Alfred Zimmermann, Alexander Rossmann (Hrsg.): Digital Enterprise Computing 2015,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015 271

Automatisiertes Data Discovery innerhalb eines Provisionierungstools

Heiko Bonhorst[1], Patrick Kopf[2] und Fekkry Meawad[3]

Abstract: Das Provisioning Tool automaIT wurde prototypisch um die Möglichkeit eines Data Discovery erweitert, mit dem Ziel nicht durch automaIT verwaltete Systeme anbinden und steuern zu können. Daten aus dem Data Discovery werden mittels dem Tool Facter gesammelt und können dynamisch in ausführbare Modelle von automaIT integriert und ausgewertet werden. Dadurch kann der Verlauf weiterer Provisionierungsschritte gesteuert werden, ohne dass es eines manuellen Eingriffs bedarf.

Keywords: Provisioning, Data Discovery, automaIT, Facter, Ruby, Linux

1 Einleitung

Der Begriff Provisioning wird häufig mit unterschiedlichen Bedeutungen in der IT, jeweils abhängig vom Kontext, verwendet. Grundsätzlich bedeutet dies die Bereitstellung von Daten, Applikationen oder Berechtigungen für den Endanwender [JD06]. Im Rahmen dieses Papers bedeutet Provisioning das automatisierte Deployment eines Software Stacks auf einem bereits vorkonfigurierten Betriebssystem. Nach Abschluss des Vorganges steht dem Benutzer der angeforderte Service (Softwares Stack) vollumfänglich produktiv zur Verfügung.

Die Software automaIT der NovaTec GmbH wird hier als Provisioning Tool eingesetzt. Von automaIT verwaltete Server werden über einen vorher ausgerollten Agenten angesteuert. Die im automaIT Server enthaltenen Modelle wiederrum werden über den Agenten auf dem Zielsystem ausgeführt. Alle Ereignisse werden dabei in einer Datenbank protokolliert, womit zu jedem Zeitpunkt der aktuelle Zustand einer verwalteten Systemlandschaft zur Verfügung steht. Die Modellentwicklung wird durch eine speziell entwickelte XML-Struktur realisiert. Eine Modellausführung kann zusätzlich noch individuell parametrisiert werden, was es ermöglicht, ein Modell in unterschiedlichen Ausprägungen zu provisionieren [NT15]. So können etwa mehrere Datenbankinstanzen mit unterschiedlichen Ports und Instanznamen erstellt werden. Für das Data Discovery wurde auf die Entwicklung eines eigenen Algorithmus verzichtet, da sich hierfür das Tool Facter anbietet.

[1] Hermann Hollerith Zentrum Böblingen, heiko.bonhorst@student.reutlingen-university.de
[2] Hermann Hollerith Zentrum Böblingen, patrick.kopf@student.reutlingen-university.de
[3] Hermann Hollerith Zentrum Böblingen, fekkry.meawad@student.reutlingen-university.de

Facter ist ein OpenSource cross-platform Tool von Puppet Labs zum Auslesen von Informationen eines Systems. Das auf Ruby basierende Tool nutzt sogenannte Facts, um entsprechende Informationen wie Hardwareausstattung, Betriebssystem oder etwa Speicherplatz zu ermitteln. Facts beinhalten dabei Ausführungscode, womit die Informationen eines Systems gezielt abgegriffen werden können. Weiterhin kann Facter durch selbstgeschriebene Custom-Facts erweitert werden, womit es möglich ist eigene zusätzliche Logiken zu integrieren. Der Umfang dieses Papers nutzt sowohl die unter [PL15a] beschriebenen Core- als auch die unter [PL15b] beschriebenen Custom-Facts zur Ermittlung relevanter Informationen eines DBMS.

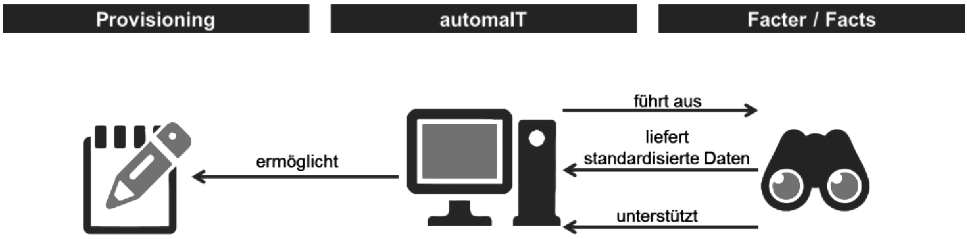

Abb. 2: Zusammenspiel von automaIT und Facter

2 Fragestellung

Durch die steigende Komplexität der heutigen Unternehmens IT steigen auch die Anforderungen an die Rechenzentren. Diese sollen auf der einen Seite schnell und leicht skalierbar sein und auf der anderen Seite flexibel. Weiterhin muss ein Großteil der Systeme innerhalb kürzester Zeit standardisiert bereitgestellt werden können. Dies wirft unter anderem die Frage nach einer Möglichkeit des Continous Delivery (CD) auf. CD beschreibt dabei eine Sammlung von Techniken, Prozessen und Werkzeugen, damit eine automatisierte Bereitstellung von Software in gleichbleibender Qualität geliefert werden kann.[4] Dies kann nur geschehen, solange IT-Umgebungen einer Systemlandschaft automatisiert an ein zentrales Verwaltungstool angebunden sind. Damit wird es möglich alle relevanten Informationen eines Systems und damit der Systemlandschaft gebündelt breitzustellen. Ziel des folgenden Use Case ist die Anbindung solcher Systeme an das Provisioningtool automaIT, welche aktuell dem Provisioningtool nicht bekannt sind, da sie beispielsweise manuell oder von einem anderen Tool bereitgestellt wurden. Dazu ist es notwendig, relevante Systeminformationen automatisiert zu ermitteln, und in automaIT zu überführen. Neben den rudimentären Informationen, wie beispielsweise der IP-Adresse, dem Festplattenspeicher und des Betriebssystems, sind vor allem auch Informationen über bereits installierte Softwarepakete und deren Konfigurationen

[4] Voraussetzung von CD ist dabei die Umsetzung von DevOps, welches Maßnahmen beschreibt, um die Kluft zwischen Entwicklung und Betrieb zu überwinden [GI13]. Im Rahmen dieses Papers wird nicht näher auf DevOps und CD eingegangen.

relevant. Dies soll mittels einer integrierten Data Discovery Funktion ermöglicht werden, um genau solche Systeme in einer IT-Landschaft zu integrieren und in den CD-Prozess mit aufnehmen zu können.

2.1 Use Case

Damit die Realisierbarkeit eines automatisierten Data Discovery bewertet werden kann, wird ein Proof of Concept anhand des im Folgenden beschriebenen Use Case durchgeführt.

Mittels automaIT sollen das PostgreSQL DBMS sowie die dazugehörigen Instanzen auf einer bis dahin für automaIT fremden IT-Umgebung automatisiert aufgesetzt werden. Damit Serverressourcen möglichst effizient eingesetzt werden, sollen mehrere Datenbankinstanzen auf einer Serverinstanz gestartet werden. Ob auf einem System genügend Ressourcen vorhanden sind, muss automaIT mittels Data Discovery eine Reihe von Informationen ermitteln.

Innerhalb den automaIT Modellen kann dazu ein entsprechender Workflow zur Überprüfung aufgebaut werden. Ziel ist, automaIT die Grunddaten von fremden Maschinen einer IT-Landschaft in einer standardisierten Form über das Data Discovery zur Verfügung zu stellen. Zu den Grunddaten zählen unter anderem die IP-Adresse, Rechnerarchitektur, Betriebssystem und Version, installierte Software Pakete und einer Vielzahl Use Case abhängiger Parameter:

- Installiertes PostgreSQL Paket
- Lokation des Datenverzeichnisses (Datadir)
- Belegter Port einer vorhanden PostgreSQL Instanz
- Nächster freier Port
- Freier Speicherplatz
- Verfügbarer Arbeitsspeicher

Diese Daten sollen automaIT so zur Verfügung gestellt werden, damit sogenannte automaIT Komponenten, diese als Standardvariablen nutzen können. Komponenten beinhalten dabei die Ausführungslogik der einzelnen Provisionierungsschritte. Zusammengehörige Komponenten werden zu einem Plugin zusammengefasst.

2.2 Nutzen

Der übergeordnete Nutzen dieser Funktionalität ist einerseits die Beschleunigung initialer Provisionierungsprozesse und anderseits die Identifikation fremder Maschinen innerhalb der vorhandener IT-Systemlandschaften. Zu den für automaIT fremden Systemen in einem Unternehmensnetzwerk zählen insbesondere bereits manuell

aufgesetzte Umgebungen. Diese Umgebungen sollen durch das Data Discovery möglichst automatisiert nach bereits vorhandener Software untersucht werden und die dabei ermittelten Ergebnisse in einem automaIT Modell abgebildet werden. Nach der automatisierten Abbildung im Modell soll es möglich sein, die Umgebungen, inklusive der dort bereits vorhanden Software, über automaIT zu managen ohne dafür manuelle Aktionen auf der bis dahin fremden Umgebung ausgeführt zu haben.

In der aufgeführten Umsetzung könnte es beispielsweise vorkommen, dass bereits eine PSQL DBMS auf der Umgebung ist, zu der aber noch keine Instanzen erstellt wurden. Nach einem automatisierten Data Discovery soll es dann möglich sein, Instanzen für das bereits verfügbare DBMS über automaIT zu erstellen und zu starten.

2.3 Optionen zur Umsetzung

Zur Umsetzung wurden verschiedene Optionen in Betracht gezogen. Zum einen wurde die Möglichkeit untersucht, eigene Skripte bzw. optionale Anwendungen für das Data Discovery zu programmieren. Zum anderen wurde verifiziert, ob bereits bestehende Tools und Werkzeuge auf dem Markt existieren, die den definierten Vorgaben entsprechen. Aufgrund vorgegebener Einschränkungen wird die Funktionalität des Data Discovery jedoch nicht Teil einer Integration in das Produkt sein. Aus diesem Grund ist die Wahl auf ein bereits existierendes Tool zum Data Discovery gefallen. Das Tool Facter der Firma Puppet Labs ist unter der Opensource Lizenz veröffentlicht und plattformunabhängig. Neben der sehr großen Community, die zur Weiterentwicklung beiträgt, werden bereits entsprechende Skripte, sogenannte Core Facts zum Auslesen einiger Standardparameter wie der IP-Adresse, Betriebssystemversion sowie verfügbarer Arbeitsspeicher mitgeliefert [PL15a]. Durch das offene System lässt sich Facter flexibel erweitern und erlaubt es eigene Custom Facts, wie unter [Kr13] beschrieben, zu definieren. Diese Facts können in automaIT eingebunden werden, womit eine Erweiterung durch beliebige Facts und damit Use Case spezifische Informationen möglich ist. Im folgenden Teil dieser Arbeit wird die Frage beantwortet, wie der untersuchte Use Case in der Praxis umgesetzt werden kann.

3 Ergebnis

Ziel der Arbeit ist automaIT mit einer Komponente zum automatisierten Data Discovery zu erweitern. Damit dies erreicht werden kann, wurde im ersten Schritt die im Folgenden beschriebene Lösungsarchitektur entwickelt. Die umgesetzte Architektur basiert im Wesentlichen auf zwei in automaIT abgebildeten Plugins.

3.1 Linux System Service Plugin

Das Linux System Service Plugin wird direkt nach der initialen Verknüpfung von automaIT und dem zu verwaltenden Server ausgeführt.[5] Dazu wurde ein sogenannter Data Discovery Setup Service (kurz DDSS)[6] erstellt. Gestartet wird dieser über das automaIT Web Front-End. Nach dem Start führt dieser die im Plugin abgebildeten und beschriebenen Prozessschritte auf der Ziel-Umgebung aus:

1. Da Puppet, und somit auch Facter, eine in Ruby geschriebene Library ist, wird die Verfügbarkeit von Ruby geprüft und bei Bedarf auf der Umgebung eine Installation ausgeführt.
2. Zur Ausführung der in Ruby geschriebenen Facts ist Facter Voraussetzung. Dazu wird bei Nichtverfügbarkeit Facter installiert.
3. Damit die für den Use Case definierten Custom-Facts ausgeführt werden können ist es im ersten Schritt notwendig, diese auf die Umgebung zu übertragen. Im zweiten Schritt wird zusätzlich der Pfad zur Facter Library definiert und gesourced, damit bei der Ausführung der Facts nicht der absolute Pfad angeben werden muss. Somit reicht die Ausführung von *facter <Name des Facts>*, um ein Fact auszuführen und das Ergebnis zu erhalten.
4. Veranschaulichung des globalen Data Discovery: Im Gegensatz zu der im zweiten Plugin beschriebenen Umsetzung des PostgreSQL Use Case, bei dem Facts erst zum Zeitpunkt des Bedarfs ausgeführt werden, wird hier die alternative Möglichkeit eines globalen Data Discovery dargestellt. Dazu werden alle auf der Umgebung verfügbaren Facts (d.h. sowohl Core [PL15a] als auch Custom [PL15b]) zu Beginn ausgeführt und in einer temporären Datei (discovery.log) abgespeichert. Aus dieser Datei können die ermittelten Werte zu einem beliebigen späteren Zeitpunkt ausgelesen werden, ohne die Ergebnisse über eine erneute Ausführung der Facts zu ermitteln.

3.2 PostgreSQL Plugin

Während das erste Plugin vorbereitende Aufgaben sowie das prototypische, globale Data Discovery übernimmt, ist das zweite Plugin für das PostgreSQL spezifische Data Discovery zuständig. Zur Abbildung des erwähnten PostgreSQL Use Case beinhaltet das Plugin drei Komponenten:

- Server: Prüfung ob und wenn ja, in welcher Version bereits ein PostgreSQL Server vorhanden ist. Basierend auf diesem Ergebnis wird der weitere Ablauf dynamisch angepasst. Das bedeutet: Sollte bereits ein Server in passender Konfiguration

[5] Voraussetzung hierfür ist, dass ein automaIT Agent bereits auf der Umgebung installiert wurde. Die dazu notwendige Logik ist bereits Teil der automaIT Software und für die weitere Beschreibung nicht relevant, weshalb hierauf nicht weiter eingegangen wird.

[6] DDSS: Besteht aus einem System Service, der Hostinitialisierung sowie dem globalen Discovery.

verfügbar sein, wird die bestehende Installation lediglich im automaIT Modell hinterlegt ohne Änderungen an der Umgebung vorzunehmen. Sollte kein oder ein nicht passend konfigurierter Server installiert sein, führt automaIT eine Installation mit ggfs. vorgelagerter Deinstallation[7] durch, und hinterlegt das Ergebnis standardmäßig im automaIT Modell.[8]

- Instanz: Installation einer beliebigen Anzahl von PostgreSQL Instanzen, die über verschiedene Ports erreichbar sind. Hierbei wird, ähnlich zum Vorgehen in der Server-Komponente, dynamisch entschieden, auf welchem Port eine neue Instanz zur Verfügung gestellt werden kann. Dies bedeutet: Wenn der Port bereits anderweitig belegt ist, wird per Fact automatisch ein anderer freier Port ermittelt.
- Datenbank: Aufsetzen der PostgreSQL Datenbank auf einer der bereits verfügbaren Instanzen. Durch ein Fact wird überprüft, ob ausreichend Speicherplatz auf einem spezifischen Logical Volume verfügbar ist. Das Ergebnis wird zur Laufzeit der Komponente ausgewertet und beeinflusst den weiteren Ablauf.

3.3 Architektur

Die zuvor beschriebene Architektur ist im folgenden Schaubild dargestellt:

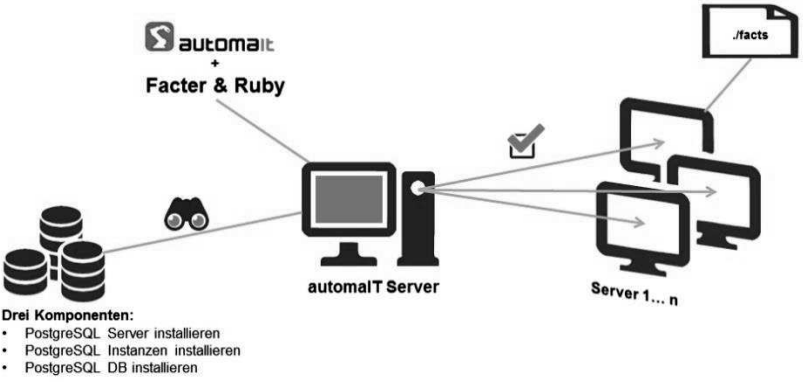

Abb. 3: Aufbau der Lösungsarchitektur

Links oben ist das Linux System Service Plugin dargestellt. Links unten ist das PostgreSQL Plugin mit den drei Komponenten zur Abbildung des PostgreSQL Use Case abgebildet.

[7] Für diesen Use Case war die vorgelagerte Deinstallation eine explizite Anforderung. Für die Abbildung anderer Use Cases kann dieses Vorgehen jedoch durchaus nicht gewollt sein und somit weggelassen werden.
[8] Die Abbildung im automaIT Modell ermöglicht es nachgelagerten Aktionen die dort hinterlegten Informationen zu nutzen.

3.4 Facts

Damit der Ablauf der Provisionierung automatisiert gesteuert werden kann, nutzen die einzelnen Komponenten verschiedene Core- und Custom-Facts, die zur Laufzeit ausgeführt werden und als Ergebnis ein Key-Value Paar zurückliefern. Zur Ausführung der Facts ist es notwendig, den Ausführungsbefehl in der automaIT Komponente anzugeben. Wird die Komponente nun ausgeführt, so wird zur Laufzeit über den automaIT Agent der hinterlegte Befehl auf der Ziel-Umgebung ausgeführt, und das Ergebnis an automaIT zurückgeliefert. Auf Basis des Ergebnisses kann dann das weitere Vorgehen abgeleitet werden.

Um dieses Vorgehen zu veranschaulichen, wird die PostgreSQL Instanz Komponente genauer betrachtet. Die Komponente ermittelt, ausgehend vom Default PostgreSQL Port (5432), per Ausführung des Custom-Facts *psql_nextFreePort* dynamisch den nächsten freien Port. Ist zu dieser Zeit bereits eine nicht über automaIT aufgesetzte Instanz auf der Umgebung verfügbar, kann automaIT trotzdem die gewünschte Provisionierung durch dynamische Anpassung der Parameter (in diesem Fall des Ports) fortführen. Alternativ ist es möglich, die über das Data Discovery ermittelten Werte im Modell zu hinterlegen, womit bei einem erneuten Durchlauf kein erneutes Data Discovery mehr notwendig ist.[9] Weiterhin wäre es über dieses Verfahren möglich, bereits vorhandene, nicht über automaIT erzeugt Instanzen, nachträglich im automaIT Modell abzubilden.

Der Aufbau dieses Facts ist zur Veranschaulichung im Folgenden dargestellt. Zu erkennen ist, dass vom Default PostgreSQL Port gestartet wird. Anschließend wird geprüft, ob der Port bereits anderweitig belegt ist. Falls ja, wird der Port iterativ um 1 erhöht und solange erneut geprüft, bis der nächste freie Port ermittelt werden konnte. Das Ergebnis des ausgeführten Facts wird wie folgt dargestellt: *psql_nextFreePort => 5433*.

```
#psql_nextFreePort
defaultPort = "5432"
freeportfound = false
while not freeportfound
results = Facter::Util::Resolution.exec(
        "netstat -lan | grep tcp | grep " << defaultPort ).split("\n")
  if results.empty?
    Facter.add("psql_nextFreePort".to_sym) do
      setcode { defaultPort }
    end
    freeportfound = true
    break
  else
    defaultPort = (defaultPort.to_i + 1).to_s
  end
end
```

Abb. 4: psql_nextFreePort Fact

[9] Wie bereits erwähnt, wäre es alternativ auch möglich gewesen die Informationen bereits im globalen Data Discovery zu ermittelt und diese im Modell zu hinterlegen.

Ausgeführt wird das Fact über den Befehl *facter psql_nextFreePort*.[10]

Auf PostgreSQL-Seite wurden zu dem oben erläuterten Fact *psql_nextFreePort* weitere Facts definiert und in den Komponenten eingesetzt:

- *psql_version*: zur Ermittlung der Version bzw. Rückmeldung, dass kein PostgreSQL installiert ist.
- *psql_instances*: Zur Ermittlung der Ports aller aktuell verfügbaren Instanzen.
- *psql_datadir*: Zur Ermittlung des Pfades des PostgreSQL Data Directory.

Weitere für den Use Case relevante Custom Facts sind:

- *lvs_#{entry}_size*: um die absolute Größe der Logical Volumes auszulesen.
- *lvs_#{entry}_free:* um den aktuell verfügbaren Platz zu ermitteln.

Der Teil *#{entry}* wird dabei jeweils dynamisch durch die Namen der verfügbaren Logical Volumes ersetzt.

Als einziges Core Fact wurde *operatingsystem* eingesetzt, womit systemspezifische Aktionen in den Komponenten ausgeführt werden können.

4 Fazit und Ausblick

Durch die prototypische Implementierung eines Data Discovery konnte gezeigt werden, dass für automaIT unbekannte Maschinen einer IT Infrastruktur optimal eingebunden und ein hoher Grad an Automatisierung geschaffen werden konnte. Dadurch ist es möglich, das Provisioning Tool automaIT auch in nicht gemanagte Infrastrukturen effizient und optimal einzuführen. Die Erweiterung durch Facter unterstützt dabei in hohem Maße die Flexibilität mit dem Ziel die Mächtigkeit des Data Discovery weiter auszubauen. Weiterhin ist nach dem Data Discovery ein Abbild der vorliegenden IT-Topologie im automaIT Modell hinterlegt, welches unter Umständen vorher nicht bekannt war.

Bevor dies jedoch erreicht werden kann, bedarf es eines erhöhten Entwicklungsaufwandes, um weitere vollumfänglichere Custom-Facts zu erzeugen, als auch bestehende automaIT Komponenten zu erweitern.

Damit eine bessere Integration des Data Discovery Mechanismus erreicht werden kann, wäre eine Integration in die automaIT interne Substitutionssprache zu empfehlen. Hierdurch könnten Facts innerhalb des XML-Code der Komponenten durch den Einsatz von Systemvariablen (z.B. *:[sys.fact:getUserId]*) genutzt und ad-hoc verwendet werden.

[10] Wird lediglich *facter* ausgeführt, so liefert Facter alle auf der Umgebung verfügbaren Key-Value Paare (sowohl Core- als auch Custom-Facts).

Neben der Vereinfachung, systemspezifische Daten abgreifen zu können, würde dadurch auch der Codeumfang reduziert werden.

Nachfolgend wird ein Beispiel aufgeführt, wie sich die Integration als Systemvariablen im Code widerspiegeln könnte. In Abbildung 4 ist der automaIT Code zur Ermittlung der UID eines Users aus einem angebundenen LDAP dargestellt. In Abbildung 5 wird ein beispielhafter Codeausschnitt mit möglicher integrierter Facter Systemvariable abgebildet.

```
<varList>
        <var name="uid" default="" />
        <var name="ldapAdminServer" default=":[target(:[ldapAdminHost]):ldapAdminServer]" />
        <var name="ldapUserCreateCmd" default=":[target(:[ldapAdminHost]):ldapUserCreateCmd]" />
      </varList>
      <try>
        <block>
          <execNative userToRunAs=":[execUser]">
            <assignOutput varName="uid" />
            <exec cmd=":[localScriptSystemPath]/user_getuid.sh">
              <arg value="--username" />
              <arg value=":[userName]" />
              <arg value="--ldapserver" />
              <arg value=":[ldapAdminServer]" />
            </exec>
          </execNative>
          <return value=":[uid]" />
        </block>
        <catch>
          <raise message="Failed to get :[userName] UID, maybe non exisiting user!" />
        </catch>
```

Abb. 5: UID-Ermittlung eines Users ohne Systemvariable

```
<varList>
        <var name="uid" default=":[sys.fact:getActUidLdap]" />
</varList>
      <try>
        <block>
          <return value=":[uid]" />
        </block>
        <catch>
          <raise message="Failed to get :[userName] UID, maybe non exisiting user!" />
        </catch>
      </try>
```

Abb. 6: UID-Ermittlung mit beispielhafter Systemvariable

Durch die automatische Anbindung und Integration unbekannter Systeme einer IT-Landschaft in ein zentrales Verwaltungstool kann im hohen Maße Continous Delivery unterstützt und betrieben werden. Dadurch ist es in etwa möglich, zentrale Systeme, die im Laufe einer IT-Landschaft gewachsen sind und nicht automatisiert verwaltet wurden, aufzunehmen und in aktuelle automatisierte Prozesse mit einzubinden. Weiterhin wird

dadurch ein hoher Grad der Standardisierung für IT-Systeme erreicht, was zur Folge hat, dass diese in gleichbleibender Qualität erzeugt und genutzt werden können. Historisch gewachsene IT-Landschaften lassen sich durch das Data Discovery kosten- und zeiteffizient einbinden. Zusätzlich kann dadurch die aktuelle IT-Topologie aufgezeichnet werden.

Literaturverzeichnis

[NT15] NovaTec GmbH, automaIT, http://automait.de, Stand: 20.03.2015.

[JD06] Jones, D.; Desai, A.: The Reference Guide To Data Center Automation. Realtimepublishers.com, 2006.

[PL15a] Puppet Labs, Core Facts, docs.puppetlabs.com/facter/latest/core_facts.html, Stand: 08.04.2015.

[PL15b] Puppet Labs, Fact Overview, docs.puppetlabs.com/facter/latest/fact_overview.html, Stand: 08.04.2015.

[Kr13] Krum, S.; Van Hevelingen, W.; Kero, B.; Turnbull, J.; McCune, J. Pro Puppet. 2. Auflage, Springer, Berlin, 2013.

[GI13] Gartner Inc. 2013, Gartner IT Glossary - DevOps, www.gartner.com/it-glossary/devops, Stand: 16.05.2015.