklung erleichtert. Es gibt Templates um einen neuen Bot aufzusetzen, aber auch Templates, mit denen die Funktionsweise des Frameworks verstanden werden kann. Beispielsweise ein Bot der die Useranfragen als Echo wiedergibt oder Bots, die bereits kleinere Konversationen führen können. Mit dem Bot Framework Emulator lassen sich die Chatbots debuggen. Mit diesem Tool kann der Entwickler sich mit einem Chatbot lokal verbinden und Nachrichten innerhalb eines Chats senden und empfangen. Somit ist man in der Lage den Chatbot selbst zu testen oder ihn von einer weiteren Person testen zu lassen [1] [2].

Das Microsoft Bot Framework bietet dem Entwickler eines Chatbots viele Hilfestellungen. Sobald ein Template geöffnet wurde, ist die Struktur des Bots bereits vorgegeben [15]. In der Abbildung 3.1 ist die Projektstruktur des CoreBot Templates zu sehen. Dieses Template wurde auch in der Implementierung dieser Studienarbeit verwendet. Es handelt sich um ein ASP.NET Core Web Projekt. Der Template-Code ist gegliedert in die Bots, wobei es in diesem Template einen Dialog- und einen DialogAndWelcome-Bot gibt. Im Ordner Cards befinden sich sogenannte Adaptive Cards. Die WelcomeCard wird zum Beispiel beim Start des Chatbots vom DialogAndWelcome-Bot angezeigt und begrüßt den Nutzer. Im Ordner CognitiveModels befindet sich Code, der mit dem Sprachverstehen des Bots zu tun hat. Dazu mehr im Kapitel Natural Language Understanding. Im Ordner Controllers befindet sich der BotController der die Web-Aufrufe der ASP.NET Applikation steuert. Im Ordner DeploymentTemplates befinden sich Vorlagen zur Bereitstellung des Bots über den Azure Resource Manager (ARM).

Im Ordner Dialogs befinden sich, wie der Name schon sagt, die Dialoge, welche der Chatbot führen kann. Ein MainDialog, von wo aus alle weiteren Dialoge aufgerufen werden existiert immer. Im CoreBot Template existieren hier beispielhaft noch ein BookingDialog, mit welchem alle notwendigen Daten für eine Flugbuchung gesammelt werden können und einen DateResolverDialog der wiederum vom BookingDialog aufgerufen wird, mit dem das Datum des Flugs bestimmt werden kann. Zudem existiert noch ein CancelAndHelpDialog in dem Benutzereingaben wie „cancel“ oder „help“ behandelt werden können. Die Klasse AdapterWithErrorHandler fängt nicht behandelte Ausnahmen ab und gibt die Möglichkeit in der Konversation hierauf einzugehen. Zum Beispiel sagt er so was Ähnliches wie: „Es gab einen Error, bitte repariere

2 Chatbots

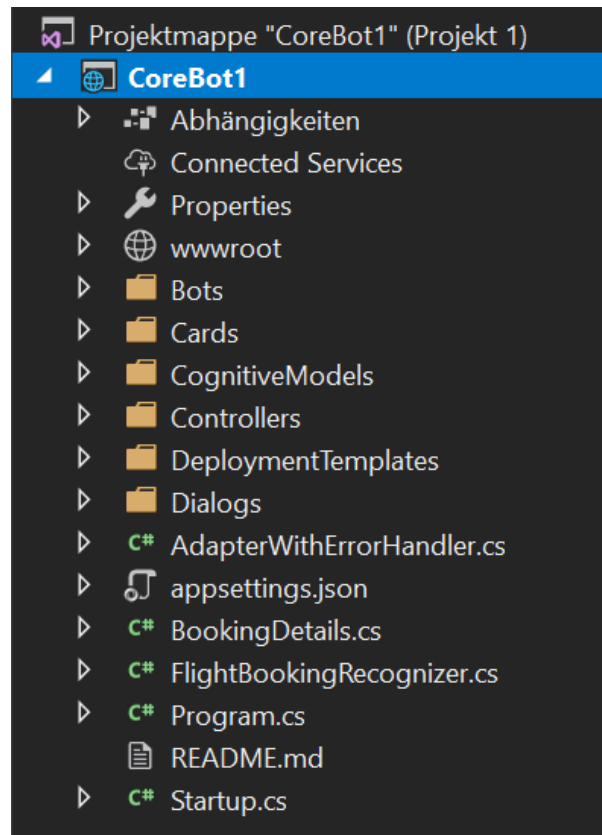


Abbildung 2.1: CoreBot Template Projektstruktur (Quelle: Screenshot)

den Quellcode.“ Im `appsettings.json` befinden sich IDs und Schlüssel für Microsoft Dienste. Genauso wie auf die `FlightBookingRecognizer`-Klasse wird hierauf später noch eingegangen. Die `BookingDetails`-Klasse sammelt die Daten für die Buchung des Flugs. `Program.cs` und `Startup.cs` sind die Einstiegspunkte der Applikation.

2.2.2 Dialoge

Das Microsoft Bot Framework unterstützt die Entwicklung eines Chatbots durch viele Funktionen, welche in den Dialogen verwendet werden können. Dialoge sind dazu da, um die Konversation zwischen Benutzer und Bot zu verwalten. Jeder Dialog erfüllt eine Aufgabe und führt spezifische Schritte der Reihe nach durch. Die Reihenfolge der aufgerufenen Dialoge lässt sich abhängig von Gegebenheiten steuern. Dialoge können hierbei an unterschiedlicher Stelle aufgerufen und auch mehrfach wiederverwendet werden.

Die Dialogbibliothek des Frameworks bietet viele Möglichkeiten, wobei in dieser Arbeit nur

2 Chatbots

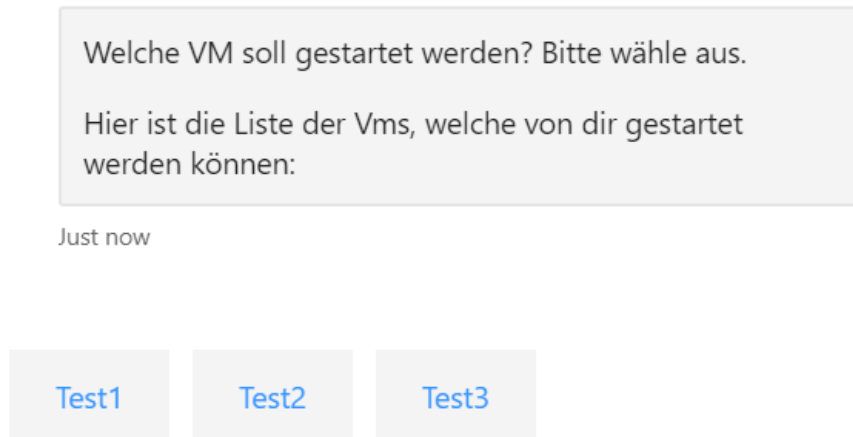


Abbildung 2.2: Choice Prompt (Quelle: Screenshot)

einzelne, häufig verwendete Komponenten erwähnt werden:

Ein typischer Dialog, genannt: Wasserfall-Dialog, besteht aus mehreren Schritten. Diese Schritte werden der Reihe nach abgelaufen. Der Chatbot kann bei jedem Text eine vom Entwickler vorgefertigte Nachricht senden. Diese Nachricht kann natürlich auch variierende Daten enthalten.

Ein weiterer Dialog ist der Dialog „Eingabeaufforderung“. Hierbei fragt der Chatbot nach einer Information, welche der Benutzer ihm dann mitteilen soll. Es gibt vom Framework unterschiedliche bereits vorgefertigte Dialoge, wenn der Chatbot zum Beispiel nach einem Text, einer Bestätigung oder einer Auswahl aus einer Liste fragen will. Hierfür können vorgefertigte Prompt-Dialoge verwendet werden. In Abbildung 2.2 ist ein sogenannter Choice Prompt zu sehen, bei dem der User eine Auswahl zwischen verschiedenen Möglichkeiten bekommt und dazwischen auswählen kann. Für diese Art von Prompt gibt es verschiedene Styles, zwischen denen der Entwickler auswählen kann. In diesem Fall kann die Antwort durch einfaches Klicken einer der Buttons ausgewählt werden. Die Antwort wird automatisch in den Chat geschrieben. Es gibt allerdings auch weitere Styles, welche zum einen anders aussehen und zum Beispiel den User dazu auffordern die Nummer des ausgewählten Objekts in den Chat zu schreiben. Der Entwickler kann die Auswahl der Styles auch auf automatisch stellen, hierbei entscheidet das Framework, je nach dem wie viele Auswahlmöglichkeiten es gibt, welchen Style es verwendet.

Die Dialoge [17], vor allem die Schritte innerhalb eines Wasserfalldialogs, werden vom Framework mit einem StepContext-Objekt und einem CancellationToken versehen [16]. Das CancellationToken kann darüber Benachrichtigen wenn Vorgänge abgebrochen werden sollen.

2 Chatbots

Der StepContext bietet sehr viele Möglichkeiten. Unter anderem bietet er die Möglichkeit Aktivitäten zum Beispiel in Form einer Nachricht für den Chat zu senden. Des weiteren können über dieses Objekt weitere Dialoge gestartet, alte beendet oder im aktuellen Dialog einen Schritt weiter gesprungen werden. Das Objekt behält dabei alle notwendigen Daten über die aktuelle Position in den Dialogen und Schritten innerhalb eines Dialogs.

Das StepContext-Objekt beinhaltet ebenfalls eine Eigenschaft Optionen, in der einem neu gestarteten Dialog ein weiteres Objekt übergeben werden kann. Im CoreBot Template kann das zum Beispiel das aktuelle BookingDetails-Objekt sein. Eine weitere Eigenschaft ist das Result-Objekt, welches jeder Schritt zurück geben kann. An einem Beispiel erklärt: Ein Schritt in einem Dialog überprüft ob die Eigenschaft Startflughafen bereits in den BookingDetails enthalten ist. Wenn ja gibt er als Result diesen Wert zurück. Wenn nein startet er einen weiteren Dialog, bzw. eine Eingabeaufforderung, die am Ende auch den Startflughafen als Result zurück gibt. In beiden Möglichkeiten bekommt der folgende Schritt im Result-Objekt den Startflughafen.

2.3 Implementierung

Hier wird die Implementierung der Dialoge beschrieben. Neben der Implementierung der Funktionen und Daten des Chatbots lag in diesem Kapitel der größte Implementierungsaufwand. Auch wenn das Framework viele Möglichkeiten bietet und Aufgaben abnehmen kann.

In Abbildung 3.2 sind die Dialog-Klassen des Chatbots zu sehen. Gestartet wird im Main-Dialog, nach dem der Bot (DialogAndWelcomeBot) den Nutzer begrüßt hat. Im MainDialog beginnt ein Wasserfalldialog mit folgenden Schritten:

Prompt

Zeigt ein Feld mit Text und einen Button an. Beim Klick auf den Button kann sich der Benutzer einloggen.

Login

Überprüft das Token, welches vom Anmelde-Dialog zurück kommt und sendet Infos darüber an den User.

Intro

Frägt den User, was der Bot machen soll.

Act

Ruft LUIS mit der Antwort des Users auf und startet abhängig von der Intention einen

2 Chatbots

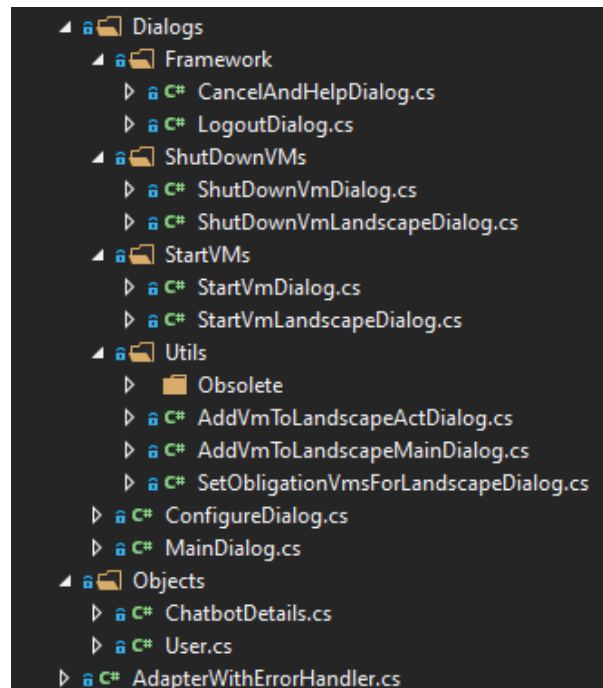


Abbildung 2.3: Dialog Klassen des Chatbots (Quelle: Screenshot)

weiteren Dialog.

Final

Hier kommen alle Dialoge, welche im Schritt davor gestartet wurden, wieder zusammen. Der Bot gibt Auskunft darüber, ob der Dialog erfolgreich abgeschlossen wurde.

Die Dialoge in den Ordnern StartVMs und ShutDownVMs, genauso wie der ConfigureDialog werden im Act-Schritt des Main-Dialogs aufgerufen. Es gibt jeweils einen Dialog dafür, das der Vorgang auf die gesamte Landschaft oder nur auf eine einzelne VM ausgeführt werden soll.

2.3.1 Einzelne VMs starten oder beenden

Bei den Dialogen StartVmDialog und ShutDownVmDialog werden folgende Schritte durchlaufen:

GetVM

Gibt dem User eine Auswahl an VMs welche er starten/beenden kann. Dieser Schritt wird übersprungen, wenn durch die Antwort von LUIS bereits bekannt ist, auf welche VM der Vorgang ausgeführt werden soll.

2 Chatbots

GetChoice

Nimmt die Auswahl des Benutzers entgegen.

Confirm

Nachdem alle Daten zum Vorgang bekannt sind, gibt der Chatbot dem User einen Überblick darüber, welche VMs jetzt gestartet/beendet werden und bittet den User um seine Bestätigung.

Final

Der Vorgang wird ausgeführt. Wenn nicht bestätigt wurde, wird der Vorgang nicht ausgeführt. In beiden Fällen wird der Dialog beendet.

2.3.2 VM-Landschaften starten

Bei dem Dialog StartVmLandscapeDialog werden folgende Schritte durchlaufen:

GetLandscape

Gibt dem User eine Auswahl an Landschaften welche er starten/beenden kann. Dieser Schritt wird übersprungen, wenn durch die Antwort von LUIS bereits bekannt ist, auf welche Landschaft der Vorgang ausgeführt werden soll.

GetChoice

Nimmt die Auswahl des Benutzers entgegen.

ObligationOrAll

Der Bot fragt den User, ob alle VMs der Landschaft oder nur die notwendigen gestartet werden sollen.

CheckForObligationVms

Falls die Auswahl auf die notwendigen VMs fällt, wird überprüft ob eine Konfiguration vorhanden ist. Falls nicht wird darauf hingewiesen und mit allen VMs der Landschaft fortgefahren.

Confirm

Es wird um eine Bestätigung für die zu startenden VMs innerhalb der Landschaft gebeten.

Final

Der Vorgang wird ausgeführt. Wenn nicht bestätigt wurde, wird der Vorgang nicht ausgeführt. In beiden Fällen wird der Dialog beendet.

2.3.3 VM-Landschaften herunterfahren

Bei dem Dialog `ShutDownVmLandscapeDialog` werden folgende Schritte durchlaufen:

GetLandscape

Gibt dem User eine Auswahl an Landschaften welche er starten/beenden kann. Dieser Schritt wird übersprungen, wenn durch die Antwort von LUIS bereits bekannt ist, auf welche Landschaft der Vorgang ausgeführt werden soll.

ListVms

Listet alle VMs innerhalb der Landschaft auf, welche heruntergefahren werden.

Confirm

Es wird um eine Bestätigung für die zu beendenden VMs innerhalb der Landschaft gebeten.

Final

Der Vorgang wird ausgeführt. Wenn nicht bestätigt wurde, wird der Vorgang nicht ausgeführt. In beiden Fällen wird der Dialog beendet.

2.3.4 Konfigurationsdialoge

Im Ordner `Utils` bleiben jetzt noch drei Dialoge übrig, die bisher noch nicht erklärt wurden. Genauso wurde der `ConfigureDialog` noch nicht erwähnt. Der `ConfigureDialog` kann aus dem `MainDialog` heraus aufgerufen werden, wenn keine VM gestartet werden soll, sondern die VMs und Landschaften nur konfiguriert werden sollen. Hierbei fragt er im ersten Schritt, um welche Konfiguration es sich handeln soll. Im zweiten Schritt, ruft er je nach Antwort entweder den `AddVmToLandscapeMainDialog` oder den `SetObligationVmsForLandscapeDialog` auf und der dritte Schritt beendet den Dialog wieder.

Im `AddVmToLandscapeMainDialog` werden folgende Schritte durchlaufen:

Start

Dieser Schritt wird beim ersten Aufruf übersprungen. Bei jedem weiteren Aufruf wird gefragt, ob noch eine weitere VM der Landschaft hinzugefügt werden soll. Falls nein wird der Dialog beendet

2 Chatbots

AskForVm

Alle VMs werden aufgelistet und gefragt, welcher VM eine Landschaft hinzugefügt werden soll.

Act

Der AddVmToLandscapeActDialog wird aufgerufen.

Final

Der Start Schritt wird wieder aufgerufen (rekursiv).

Im AddVmToLandscapeActDialog werden folgende Schritte durchlaufen:

ListTags

Es werden alle Landschaften aufgelistet, die der VM bereits zugeordnet sind.

Name

Es werden alle Landschaften aufgelistet, welche in Azure zu finden sind. Es wird nach dem Namen der Landschaft gefragt, welcher der VM zugeordnet werden soll

Confirm

Frägt nach einer Bestätigung.

Final

Der Vorgang wird ausgeführt. Wenn nicht bestätigt wurde, wird der Vorgang nicht ausgeführt. In beiden Fällen wird der Dialog beendet.

Im Ordner Obsolete sind alte Dialoge, welche während der Entwicklung des Bots zwar implementiert, jedoch in der Version, welche mit Erstellung dieser Arbeit aktuell ist, nicht verwendet werden.

3 Natural Language Understanding

3.1 Allgemeines über Natural Language Understanding

In der Taxonomie von Sprachen, wird zwischen 3 Arten unterschieden:

- Kommandos
- Künstliche Sprache
- Natürliche Sprache

Beispiele für Kommandos wären die Bedienung von Geräten wie Telefonen oder Navigationssystemen. Künstliche Sprache wäre zum Beispiel Sprache, welche wie im Flugfunk oder beim Militär standardisiert wurde. Die natürliche Sprache ist die Sprache, welche bei Ansprachen und Nachrichtensprechern, aber auch bei ungeplanten, spontanen Gesprächen und Konversationen zum Einsatz kommt.

Ein Chatbot muss je nach dem, was der Benutzer geschrieben hat, entscheiden, was er als nächstes zurückliefert. Damit der Chatbot nicht nur einer Kommandozeile gleicht, bei der nur auf ganz bestimmte Kommandos bestimmte Antworten kommen oder Prozesse angestoßen werden, soll der Chatbot natürliche Sprache verstehen können. Hierbei muss sich mit der Technologie Natural Language Understanding auseinandergesetzt werden.

Im Unterschied zum Natural Language Understanding (NLU) hört man im Zusammenhang mit Chatbots oft Natural Language Processing (NLP). [6] Beim Natural Language Processing wird der zu verarbeitende Text zum Beispiel nach Schlüsselwörtern durchsucht, um die Bedeutung des Textes herauszufinden. Daraufhin wird eine passende, gespeicherte Antwort gegeben. Dies ist bei einfachen Fragen eine Methode mit guter Trefferquote. Bei Fragen die komplizierter werden und bei denen eine Intention anstatt eines Schlüsselwortes gefunden werden soll, stößt das Natural Language Processing jedoch an seine Grenzen.

3 Natural Language Understanding

Hier wird das Natural Language Understanding eingeordnet. Der Anfrage des Nutzers muss eine Intention zugeordnet werden. Hierfür werden zum Beispiel folgende semantische wie auch kontextsensitive Eigenschaften des Textes untersucht:

- Subjekte und Objekte und sonstige Ausdrücke identifizieren
- Den Vorgang oder die Relation erkennen
- Pronomen richtig einordnen
- Und vieles mehr

3.2 LUIS

Language Understanding Intelligent Service (LUIS) ist ein Service der Firma Microsoft. Er basiert auf einem Machine-Learning-Verfahren und unterstützt Entwickler dabei, natürliche Sprache in Applikationen, Bots oder Internet-of-things-Geräte zu bringen. [3] LUIS kann zum Natural Language Understanding zugeordnet werden.

In der LUIS-Weboberfläche lässt sich der Service zusammenbauen. Hierbei werden vier Schritte durchlaufen:

- Erstellen
- Trainieren
- Testen
- Veröffentlichen

Im ersten Schritt können Intentionen angelegt werden. Zu jeder Intention können Beispiel-Anfragen erfasst werden. Diese sind dazu da, um LUIS im nächsten Schritt zu trainieren. Zusätzlich zu den Intentionen können Entitäten angelegt werden. Eine Entität ist wie eine Variable. Sie sind dazu da, um spezifische Informationen oder Werte aus einem Anfragetext zu erfassen. Es gibt mehrere Arten von Entitäten. Einfache Entitäten, Entitäten, welche aus mehreren anderen Entitäten zusammengesetzt sind, Listen oder reguläre Ausdrücke.

Im zweiten Schritt kann LUIS auf Basis der eingetragenen Beispiel-Anfragen trainiert werden. Der Entwickler muss dabei nichts weiter tun, als im Vorfeld die Anfragen einzutragen und das Training anzustoßen.

3 Natural Language Understanding

Die trainierte LUIS kann daraufhin auf ihre Funktion getestet werden, indem der Entwickler Beispiel-Anfragen schickt und überprüft, ob LUIS die richtige Intention erfasst hat.

Wenn der Entwickler mit dem Ergebnis zufrieden ist, kann diese Version von LUIS veröffentlicht und von zum Beispiel einer Chatbot-Applikation verwendet werden. Hierfür existiert das LUISGen Command Line Tool, mit dem eine C#-Klasse, alternativ auch ein Typescript-Interface, generiert werden kann. [4] [6] Das Tool generiert die Klasse aus einem Json, welcher zum Import/Export von LUIS gedownloadet werden kann. Dies unterstützt bei der Verarbeitung des Outputs von LUIS.

3.3 Benutzer

Bei der Interaktion zwischen Mensch und Maschine über natürliche Sprache wird oft zwischen zwei unterschiedlichen Benutzern unterschieden. Der kooperierende und der nicht kooperierende Benutzer. Hierbei ist der kooperierende Benutzer darauf aus, dass die Maschine ihn versteht. Er wird voraussichtlich versuchen seine Sprache klar und deutlich zu formulieren und so wenig Fehler oder Unklarheiten wie möglich zu verursachen.

Der nicht kooperierende Benutzer ist das Gegenteil hiervon. Er wird versuchen das System auf die Probe zu stellen. Wie der Name schon sagt, versucht er nicht mit der Maschine zu kooperieren, sondern gibt Eingaben in die Maschine, welche selbst vom Menschen schwierig aber gerade so noch verstanden werden können.

In Falle dieser Arbeit kann eher von einem kooperierenden Benutzer ausgegangen werden. Trotzdem muss bei der Implementierung berücksichtigt werden, dass natürliche Sprache sehr viele verschiedene Ausdrucksweisen für die selbe Intention haben kann.

3.4 Implementierung

Die Funktionen des Chatbots werden im nachfolgenden Kapitel weiter erläutert. Allerdings gibt es jetzt für jede Funktion des Chatbots eine Intention in LUIS. Diese wurde über das Web-Portal angelegt. Hier finden sich zum Beispiel die Intentionen „Starte einzelne VM“ und „Beende einzelne VM“. Für jede Intention wurden dann so viele Beispielanfragen wie möglich hinzugefügt. Die Intention „Starte einzelne VM“ kann hierbei auf viele verschiedene Arten beschrieben werden.

In der Abbildung 4.1 sind ein Teil der Äußerungen zu sehen, welche für die Intention „Starte

3 Natural Language Understanding

StartSingleVm

Bezeichnete Entitäten: **VmName**








 Bearbeiten	 Absicht neu zuweisen 	 Als Muster hinzufügen	 Löschen 
<input type="checkbox"/> Beispieläußerung	Bewertung 		
Geben Sie ein Beispiel dafür ein, was ein Benutzer sagen könnte,			
fahre VmName hoch	0.93		
starte VmName	0.94		
starte VmName vm	0.78		
fahre VmName vm hoch	0.78		
starte virtuelle maschine	0.89		
fahre vm hoch	0.84		

Abbildung 3.1: LUIS: StartSingleVm Beispieläußerungen (Quelle: Screenshot)

einzelne VM“ verwendet wurden. Die blau hinterlegte Schrift sagt hierbei aus, dass es sich um eine Entität handelt. Bei dieser Äußerung würde LUIS den hier stehenden Text als Name der Virtuellen Maschine zurück geben können.

Nachdem genügend Äußerungen hinzugefügt wurden, konnte mit dem trainieren, testen und veröffentlichen begonnen werden. LUIS wurde zu einer Azure Ressource hinzugefügt. Die Einstellungen für LUIS befinden sich im appsettings.json und werden von einem Recognizer verwendet.

Im Chatbot wird LUIS nach der Begrüßung, dem erfolgreichen Login und der Startnachricht aufgerufen. Also im Prinzip auf die erste Nachricht die der Nutzer sendet. Dies findet im MainDialog des Chatbots statt. Mithilfe des bereits erwähnten Tools LUISGen ist es möglich, die Antwort von LUIS als C#-Klasse zu repräsentieren. Mithilfe dieses Objekts lässt sich einfach auf die Intention und die Entitäten zugreifen.

Es ist notwendig, die Entitäten die von LUIS zurück kommen, vorerst noch zu prüfen. In dieser Studienarbeit überprüft der Chatbot zuerst, ob angegebene Entitäten, wie virtuelle Maschinen oder Landschaften überhaupt in Azure existieren. Wenn nicht muss ohne Entitäten fortgefahren werden. Es wird nun je nach Intention ein neuer Dialog gestartet.

4 Funktionen des Chatbots

Der Chatbot kann an diesem Punkt mittels LUIS die Intention der User-Anfrage erkennen und abhängig hiervon eine Antwort geben. Diese Antwort enthält Informationen passend zur Anfrage. Der Chatbot in diesem Projekt soll aber darüber hinaus noch zusätzliche Funktionen haben. Wie in der Einleitung beschrieben, soll der Chatbot in der Lage sein Azure Ressourcen zu steuern. Abhängig von Intention und Entitäten der Anfrage, sollen ganz bestimmte Dienste gesteuert werden.

4.1 Eingrenzung der Funktionen

Da die Möglichkeiten eines Chatbots unendlich groß sind, wurde sich in diesem Projekt auf die Steuerung von Azure Ressourcen fokussiert. Jedoch muss auch hier eine Eingrenzung stattfinden. Dabei muss festgelegt werden, welche Ressourcen genau gesteuert werden sollen und welche Parameter hierbei eine Rolle spielen. Nur wenn dies festgelegt ist, kann LUIS daraufhin trainiert werden und die Funktionen des Bots korrekt implementiert werden.

Vorerst wurde sich in dieser Studienarbeit deshalb auf das Starten und Beenden von virtuellen Maschinen beschränkt. Es sollen einzelne aber auch VM-Landschaften hochgefahren und wieder runter gefahren werden. Der Chatbot soll diese Funktionen ausführen können und den Benutzer dabei unterstützen, welche VMs zusammen in eine Landschaft gehören und welche VMs innerhalb einer Landschaft zusammen gestartet werden, also notwendig sind und welche optional, je nach Anwendungsfall, auch gestartet werden können.

4.2 Implementierung

Wie in der Einleitung beschrieben, gibt es viele Möglichkeiten Azure Ressourcen zu steuern. Die Methode zur Steuerung der Azure Ressourcen, die in diesem Projekt verwendet werden soll, kommt in einem NuGet-Paket: „Microsoft.Azure.Management.Fluent“ und ist Teil der .NET

4 Funktionen des Chatbots

Azure SDK. Dieses stellt Microsoft bereit. Hierüber kann sich ein Programm aus dem C#-Code heraus bei Azure authentifizieren und Funktionen ausführen, welche verschiedenste Operationen auf die Azure Ressourcen ausführen.[20] Beispiele für mögliche Operationen wären:

- Ressourcen erstellen
- Ressourcen updaten
- Ressourcen starten/beenden
- Informationen über Ressourcen abrufen

Allerdings, wie bei der Eingrenzung der Funktionen beschrieben, konzentriert sich diese Arbeit auf die Ressource: Virtuelle Maschine und auch hier nur auf das Starten und Beenden dieser.

Um dies in C# umzusetzen muss zuerst ein Azure-Objekt erstellt werden. Dies benötigt für die Authentifizierung Credentials, welche in diesem Falle über ein Token bereitgestellt werden. Dies ist im folgenden Kapitel Authentifizierung beschrieben. Mithilfe dieses Azure-Objektes können nun viele Vorgänge von Azure aus dem C#-Code heraus gesteuert werden. [20] Zum einen können alle verfügbaren virtuellen Maschinen abgerufen werden. Aus den C#-Objekten der virtuellen Maschinen können mehrere Informationen, wie der Name und die Tags oder auch den aktuellen Status der VMs gewonnen werden. Nun kann eine jeweilige VM gestartet werden. Dies dauert allerdings eine gewisse Zeit, weshalb diese Funktionen asynchron ausgeführt werden müssen. Das Beenden der VMs dauert ebenfalls lange. Hierbei gibt es zwei unterschiedliche Arten des Beendens. Einmal das einfache Power Off und zum anderen das Aufheben der Virtuellen Maschine. Das Power Off gleicht dem Herunterfahren eines normalen Computers. Allerdings werden hierbei noch Kosten von Azure berechnet, da nicht alle Ressourcen frei gegeben werden, die VM nicht vom Host entfernt und die Zuordnung nicht aufgehoben wird. Allerdings verliert man in diesem Fall auch seine dynamische IP-Adresse nicht. Für das Beenden von VMs über den Chatbot wurde sich in dieser Arbeit für das Aufheben der VM entschieden, da hier keine Kosten mehr entstehen und die VM ebenfalls heruntergefahren wird. [9] [21]

4.3 Authentifizierung

Um sicherzustellen, dass keine unbefugten Nutzer Zugriff auf die Funktionen des Chatbots bekommen, muss sich ein Benutzer zuerst bei dem Chatbot authentifizieren, bevor der Bot

4 Funktionen des Chatbots

API-Berechtigungen anfordern

< Alle APIs

Azure Service Management
https://management.azure.com/ Dokumente

Welche Art von Berechtigungen sind für Ihre Anwendung erforderlich?

Delegierte Berechtigungen
Ihre Anwendung muss als der angemeldete Benutzer auf die API zugreifen.

Anwendungsberechtigungen
Ihre Anwendung wird als Hintergrunddienst oder Daemon ohne angemeldeten Benutzer ausgeführt.

Berechtigungen auswählen Alle aufklappen

Suchtext eingeben

Berechtigung	Administratoreinwilligung erforderlich
<input type="checkbox"/> user_impersonation Access Azure Service Management as organization users (preview)	Ja

Abbildung 4.1: Azure Service Management Berechtigungen (Quelle: Screenshot)

Aufträge entgegen nimmt. Ebenso muss sich der Chatbot bei Azure authentifizieren, um Vorgänge wie das Starten und Beenden von VMs anzustoßen. Hierfür gäbe es zwei unterschiedliche Möglichkeiten, wie die Authentifizierung realisiert werden könnte.

Zum einen kann der Bot, bzw. Server, alle Rechte besitzen und der User authentifiziert sich beim Bot. Einmal beim Bot authentifiziert, hat der User so nun alle Rechte, welche der Bot auch hat. Der Bot müsste daher alle Rechte für alle Ressourcen haben, welche über den Chatbot gesteuert werden sollen und würde sich selbst bei Azure authentifizieren. Die Steuerung der Azure Ressourcen wäre benutzerunabhängig. Schon bei der Authentifizierung beim Chatbot müsste überprüft werden, ob der Benutzer überhaupt Zugriff auf den Bot haben darf.

Eine weitere Möglichkeit wäre, die Impersonierung. Der Benutzer authentifiziert sich beim Chatbot, und der Bot bekommt die Rechte des Benutzers übertragen. Somit darf der Chatbot genau die Dinge tun, welche der User ebenfalls tun dürfte. Der Chatbot hat somit gegenüber Azure ebenfalls die Rechte welche der Benutzer hat. Dies hat den Vorteil, das jeder Benutzer des Chatbots genau die Vorgänge ausführen darf, welche er auch in Azure dürfte. Der Nachteil ist aber, das die Rechte des Benutzers in Azure auch gepflegt werden müssen. Ohne die Rechte in Azure, können auch über den Chatbot die Ressourcen nicht gesteuert werden.

Hierbei wurde sich für die Impersonierung entschieden, da diese Version dem Chatbot an sich keine Rechte verleiht, sondern dieser die Identität des Users übernehmen kann. Wie in Abbildung 4.1 zu sehen, ist dies auch von Azure die gegebene Möglichkeit. Es sind keine weiteren über die Azure Service Management Berechtigung möglich.

Für die Authentifizierung wurde Azure Active Directory verwendet. [11] Die Authentifizierung läuft folgendermaßen ab: Innerhalb von Azure existiert eine Bot Applikation und eine

4 Funktionen des Chatbots

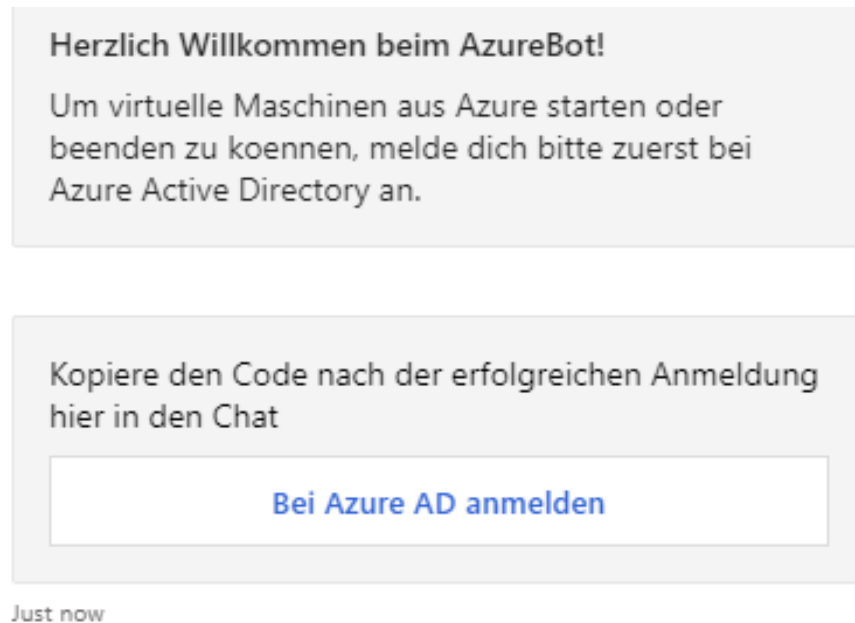


Abbildung 4.2: Authentifizierungs-Prompt (Quelle: Screenshot)

Azure Active Directory Identity Applikation. Diese ist beim Bot registriert. Hierbei gibt es mehrere unterschiedliche API-Berechtigungen. Für dieses Szenario wurde die Azure Service Management Berechtigung gewählt. Hier gibt es die User-Impersonisation-Berechtigung, welche Benutzer der selben Organisation Rechte für Erreichen des Azure Service Management gibt (Abbildung 4.1).

Im Chat mit dem Bot bekommt der User im Anmelde-Dialog jetzt einen Link, welcher er über einen Button öffnen kann. Dieser Button ist in Abbildung 4.2 zu sehen. Hier muss er sich mit seinem Microsoft-Konto anmelden. Er bekommt daraufhin einen kurzen Code, bestehend aus mehreren Zahlen. Sobald er diesen wieder im Chat an den Bot sendet, kann dieser dem Benutzer ein Token bereit stellen. Bei dem Token handelt es sich um JSON Web Token (JWT). Aus dem Token können mehrere Infos, wie zum Beispiel E-Mail und Name des Benutzers gezogen werden. Falls die Authentifizierung fehlschlägt, wird auch kein Token ausgestellt und dem Benutzer wird der Zugriff auf die Dialoge des Chatbots verwehrt, bis die Authentifizierung erfolgreich war.

Der ausgestellte Token kann vom Bot jetzt dafür verwendet werden, sich bei Azure mit den Rechten des Benutzers zu authentifizieren, um die im Kapitel zuvor beschriebenen Funktionen auszuführen. [13]

4.4 Identifikation der Landschaften

Wie bereits erwähnt gibt es Azure Ressourcen, im Falle dieser Arbeit virtuelle Maschinen, welche zusammen gehören, in diesem Kontext VM-Landschaften genannt. VM-Landschaften enthalten mehrere VMs, welche für unterschiedliche Szenarien zusammen gestartet werden können. Es gibt ebenfalls Szenarien, bei denen es notwendig ist, dass mehrere virtuelle Maschinen zusammen gestartet werden, da ansonsten die Funktionalität nicht gegeben ist.

Damit der Bot jetzt nicht nur in der Lage ist, einzelne VMs starten und beenden zu können, sondern auch komplette Landschaften hochfahren kann, muss er wissen welche VMs zu einer Landschaft hinzugehören. Hierfür muss der Bot auf Daten eines Modells zugreifen, welche die Fachlichkeit der Ressourcen und Landschaften abbilden. Azure bietet die Ressourcengruppen, um Ressourcen zu kategorisieren. Da die Ressourcengruppen allerdings auch für andere Kategorisierung verwendet wird, kann dieses Feature zum identifizieren von VMs zu einer Landschaft nicht verwendet werden.

Eine weitere Möglichkeit sind die Tags. Jeder Ressource können mehrere Tags hinzugefügt werden. Es besteht also die Möglichkeit jeder Ressource in einer Landschaft den Tag Schlüssel=Landschaft mit Wert=NameDerLandschaft hinzuzufügen. Somit können alle VMs mit diesem Tag zur Landschaft identifiziert werden.

Eine weitere Möglichkeit wäre es die Daten der VMs in einem externen Datenspeicher, wie zum Beispiel einer Datenbank oder einem Speicherkonto als Extensible Markup Language (XML) oder Json in Azure zu hinterlegen. Hier könnten die VMs ebenfalls zu einer Landschaft zugehörig abgespeichert werden.

Bei der Entwicklung dieses Chatbots wurde sich für die Methode mit den Tags entschieden. Dies bietet zum einen den Vorteil, dass die Ressourcengruppen für andere Zwecke verwendet werden können, jedoch lassen sich die Zugehörigkeit von Ressourcen zu einer Landschaft trotzdem noch über das Azure-Portal einsehen und ändern. Ebenfalls gibt es die Möglichkeit, nicht nur bei der Ressource die Zugehörigkeit zu einer Landschaft einzusehen, sondern auch alle VMs mit dem selben Landschafts-Tag anzeigen zu lassen. Bei einem externen Datenspeicher müsste man die Daten dort abändern oder einsehen, wo diese gespeichert sind, wenn man dies manuell machen will. Der entscheidende Vorteil der Tags ist die Nähe zu den Ressourcen in Azure. Andere Datenspeicher müssten zu den Ressourcen verlinkt werden, wobei Tags direkt einer Ressource hinzugefügt werden können. Somit können diese Ressourcen auch deployed, umbenannt oder verschoben werden, ohne dass dies Auswirkungen auf das fachliche Modell hat.

Mit dieser Lösung hat nun der Administrator der VMs die Möglichkeit in Azure manuell

4 Funktionen des Chatbots

jede VM zu einer Landschaft zuzuordnen. Und dies mit vergleichsweise wenig Aufwand. Zusätzlich soll der Bot jedoch auch die Möglichkeit haben, selbst VMs, welche nicht bereits einer Landschaft zugeordnet sind, zu einer Landschaft zuzuordnen. Auch dies geht mit den Tags in Azure. Dies bietet ebenfalls den Vorteil, das mit dem Bot generierte Daten zur Zugehörigkeit von VMs zu einer Landschaft, auch über das Azure-Portal gut visualisiert sind. Im Kapitel Chatbot/Implementierung sind die Dialoge, welche VMs zu einer Landschaft hinzufügen bereits beschrieben gewesen.

Die Funktionalität, nachdem der Chatbot die Infos gesammelt hat, bietet ebenfalls die Schnittstelle, welche auch für das Starten und Beenden der VMs verwendet wurde. Mithilfe der Azure Service Management API lassen sich VMs aktualisieren und somit ein Tag hinzufügen.

4.5 Datenspeicherung

Die Funktionalität welche bisher noch nicht beschrieben wurde, ist folgende: Innerhalb einer Landschaft kann es vorkommen, das einige VMs zwingend zu starten sind, wenn die Landschaft gestartet werden soll. Andere VMs, die sich in der selben Landschaft befinden, sind optional zu starten. Auch hierbei soll der Chatbot den User dabei unterstützen. Wie bei der Beschreibung der Dialoge bereits beschrieben, soll der Chatbot beim Starten einer Landschaft nachfragen, ob alle VMs der Landschaft gestartet werden sollen oder nur die notwendigen. Wenn sich für nur die notwendigen VMs entschieden wurde, sollte der Chatbot wissen, welche VMs hierbei gestartet werden sollen. Dafür muss der User dies dem Chatbot einmalig mitteilen und der Chatbot kann sich dies dann für spätere Vorgänge merken. Ebenfalls soll aber auch die Möglichkeit bestehen, die als notwendig vorgemerkten VMs wieder abzuändern.

Die Dialoge um diese Daten zu sammeln und zu verwenden wurden bereits beschrieben. Allerdings bleibt die Frage offen, wie und wo diese Daten abgelegt werden sollen. Hierbei gäb es mehrere Möglichkeiten. Es könnten ebenfalls die Tags, wie bei der Identifikation der Landschaften verwendet werden. Es könnte jedoch auch ein externer Datenspeicher, wie eine Datenbank oder ein Speicherkonto verwendet werden.

Da sowohl der Bot als App-Service/Web-App, also auch LUIS als Cognitive Service in Azure läuft, ist es zur Vollständigkeit sinnvoll, ebenfalls einen Speicher innerhalb von Azure zu verwenden. Hierbei wäre die Möglichkeit einer Datenbank oder eines Speicherkontos gegeben. Eine Datenbank wäre hierbei sehr viel mächtiger und würde mehr Möglichkeiten bieten. Da dies in diesem Szenario allerdings nicht notwendig war, könnte sich für ein einfaches Speicherkonto entschieden werden. [14] [18] Hier kann innerhalb eines Containers für jeden Nutzer ein

4 Funktionen des Chatbots

Textdokument angelegt werden, indem ein JavaScript Object Notation (JSON) abgespeichert wird. Dies bietet zusätzlich zur Übersichtlichkeit und Einfachheit der Implementierung auch den Vorteil, dass ein Mehrfachzugriff ausgeschlossen wird, da jeder User nur auf sein eigenes Dokument zugreifen muss.

Zuerst muss entschieden werden, ob die Auswahl von notwendigen und optionalen VMs pro Landschaft für jeden Nutzer gilt oder ob diese Auswahl von Nutzer zu Nutzer unterschiedlich sein kann. Hierbei wurde sich dafür entschieden, diese Auswahl für alle User gleich zu halten. Diese Entscheidung viel vor allem aufgrund der Tatsache, dass die Intention hinter diesem Bot ist, User mit wenig Wissen durch den Bot zu unterstützen. Somit sollen User mit wenig Wissen von den Daten der weiteren User profitieren, was bei einer von User zu User unterschiedlichen Auswahl nicht der Fall wäre.

Die Entscheidung für die Art der Datenspeicherung fiel ebenfalls auf die Tags. Dadurch wird von den selben Vorteilen profitiert, wie auch bei der Identifikation der Landschaften. Welche VMs innerhalb einer Landschaft Pflicht sind oder nicht, kann in direkter Nähe der Ressource und innerhalb von Azure sehr einfach eingesehen und um konfiguriert werden, ohne auf eine externe Datenquelle zugreifen zu müssen.

Hierbei bekommt die VM zusätzlich zu den Landschafts-Tags, welche sie zu einer Landschaft zuordnen, einen weiteren Tag. Dieser sieht folgendermaßen aus: Schlüssel=NameDerLandschaft, Wert=Pflicht. Somit ist diese VM im Kontext der Landschaft, als notwendige VM gekennzeichnet. Wenn nun nicht alle sondern nur die notwendigen VMs einer Landschaft hochgefahren werden sollen, können alle VMs dieser Landschaft nach diesem Tag durchsucht werden. Nur die VMs, welche diesen Tag besitzen, werden dann gestartet.

5 Schluss

Im Anhang befindet sich der Chatverlauf eines beispielhaften Gesprächs mit dem Chatbot, um den Verlauf der Konversation besser darstellen zu können.

5.1 Veröffentlichung in Kanälen

Auf den Chatbot soll jetzt auch zugegriffen werden können. Hierfür wurde innerhalb von Azure ein App Service und eine Web-App Bot erstellt, um den Chatbot dort zu veröffentlichen. Um nun auch tatsächlich mit dem Chatbot interagieren zu können, ohne den Bot Framework Emulator zu verwenden, der während der Entwicklung und zum Testen verwendet wurde, muss der Chatbot auch über mindestens einen Kanal verfügbar gemacht werden.

5.1.1 Möglichkeiten

Hierbei gibt es mehrere Möglichkeiten. Azure selbst, beziehungsweise die Web App, bietet eine Option: Im Web Chat testen. Hier kann der Bot, aus dem Azure Portal selbst, verwendet werden. Da jedoch genau dies verhindert werden soll, das der Benutzer das Azure Portal verwenden muss, soll ein weiterer externer Kanal hinzugefügt werden. Hierfür gäbe es viele Möglichkeiten, wie zum Beispiel:

- Microsoft Teams
- Cortona
- Skype
- Telegram
- Facebook
- Und viele mehr

5 Schluss

Wie anhand dieser Liste zu erahnen ist, bietet es sich an, einen Kanal zu wählen, über den bereits Kommunikation stattfindet. Hierbei chattet der Benutzer anstatt mit einer anderen Person, eben mit dem Chatbot.

5.1.2 Teams

Für diese Arbeit wurde sich für den Kanal Teams entschieden. Teams ist eine Plattform der Firma Microsoft, welche für Chats, Besprechungen, Notizen und vieles mehr genutzt werden kann.[22] Wie der Name schon sagt, ist Teams darauf ausgelegt, die Arbeit im Team, auch über größere Distanzen, zu unterstützen. Die Wahl sollte auf einen Kanal fallen, welcher bereits für die Kommunikation verwendet wird. Da dies bei Teams der Fall ist und Teams ebenfalls als Kanal für das Microsoft Chatbot Framework unterstützt wird, wurde sich für Teams entschieden.

Teams bietet auch die Möglichkeit, die eigenen Funktionalitäten zu erweitern. Über das App Studio von Microsoft Teams, können eigene Teams-Applikationen erstellt oder integriert werden. [12] Hier können auch Bots hinzugefügt werden, was für den Chatbot in dieser Arbeit getan wurde. Dafür wurde der in Azure gehostete Chatbot verknüpft. Es wurden mehrere Einstellungen, wie Name, Website, Beschreibung etc. hinzugefügt und dann konnte der Bot innerhalb von Teams installiert und getestet werden. Am Code des Bots mussten ebenfalls einige Änderungen gemacht werden. [19] [24] Jetzt kann der Bot aus Microsoft Teams aus verwendet und im Teams App Store eingereicht werden. Da die App in Teams lediglich auf den in Azure veröffentlichten Bot zugreift, wird auch immer die Version des Bots in Teams verwendet, welche die aktuellste Veröffentlichung in der Azure Web App ist.

5.2 Fazit

Das erwartete Ergebnis, ein Chatbot zum Starten und Beenden von virtuellen Maschinen der Plattform Microsoft Azure zu entwickeln wurde erfüllt. Mit dem Bot kann direkt oder über Microsoft Teams aus kommuniziert werden. Hierbei wurden Kenntnisse über das Microsoft Bot Framework inklusive dem Language Understanding Dienst LUIS und der Azure Plattform erlangt. Es wurde ein technisches Modell über virtuellen Maschinen und die Beziehungen zwischen diesen über die Tag-Funktionalität in Azure abgebildet. Dafür wurden VM-Landschaften eingeführt. Dieses Modell beinhaltet ebenfalls Informationen zur Notwendigkeit von einzelnen virtuellen Maschinen im Kontext einer Landschaft. Durch den Language Understanding Dienst und die Dialoge wurde das Problem gelöst, einen Bot natürliche Sprache verstehen zu lassen.

5 Schluss

Hierbei kann mit dem Bot kommuniziert werden und es können ihm Aufträge mitgegeben werden, ohne spezifische Kommandos verwenden zu müssen. Damit unterscheidet sich dieser von einem Command Line Interface, welches nur einfache Befehle entgegen nimmt.

Der Bot nimmt einem Team die aufwendige Aufgabe ab, jedem Team-Mitglied das Wissen zu übermitteln, wie er technische Prozesse innerhalb von Azure anstoßen kann. Der Anwender kann diese technischen Prozesse mithilfe des Bots starten, ohne die technische Bedienung beherrschen zu müssen. Er benötigt somit kein Wissen über Azure-spezifische Themen, genauso muss er sich nicht im Azure-Portal zurecht finden. Ein einfaches Starten einer VM funktioniert nach der erfolgreichen Authentifizierung am Bot über eine kurze und schnelle Konversation. Zusätzlich unterstützt der Bot den Benutzer beim Zurecht finden innerhalb der virtuellen Maschinen und Landschaften. Er bietet sogar ein Modell über VMs und Landschaften welches unabhängig vom Bot, über das Azure-Portal einsehbar und konfigurierbar ist. Durch den Bot fällt das erklären oder einarbeiten in das Azure Portal und in die spezifische VM-Umgebungen gänzlich weg. Für diese Anwendungsfälle muss dem Benutzer lediglich Zugang zum Bot erteilt werden, was über den Kanal Teams sehr einfach gestaltet wurde. Den Zugang zum Bot bereitzustellen bietet wirtschaftlich betrachtet einen großen Zeit/Aufwands-Vorteil, gegenüber dem Einarbeiten/Erklären in die Thematik.

5.3 Ausblick

Dieser Chatbot bringt viele Möglichkeiten mit, wie er in Zukunft erweitert werden könnte. Zum einen könnte die Auswahl der Ressourcen innerhalb von Azure erweitert werden. Anstatt nur virtuelle Maschinen könnte sich der Bot um weitere Ressourcen kümmern, bis zu allen Ressourcen, welche möglich sind. Des weiteren könnten auch die Operationen, welche der Bot auf die Ressourcen ausführen kann, erweitert werden. Es könnten mehr Infos über die Ressourcen abgefragt und das Organisieren der Landschaften ausgebaut werden.

Falls dies in einem Szenario notwendig werden sollte, könnte ein weiterer Bot implementiert werden, der alle Rechte in Azure hat, anstatt zu Impersonieren. Dies wäre in einem Szenario notwendig, wo Stakeholder kein Azure-Account mit Zugriff zu den Ressourcen haben können.

Eine weitere Möglichkeit den Bot zu erweitern, wäre die Speicherkonten durch eine Datenbank zu ersetzen. Hierbei wären viele Erweiterungen möglich. Beispielhaft könnte der Bot Statistiken mitschreiben und dem Benutzer somit Operationen vorschlagen, welche statistisch am häufigsten vorkommen.

Für alle Erweiterungsmöglichkeiten, ist es natürlich notwendig noch weitere Dialoge zu

5 Schluss

implementieren. Hierbei sind auch nahezu unendlich Erweiterungsmöglichkeiten vorstellbar, wie die Dialoge und LUIS gestaltet werden können.

Zu guter Letzt besteht noch die Möglichkeit, den Chatbot um weitere Channels zu erweitern. Im Kapitel, Veröffentlichung in Channels, ist beschrieben in welchen Channels der Bot bereits benutzt werden kann. Dies kann natürlich um sehr viel mehr weitere Channels erweitert werden. Je nach dem von wo aus der Bot verwendet werden soll.

6 Anhang

6.1 Link zum Repository

Repository: <https://github.com/SimonLay98/AzureBot>

6.2 Beispielkonversationen

6 Anhang

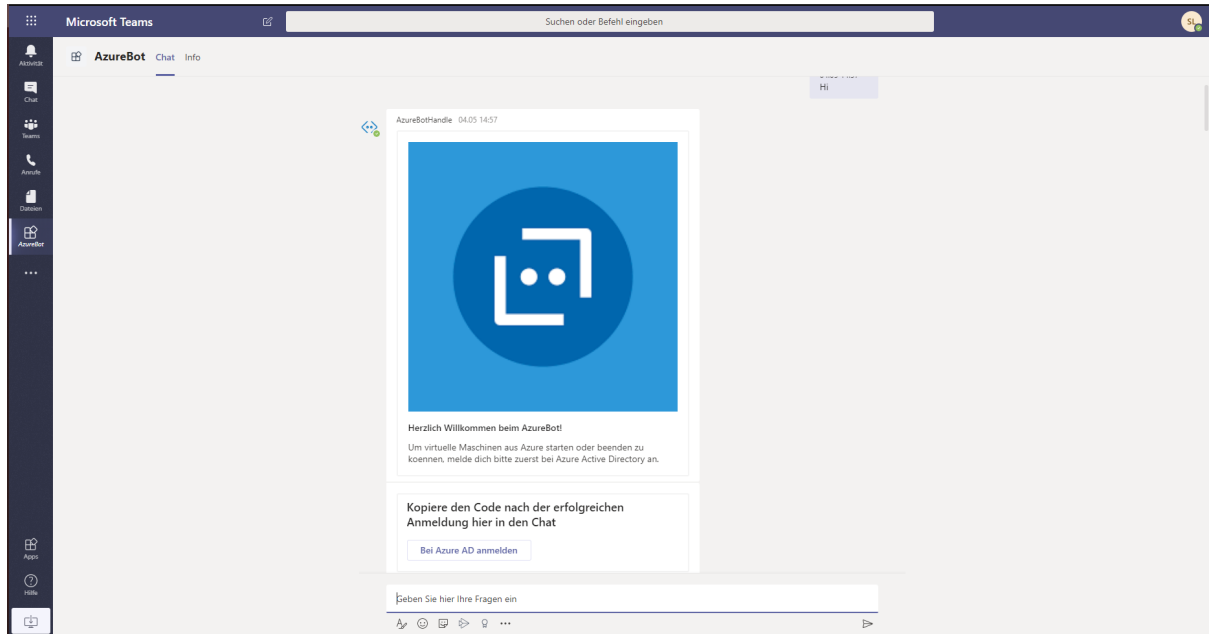


Abbildung 6.1: Teams-Integration: Begrüßung (Quelle: Screenshot)

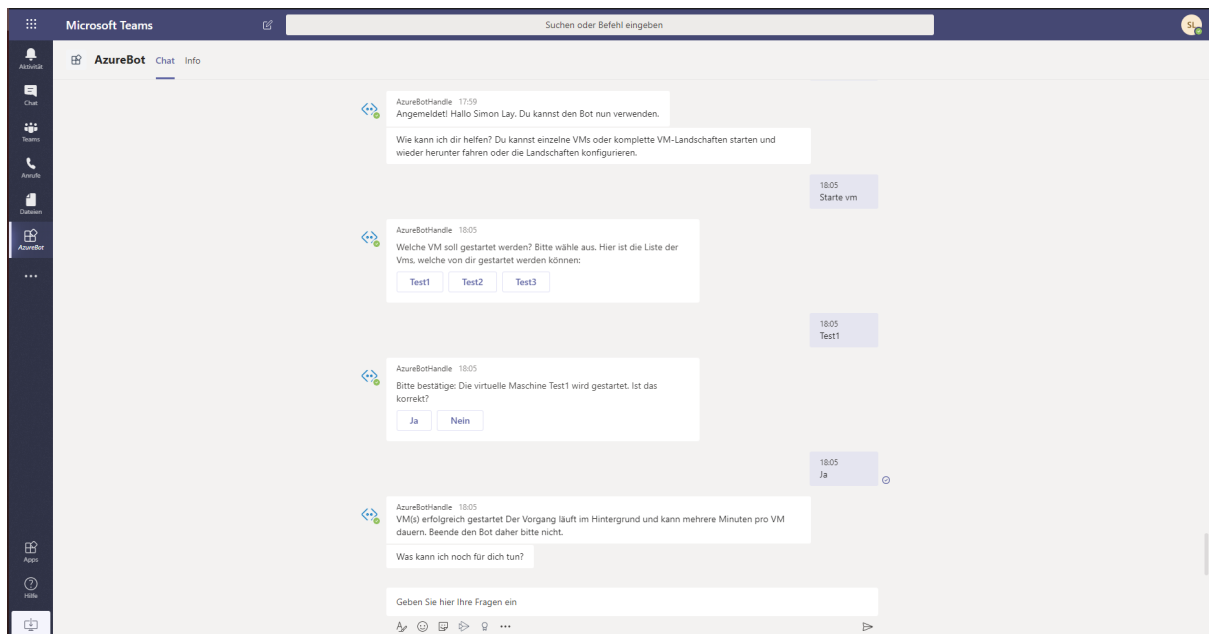


Abbildung 6.2: Teams-Integration: Starte VM (Quelle: Screenshot)

6 Anhang

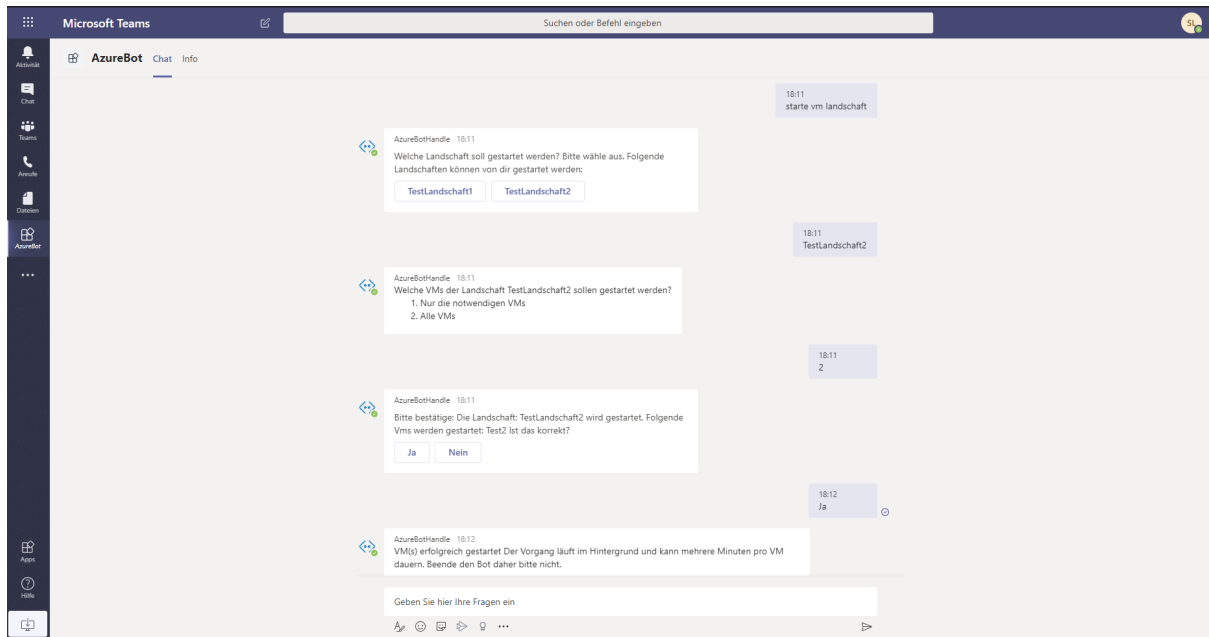


Abbildung 6.3: Teams-Integration: Starte VM-Landschaft (Quelle: Screenshot)

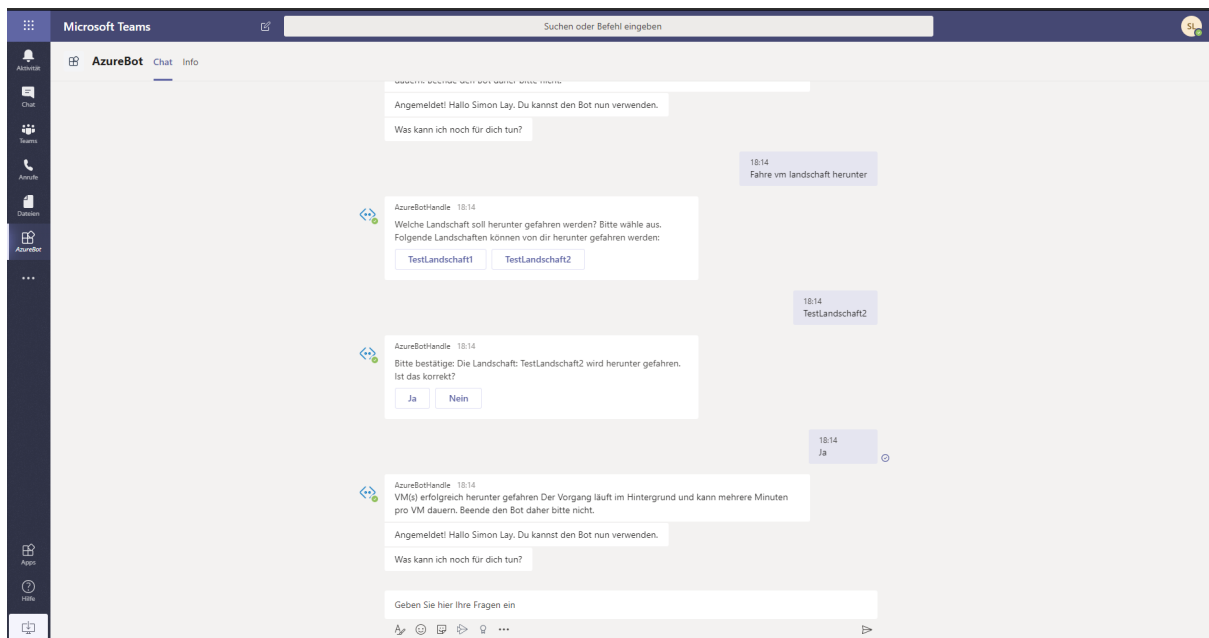


Abbildung 6.4: Teams-Integration: Beende VM-Landschaft (Quelle: Screenshot)

6 Anhang

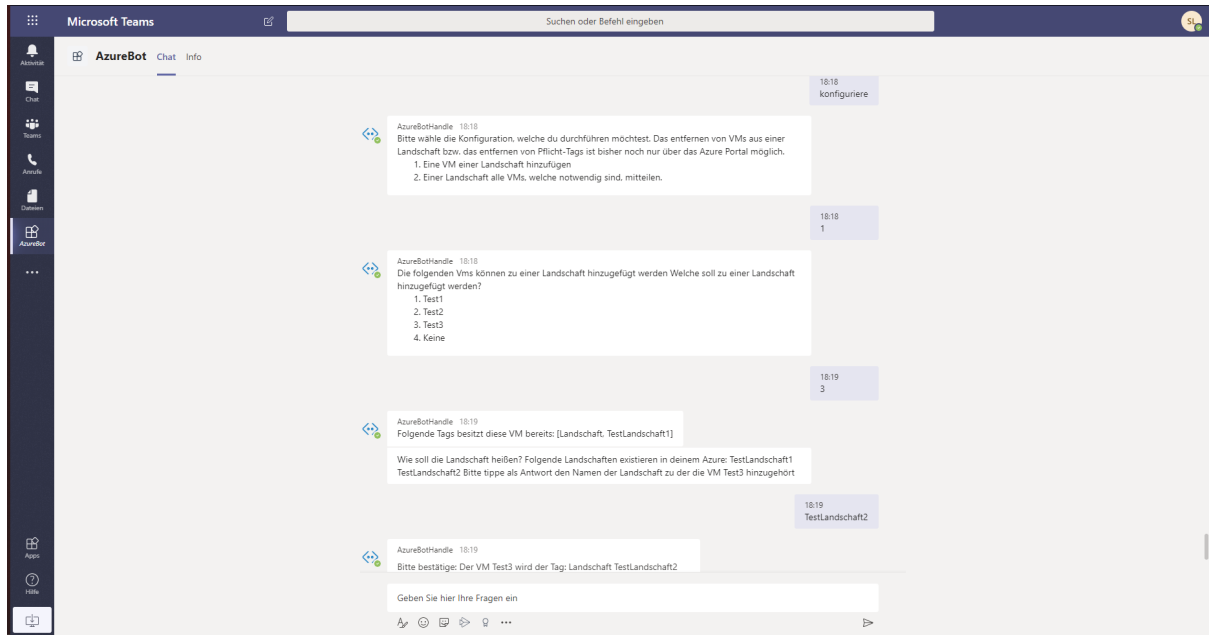


Abbildung 6.5: Teams-Integration: Konfiguriere: VM zu Landschaft hinzufügen 1 (Quelle: Screenshot)

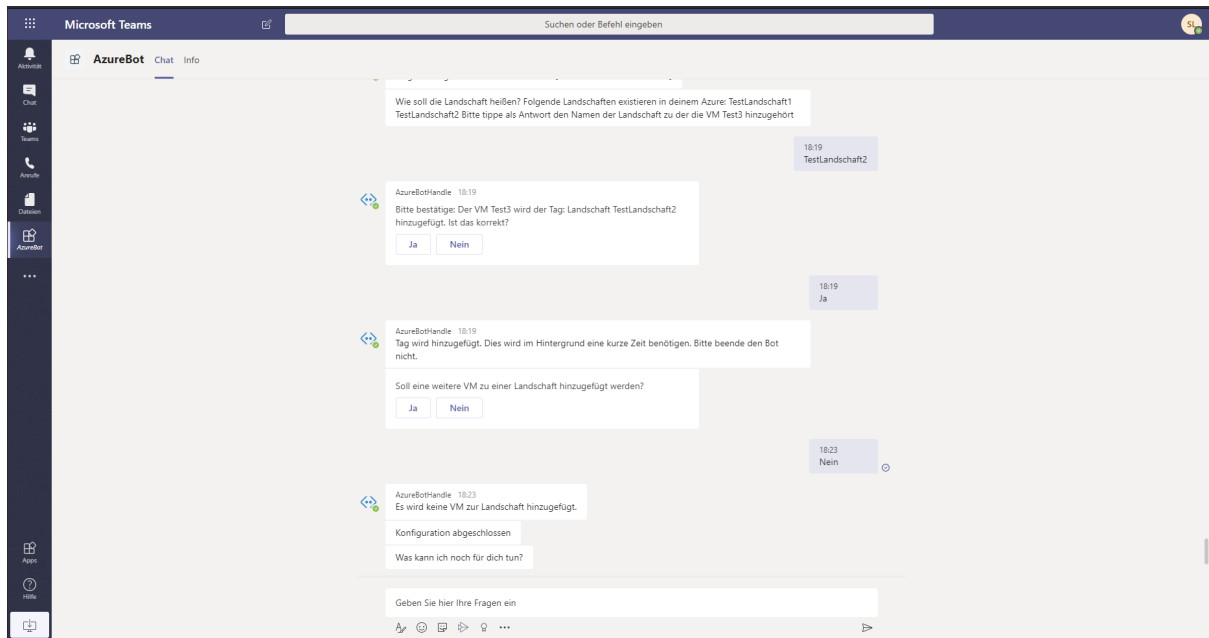


Abbildung 6.6: Teams-Integration: Konfiguriere: VM zu Landschaft hinzufügen 2 (Quelle: Screenshot)

6 Anhang

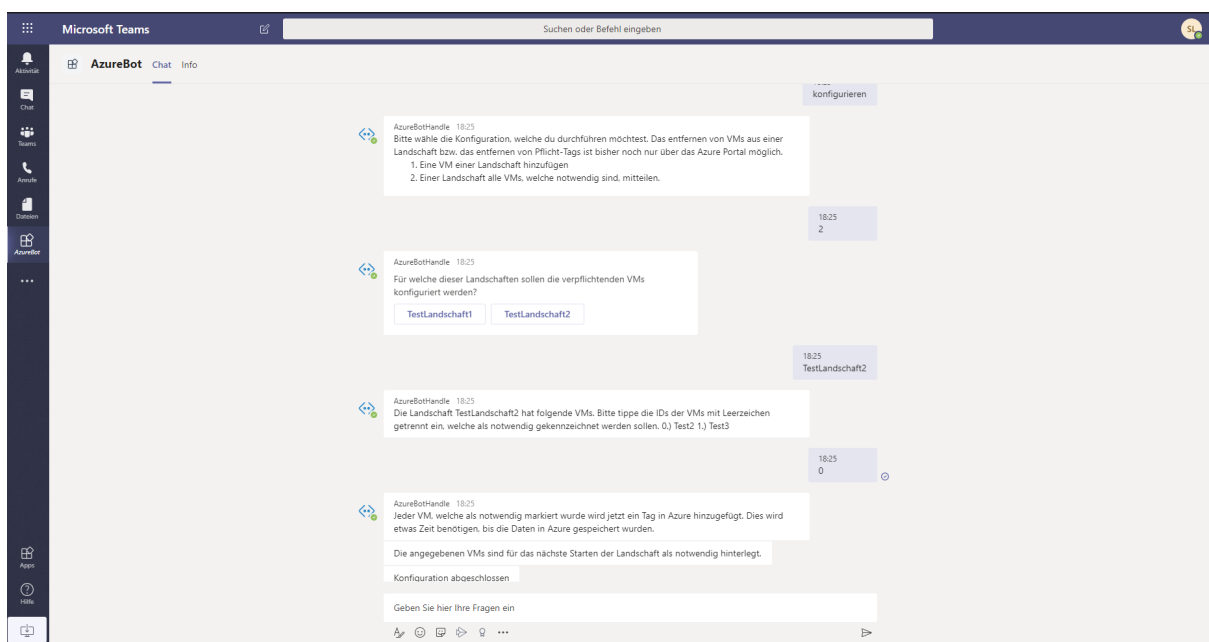


Abbildung 6.7: Teams-Integration: Konfiguriere: Notwenige VMs einer Landschaft (Quelle: Screenshot)

Literaturverzeichnis

- [1] : *How to Create .NET Bot (DIY #1)*. – URL <https://devcom.com/tech-blog/create-net-bot/>. – Zuletzt verwendet: 11.05.2020
- [2] : *How to Create .NET Bot (DIY #2)*. – URL <https://devcom.com/tech-blog/create-your-own-net-bot/>. – Zuletzt verwendet: 11.05.2020
- [3] : *LUIS (Language Understanding)*. – URL <https://www.luis.ai/home?force=1>. – Zuletzt verwendet: 11.05.2020
- [4] : *LUISGen Command Line Tool*. – URL <https://github.com/microsoft/botbuilder-tools/blob/master/packages/LUISGen/src/npm/readme.md>. – Zuletzt verwendet: 11.05.2020
- [5] : *Six ways to manage your Azure resources*. – URL <https://markheath.net/post/six-ways-manage-azure-resources>. – Zuletzt verwendet: 11.05.2020
- [6] : *Was ist der Unterschied zwischen NLP und NLU*. – URL <https://kauz.net/2016/12/30/was-ist-der-unterschied-zwischen-nlp-und-nlu/>. – Zuletzt verwendet: 11.05.2020
- [7] GITHUB: *Bot*. – URL <https://github.com/topics/bot>. – Zuletzt verwendet: 11.05.2020
- [8] INFORMATIONSTECHNIK, Bundesamt für Sicherheit in der: *Cloud Computing Grundlagen*. – URL https://www.bsi.bund.de/DE/Themen/DigitaleGesellschaft/CloudComputing/Grundlagen/Grundlagen_node.html. – Zuletzt verwendet: 11.05.2020
- [9] MACUN, Emre: *AZURE VM STOP VS. DEALLOCATED*. – URL <https://peakup.org/blog/azure-vm-stop-vs-deallocated/>. – Zuletzt verwendet: 11.05.2020

Literaturverzeichnis

- [10] MERINEAU, Etienne: *The 8 best chatbots of 2016*. – URL <https://venturebeat.com/2016/12/21/8-top-chatbots-of-2016/>. – Zuletzt verwendet: 11.05.2020
- [11] MICROSOFT: *Add authentication to a bot*. – URL <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-authentication?view=azure-bot-service-4.0&tabs=aadv1%2Ccsharp#register-the-azure-ad-oauth-application-with-the-bot>. – Zuletzt verwendet: 11.05.2020
- [12] MICROSOFT: *App Studio für Microsoft Teams*. – URL <https://docs.microsoft.com/de-de/microsoftteams/platform/concepts/build-and-test/app-studio-overview>. – Zuletzt verwendet: 11.05.2020
- [13] MICROSOFT: *Authenticate with the Azure Libraries for .NET*. – URL <https://docs.microsoft.com/en-us/dotnet/azure/sdk/authentication?view=azure-dotnet#mgmt-auth>. – Zuletzt verwendet: 11.05.2020
- [14] MICROSOFT: *Azure Storage-APIs für .NET*. – URL <https://docs.microsoft.com/de-de/dotnet/api/overview/azure/storage?view=azure-dotnet>. – Zuletzt verwendet: 11.05.2020
- [15] MICROSOFT: *BotBuilder-Samples*. – URL <https://github.com/microsoft/BotBuilder-Samples/tree/master/generators/dotnet-templates>. – Zuletzt verwendet: 11.05.2020
- [16] MICROSOFT: *CancellationToken Struktur*. – URL <https://docs.microsoft.com/de-de/dotnet/api/system.threading.cancellationtoken?view=netframework-4.8>. – Zuletzt verwendet: 11.05.2020
- [17] MICROSOFT: *Dialogbibliothek*. – URL <https://docs.microsoft.com/de-de/azure/bot-service/bot-builder-concept-dialog?view=azure-bot-service-4.0>. – Zuletzt verwendet: 11.05.2020
- [18] MICROSOFT: *Erstellen eines Azure-Speicherkontos*. – URL <https://docs.microsoft.com/de-de/azure/storage/common/storage-account-create?tabs=azure-portal>. – Zuletzt verwendet: 11.05.2020

Literaturverzeichnis

- [19] MICROSOFT: *Erstellen eines Bots für Microsoft Teams*. – URL <https://docs.microsoft.com/de-de/microsoftteams/platform/bots/how-to/create-a-bot-for-teams>. – Zuletzt verwendet: 11.05.2020
- [20] MICROSOFT: *Erstellen und Verwalten von virtuellen Windows-Computern in Azure mithilfe von C#*. – URL <https://docs.microsoft.com/de-de/azure/virtual-machines/windows/csharp>. – Zuletzt verwendet: 11.05.2020
- [21] MICROSOFT: *Lebenszyklus und Status virtueller Computer*. – URL <https://docs.microsoft.com/de-de/azure/virtual-machines/windows/states-lifecycle>. – Zuletzt verwendet: 11.05.2020
- [22] MICROSOFT: *Microsoft Teams*. – URL <https://www.microsoft.com/de-de/microsoft-365/microsoft-teams/group-chat-software>. – Zuletzt verwendet: 11.05.2020
- [23] MICROSOFT: *Steuern und Organisieren von Azure-Ressourcen mit Azure Resource Manager*. – URL <https://docs.microsoft.com/de-de/learn/modules/control-and-organize-with-azure-resource-manager/>. – Zuletzt verwendet: 11.05.2020
- [24] MICROSOFT: *unktionsweise von Microsoft Teams-Bots*. – URL <https://docs.microsoft.com/de-de/azure/bot-service/bot-builder-basics-teams?view=azure-bot-service-4.0&tabs=csharp>. – Zuletzt verwendet: 11.05.2020
- [25] MICROSOFT: *Übersicht über den Azure-Ressourcen-Manager*. – URL <https://docs.microsoft.com/de-de/azure/azure-resource-manager/management/overview>. – Zuletzt verwendet: 11.05.2020
- [26] ORF, Darren: *Google Assistant Is a Mega AI Bot That Wants To Be Absoutely Everywhere*. – URL <https://gizmodo.com/google-assistant-is-a-mega-chatbot-that-wants-to-be-abs-1777351140>. – Zuletzt verwendet: 11.05.2020