Systemarchitektur eines verteilen, ereignisgetriebenen Konfigurationsmanagement-Frameworks auf Basis von Open-Source-Komponenten

Martin Helmich

Hochschule Osnabrück
martin.helmich@hs-osnabrueck.de

Mittwald CM Service GmbH & Co. KG
m.helmich@mittwald.de

Abstract: Moderne Open-Source-Konfigurationsmanagementsysteme wie CFEngine, Puppet oder Chef arbeiten als zentralisierte Systeme, in denen der Soll-Zustand eines IT-Systems in deklarativer Art beschrieben wird. Diese Definitionen werden von auf den Zielsystemen laufenden Agenten abgerufen und mit dem Ist-Zustand abgeglichen. Dieser Ansatz ist nur bedingt geeignet für Systeme mit extrem häufigen Konfigurationsänderungen, die zudem unter Einhaltung bestimmter Zeitbedingungen umgesetzt werden müssen. Darüber hinaus ist es in vielen Lösungen nicht möglich, hostübergreifende Abhängigkeiten von Systemzuständen zu definieren.

Dieser Artikel beschreibt eine mögliche Architektur für ein Softwaresystem, das als Teilkomponente eines komplexen Konfigurationsmanagementsystems genutzt werden kann. Insbesondere wird darauf eingegangen, wie aus frei verfügbaren Open-Source-Komponenten eine hochskalierbare Message-Broker-basierte Architektur aufgebaut werden kann, die in der Lage ist, beliebig programmierbare Konfigurationsänderungen mit sehr geringer Latenz auf einer großen Anzahl heterogen konfigurierter Hosts durchzuführen.

Im Unterschied zu den oben genannten Lösungen arbeitet das hier vorgestellte System nicht mit einer deklarativen Beschreibung des Soll-Zustandes, sondern nimmt Aufträge entgegen, die in Form imperativer Skripte auf den Zielsystemen ausgeführt werden. Ferner ist es imstande, Abhängigkeiten einzelner Konfigurationsänderungen untereinander zu berücksichtigen und somit auch komplexe, aufeinander aufbauende, hostübergreifende Folgen von Aufträgen zu automatisieren.

1 Einleitung und Motivation

Zur effizienten Administration größerer IT-Systeme werden heutzutage Konfigurationsmanagementlösungen wie CFEngine [CFE14a], Chef [Che14] oder Puppet [Pup14b] eingesetzt [DJV10]. In bestimmten Fällen stoßen diese jedoch an ihre Grenzen; sei es in der Anzahl der zu verwaltenden Systeme, der Häufigkeit, mit der Änderungen an verwaltete Systeme propagiert werden müssen, oder hinsichtlich der Zugänglichkeit durch den Endanwender.

Ein spezieller Anwendungsfall besteht bei Hosting- oder Cloud-Dienstleistern, die bestimmte Bereiche der von ihnen zur Verfügung gestellten Infrastruktur ihren Endkunden selbst zur freien Konfiguration überlassen möchten. Besondere Herausforderungen in solchen Systemen bestehen darin, Konfigurationsänderungen schnell umzusetzen und ein besonders hohes Maß an Stabilität sicherzustellen. Aufgrund der Vielzahl unprivilegierter Nutzer muss zudem ein hohes Maß an Sicherheit garantiert werden. Nicht zuletzt muss dem Nutzer eine gut bedienbare Nutzerschnittstelle zur Verfügung gestellt werden.

Ein konkreter Fall eines solchen Systems ist das Serverautomatisierungssystem des Webhosting-Unternehmens MITTWALD CM SERVICE [Mit14]. Dieses betreibt eine Infrastruktur aus derzeit ca. 8.000 Servern (Tendenz steigend), von denen einige dediziert einzelnen Kunden zur Verfügung stehen und andere von mehreren Kunden gemeinsam genutzt werden (*shared hosting*). Das System bietet dem Kunden eine grafische Oberfläche an, über die gebuchte Webhosting-Instanzen konfiguriert werden können (z.B. Domains, DNS, E-Mail).

Ziel dieses Artikels ist es, eine auf Open-Source-Komponenten basierende Architektur für eine Teilkomponente eines Konfigurationsmanagementsystems vorzustellen, die eine effiziente Ausführung von beliebig programmierbaren Aufgaben auf einzelnen Hosts oder Host-Gruppen eines großen Netzwerks ermöglicht und gleichzeitig die Skalierbarkeit des Gesamtsystems sicherstellt. Insbesondere wird dabei auf die Implementierung einer Publish-Subscribe-Architektur [TvS07] mittels des Advanced Message Queuing Protocol (AMQP) [OAS12] eingegangen. Diese Architektur soll ein ähnliches System ablösen, welches im Unternehmen bereits im Einsatz ist. Dieses bearbeitet derzeit bereits mehrere 10.000 Konfigurationsänderungen pro Tag, wird jedoch in mittelfristiger Zukunft voraussichtlich an die Grenzen seiner Skalierbarkeit stoßen.

2 Stand der Forschung

Die Grundlage moderner Konfigurationsmanagementsysteme bildet [Bur95], in welchem das Werkzeug CFEngine erstmalig vorgestellt wurde. Eine formalere Beschreibung der theoretischen Grundlagen solcher deklarativen Konfigurationsmanagementwerkzeuge erfolgt in [CHIK03] und [BC06]. In [CHIK03] wird insbesondere der Begriff der Konvergenz definiert. Dieser Begriff beschreibt die Eigenschaft eines Systems, stets einen zentral definierten Soll-Zustand anzustreben. In [DJV10] werden aktuelle Werkzeuge (sowohl als Open-Source-Software (OSS) als auch kommerziell) anhand verschiedener Kriterien untersucht. Als relevante OSS-Werkzeuge werden dabei Chef [Che14], Puppet [Pup14b] und das bereits genannte CFEngine [CFE14a] identifziert.

Neuere Ansätze hinsichtlich des Konfigurationsmanagements sind Salt [Sal14] und Ansible [Ans14]. Die Besonderheit von Salt ist der Nachrichtenaustausch mithilfe von Message Queues gegenüber den von den übrigen Lösungen genutzten REST-APIs [ZGW+13]. Die in diesem Artikel vorgestellte Architektur arbeitet ebenfalls mit Message Queues, setzt jedoch mit einem AMQP-Broker auf eine dedizierte Middleware, während Salt mit ZeroMQ auf eine eingebettete Queue setzt [Hos13].

In [VJ13] wird ein Ansatz vorgestellt, mit welchem auch komplexere Konfigurationsaufgaben automatisiert werden können. In Ergänzung zu dem üblichen deklarativen Ansatz erlaubt [VJ13] auch imperative Programmierung der System-Konfigurationen und kann Systemzustände hostübergreifend modellieren.

Im Bereich der Message-Broker hat sich in den letzten Jahren das Advanced Message Queueing Protocol (AMQP) als Standard etabliert. AMQP wurde erstmals vorgestellt in [Vin06] und später endgültig standardisiert in [OAS12]. Eine Beschreibung des AMQP zugrunde liegenden Modells liefert [KD14] (vgl. auch Abb. 1). [YZF+11] untersuchen den Aufbau skalierbarer Architekturen in Cloud-Computing-Umgebungen unter Einsatz von AMQP-Queues, hier insbesondere RabbitMQ [Piv14b]. Zielsetzung ist hier jedoch primär die effiziente Ausführung rechenintensiver Aufgaben. In [FLRU13] wird die Leistungsfähigkeit von AMQP-basierten Services (auch hier insbesondere RabbitMQ) mit RESTful Webservices verglichen.

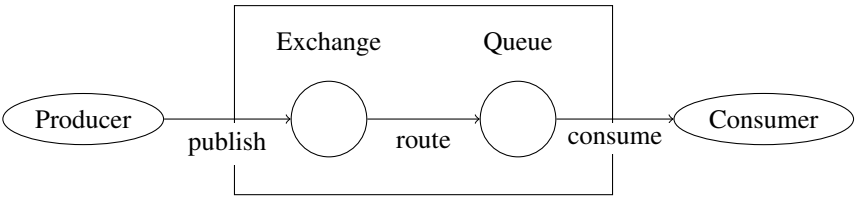

Abbildung 1: Das AMQP zugrunde liegende Modell nach [KD14].

3 Anforderungen an das System

3.1 Funktionale Anforderungen

Im Unterschied zu ganzheitlichen Lösungen wie [Che14], [Pup14b] oder [CFE14a] liegt der Fokus der hier vorgestellten Architektur ähnlich wie bei [Pup14a] ausschließlich auf der effizienten Ausführung von Konfigurationsaufträgen auf den Hosts eines Netzwerks. Diese Lösung ist dazu vorgesehen, in ein Gesamtsystem integriert zu werden und soll entsprechende Schnittstellen für Drittapplikationen anbieten, die ihrerseits den Sollzustand des Systems modellieren und entsprechende Änderungen an die Zielsysteme propagieren.

Primär muss das System eine Programmierschnittstelle anbieten (beispielsweise per SOAP [GHM+07] oder RESTful-HTTP [Fie00]), über die Drittanwendungen Aufträge an beliebige Hosts verteilen können. Diese werden dann in Form vordefinierter und parametrisiert aufgerufener Skripte auf den Hosts ausgeführt, um einen bestimmten Konfigurationszustand herzustellen.

Die auf den Zielsystemen auszuführenden Skripte sollen versioniert in einem zentralen Repository verwaltet werden können. Dabei muss eine heterogene Systemzusammensetzung berücksichtigt werden; je nach Konfiguration des Hosts (Betriebssystem, Distribution, Version, ...) sind zur Bearbeitung desselben Auftrags möglicherweise unterschiedliche Skripte notwendig.

3.2 Qualitative Anforderungen

Eine zentrale Anforderung liegt in der Skalierbarkeit des Systems. So stellen [DJV10] fest, dass zahlreiche der OSS-Werkzeuge nur in Netzwerken bis 10.000 Hosts skalieren. CFEngine ist nach Herstellerangaben [CFE14b] in Standardkonfigurationen für ca. 2.000 Hosts pro Management-Server geeignet; zur weiteren Skalierung sind sehr spezielle Anpassungen erforderlich. Eine weitere Einschränkung ergibt sich hinsichtlich der Anzahl der pro Host konfigurierbaren Ressourcen [CFE14b]. In der vorliegenden Systemarchitektur ergibt es sich, dass auf einzelnen Hosts teils sehr viele Ressourcen verwaltet werden müssen (etwa mehrere 100.000 Firewall-Regeln oder mehrere Millionen DNS-Einträge). Zudem bestehen strenge Anforderungen hinsichtlich der Zeit, in der Aufträge an die Hosts zugestellt und dort bearbeitet werden müssen. Die Zustell- und Bearbeitungszeiten sollen mit zunehmender Skalierung des Systems konstant bleiben.

Da das System in einem Netzwerk aus heterogen konfigurierten Hosts eingesetzt werden soll, besteht ebenfalls die Anforderung der Portabilität. Die Systeme umfassen diverse Linux-Distributionen wie etwa Debian, Suse, CentOS und Embedded-Varianten (z.B. auf Netzwerkhardware); auch die Nutzung von Windows-Hosts ist zukünftig nicht ausgeschlossen.

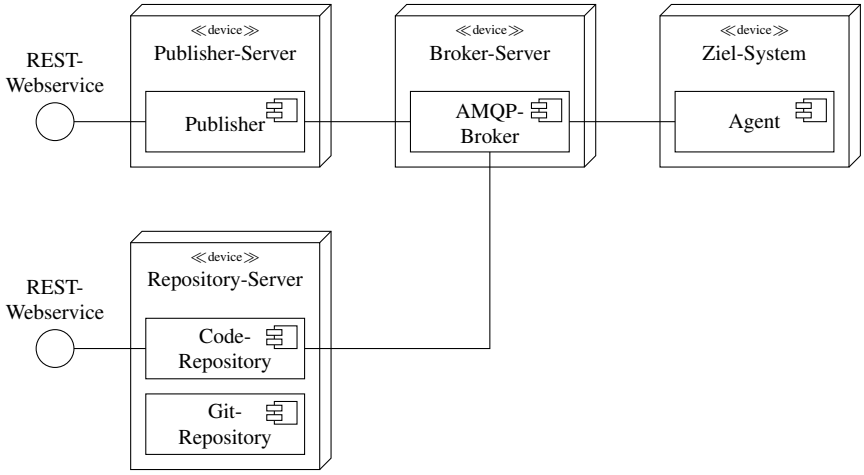

Abbildung 2: Ein exemplarisches Deployment-Diagramm der vorgestellten Systemarchitektur.

4 Systemarchitektur

Die einzelnen Systemkomponenten bestehen aus einem Publisher, der Drittapplikationen Schnittstellen anbietet, um beliebige Konfigurationsänderungen auf Hosts im Netzwerk durchzuführen. Der Publisher veröffentlicht zugestellte Konfigurationsaufträge in einem Message Broker, der diese an auf den Hosts laufende Software-Agenten zustellt. Ein Code-Repository hält in verschiedenen Skriptsprachen frei programmierbare Vorlagen für auf den Hosts auszuführende Skripte bereit, die von den Software-Agenten bei Bedarf abgerufen werden. Abbildung 2 zeigt eine exemplarische Verteilung der hier vorgestellten Architektur.

4.1 Publisher-Komponente

Ein Publisher ist jeweils dafür zuständig, Aufträge für eine von ihm verwaltete Gruppe von Hosts entgegen zu nehmen. Hierzu wird eine REST-Schnittstelle [Fie00] angeboten, die von Dritt-Applikationen angesprochen werden kann. Um trotz der grundsätzlich asynchronen Bearbeitung von Aufträgen einen Rückkanal anzubieten (etwa, um Applikationen über abgeschlossene Aufträge zu informieren), kann mit HTTP-Callbacks [Til11] gearbeitet werden. Listing 1 zeigt eine exemplarische REST-Anfrage, mit welcher eine Liste von DNS-Einträgen auf einem Host-Cluster erstellt werden kann; wurden die Aufträge bearbeitet, sendet der Publisher die Ergebnisse an die im `X-TXE-CallbackURI`-Header spezifizierte URI.

Listing 1: Beispiel eines REST-Aufrufs zum Erstellen eines neuen Auftrags

```
POST /tasks HTTP/1.1
Host: publisher.local
Authentication: ...
X-TXE-CallbackURI: https://client.local/finished/dns
X-TXE-HMAC: ...

[{
  "prodecure": "dns_create_records",
  "clusters": ["dns.external"],
  "hosts": ["dns01.internal", "dns02.internal"],
  "data": {
    "records": [
      {"type": "A", "host": "example.com", value: "1.2.3.4"},
      {"type": "AAAA", "host": "example.com", value="::abcd"}
    ]
  }
}]
```

4.2 AMQP-Broker

Die Zustellung der Aufträge an die Host-Systeme erfolgt durch eine nachrichtenorientierte Middleware [DEF+08]. Die bekanntesten Open-Source-Implementierungen des Standard-Protokolls AMQP [OAS14] sind Apache QPID [Apa14] und RabbitMQ [Piv14b]. Abbildung 3 zeigt eine exemplarische Konfiguration eines AMQP-Brokers, um Aufgaben an einzelne Hosts zu verteilen. Die Knoten N_2 und N_3 sind zudem in einem Cluster zusammengefasst und Nachrichten mit dem Routing Key *task.c1* werden im Round-Robin-Verfahren an jeweils einen Knoten des Clusters zugestellt.

Diese Konfiguration ist protokollgemäß jedoch weder statisch noch zentral vorgegeben, sondern wird zur Laufzeit von den einzelnen Clients konfiguriert [KD14]. Auf diese Weise ist es möglich, dass Agenten gegenüber dem Broker selbst spezifizieren, welche Nachrichten benötigt werden. Zudem können so auf einfache Weise Gruppen von Hosts verwaltet werden. Je nach Konfiguration ist es möglich, dass jeweils ein Host einer Gruppe eine Nachricht erhält oder alle Hosts jede Nachricht erhalten.

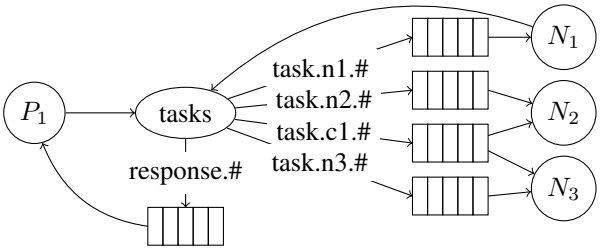

Abbildung 3: Eine exemplarische AMQP-Architektur mit einem Publisher und drei Knoten, die Tasks bearbeiten.

4.3 Subscriber-Komponente

Auf den zu verwaltenden Endsystemen arbeitet ein Software-Agent, dem vom AMQP-Broker neue Aufträge zugesandt werden. Hierzu wird eine Vorlage für ein auszuführendes Skript benötigt, die der Agent bei Bedarf aus dem Code-Repository nachlädt [ZGW+13]. Die Zustellung dieser Vorlagen kann ebenfalls über AMQP erfolgen (Abbildung 6). Auf diese Weise kann auch erreicht werden, dass ein Agent, nachdem einmalig eine bestimmte Skript-Vorlage angefordert wurde, automatisch über Änderungen an dieser Vorlage benachrichtigt wird. Abbildung 6 sowie Listings 2 und 3 zeigen die entsprechende AMQP-Konfiguration sowie exemplarische Anfragen und Antworten zur Anforderung von Vorlagen.

Zudem müssen die Software-Agenten auf den Zielsystemen in der Lage sein, komplexe Ketten voneinander abhängiger Aufträge host-übergreifend zu bearbeiten. Abbildung 4 zeigt eine solche Kette von Aufträgen, die sich über fünf verschiedene Hosts erstreckt: Zum Anlegen einer neuen Webhosting-Instanz mit vorinstallierter Anwendungssoftware müssen neue DNS-Records erstellt (t_D), Datenbank-Benutzer (t_U) und Datenbanken (t_B) angelegt, eine neue virtuelle Maschine erstellt und provisioniert (t_V), ein virtueller Host konfiguriert (t_H), Softwarepakete und deren Abhängigkeiten installiert (t_A) und zeitgesteuerte Dienste eingerichtet (t_C) werden. In diesem Beispiel kommt hinzu, dass einer dieser Hosts erst während dieser Auftragskette erstellt wird; dementsprechend muss der Agent in der Lage sein, sich selbst auf weiteren Host-Systemen zu installieren.

Für die Ausführung solcher Auftragsketten wären mehrere Lösungen denkbar. Im einfachsten Falle könnten die Aufgaben vom Publisher topologisch sortiert und sequentiell abgearbeitet werden. Für eine Host-übergreifende Lösung könnte der Publisher sicherstellen, dass die Aufträge erst dann zugestellt werden, wenn die zuvor benötigten Aufträge bearbeitet wurden. Eine solche sequentielle Bearbeitung von Auftragsketten wäre jedoch potentiell ineffizient, da (etwa wie in Abbildung 4) durchaus Teilaufgaben parallel bearbeitet werden können. Für eine effiziente Ausführung muss der Publisher also sicherstellen, dass nach Bearbeitung eines Auftrages stets alle weiteren abhängigen Aufträge (sofern keine weiteren Abhängigkeiten existieren) an die Zielsysteme zugestellt werden.

4.4 Code-Repository

Das Code-Repository verwaltet die Vorlagen für auf den Hosts auszuführende Skripte. Da das System eine große Anzahl heterogen konfigurierter Hosts verwalten muss, ist es sinnvoll, für jedes Skript Zielsystem-spezifische Facetten eines Skripts zu verwalten. So muss beispielsweise zur Installation eines Webservers auf einem Debian Linux-Host anders vorgegangen werden als auf einem CentOS-Host (Abbildung 5). Um eine Versionierung der Vorlagen sicherzustellen, werden diese im Hintergrund mithilfe eines SCM-Systems, wie etwa Git [Git14], verwaltet.

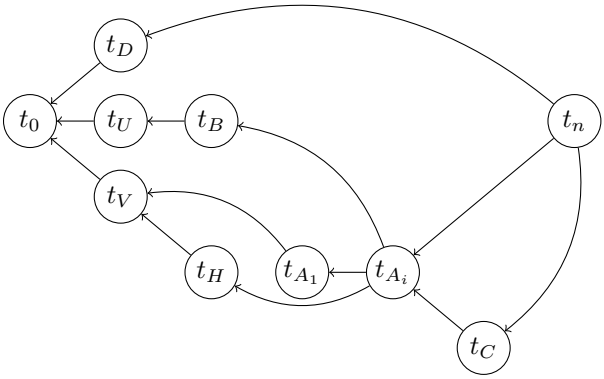

Abbildung 4: Beispiel einer komplexen Kette auf Aufgaben, die sich über fünf verschiedene Hosts verteilt (von denen einer dynamisch während dessen erstellt wird).

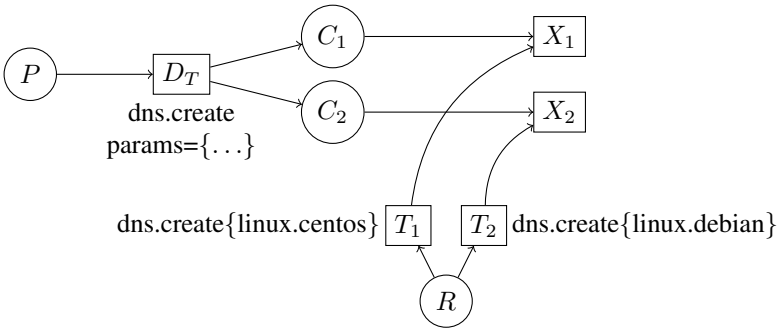

Abbildung 5: Ablauf, nach welchem die Software-Agenten C ausführbare Skripte X auf Grundlage der vom Publisher übermittelten Eingabeparameter D_T und einer von der Konfiguration des Host-Systems abhängigen Vorlage T erstellen.

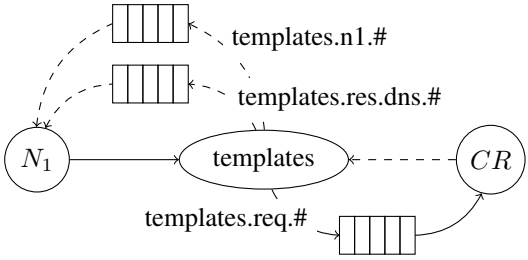

Abbildung 6: Nutzung von AMQP zum Zustellen von Skript-Vorlagen an den Software-Agenten.

Listing 2: Beispiel-AMQP-Nachricht eines Agenten zur Anforderungen einer bestimmten Vorlage

```
Routing-Key:
  templates.req.dns.create

{
  "template": "dns.create",
  "target_queue": "templates.
      node100",
  "host_name": "n1",
  "host_properties": {
    "os": "linux",
    "dist": "rhel",
    "version": "6.4",
    "service": "bind"
  }
}
```

Listing 3: Antwort auf die Anfrage nach einem bestimmten Template (Listing 2).

```
Routing-Key:
  templates.n1

{
  "template": "dns.create.linux.
      rhel.any.bind",
  "language": "bash",
  "content": "..."
}
```

4.5 Skalierungsszenarien

Da die Systemarchitektur auf einen AMQP-Broker als zentrale Komponente angewiesen ist, stellt sich hier die Frage nach der Skalierbarkeit des Systems. Diese Frage wurde auch in [MSPP10] behandelt. Hier werden verschiedene Möglichkeiten untersucht, föderierte AMQP-Broker mit Apache QPID zu realisieren. Auch RabbitMQ bietet mit *federation* und *clustering* sogar mehrere Mechanismen zur Verteilung und Skalierung von Brokern [Piv14a].

Alle übrigen Systemkomponenten sind aufgrund der Publish-Subscribe-Architektur von AMQP automatisch skalierbar. So können theoretisch beliebig viele Publisher Aufträge annehmen und über den Broker veröffentlichen und ebenso können beliebig viele Code-Repositories neue Skript-Vorlagen veröffentlichen. Eine Synchronisation mehrerer Code-Repositories untereinander kann über die eingebauten Synchronisierungsmechanismen von Git erfolgen.

5 Zusammenfassung und Ausblick

Abbildung 7 zeigt abschließend noch einmal die Kommunikationsbeziehungen der einzelnen Komponenten untereinander beim Bearbeiten eines einzelnen Auftrags durch das System.

Im Unterschied zu Lösungen wie [Che14], [Pup14b] oder [CFE14a] versucht die in diesem Artikel vorgestellte Architektur nicht, das Problem des Konfigurationsmanagements ganzheitlich zu lösen. Stattdessen konzentriert sich der hier vorgestellte Vorschlag bewusst

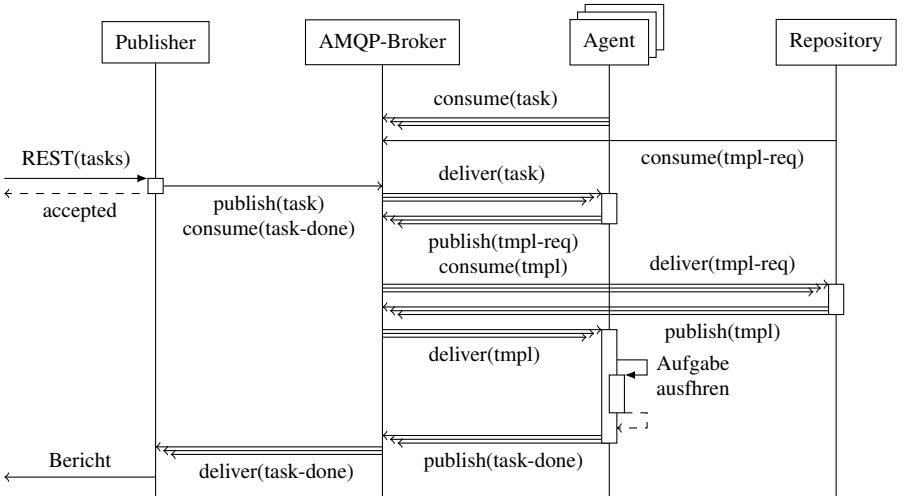

Abbildung 7: Ein Sequenzdiagramm, welches die Ausführung eines einzelnen Auftrags auf drei Zielsystemen verdeutlicht. Die Nachrichten von und zum AMQP-Broker entsprechen denen in [OAS14] definierten Methoden des AMP-Protokolls.

darauf, das Teilproblem der effizienten Ausführung von Skripten auf einer Vielzahl von heterogen konfigurierten Hosts (10.000 und mehr) unter Einhaltung strenger Zeitbedingungen von wenigen Minuten zu lösen.

Das vorgestellte System bietet eine zentrale Schnittstelle an, um beliebige Aufträge auf entfernten Systemen durchzuführen. Um ein ganzheitliches Konfigurationsmanagement zu erreichen, werden weitere Systemkomponenten benötigt, die den erwünschten Zustand des Systems modellieren und Änderungen an diesem Zustand über entsprechende Schnittstellen publizieren.

Im Unterschied zu vergleichbaren Lösungen abstrahiert die hier vorgestellte Architektur die Kommunikation mit den einzelnen Hosts durch den Einsatz einer nachrichtenorientierten Middleware. Weiterhin wird auf den Endsystemen auszuführender Code zentral vorgehalten, und es können komplexe, hostübergreifende Auftragsketten orchestriert werden. Die zugrunde liegende AMQP-Middleware ermöglicht es, alle Komponenten redundant und somit hochverfügbar und skalierbar zu halten.

Literatur

[Ans14] Ansible, Inc. Ansible is Simple IT Automation. http://www.ansible.com/home, 2014.

[Apa14] Apache Software Foundation. Apache Qpid: Messaging built on AMQP. http://qpid.apache.org/, 2014.

[BC06] Mark Burgess und Alva L. Couch. Modeling Next Generation Configuration Management Tools. In *LISA*, Seiten 131–147, 2006.

[Bur95] Mark Burgess. Cfengine: a site configuration engine. In *USENIX Computing systems*, Jgg. 8, Seiten 309–337, 1995.

[CFE14a] CFEngine AS. CFEngine: Configuration management software. http://cfengine.com/index, 2014.

[CFE14b] CFEngine AS. CFEngine: Scale and Scalability. https://cfengine.com/archive/manuals/reader, 2014.

[Che14] Chef Software, Inc. Chef: IT automation for speed and awesomness. http://www.getchef.com/, 2014.

[CHIK03] Alva L. Couch, John Hart, Elizabeth G. Idhaw und Dominic Kallas. Seeking Closure in an Open World: A Behavioral Agent Approach to Configuration Management. *Proceedings of the 17th Large Installations Systems Administration (LISA) conference (San Diego, CA, USA, 10/2003), Usenix Association*, 3:125–148, 2003.

[DEF+08] Jürgen Dunkel, Andreas Eberhart, Stefan Fischer, Carsten Kleiner und Arne Koschel. *Systemarchitekturen für verteilte Anwendungen: Client-Server, Multi-Tier, SOA, Event Driven Architecture, P2P, Grid, Web 2.0*. Carl Hanser-Verlag, München, 1. Auflage, September 2008.

[DJV10] Thomas Delaet, Wouter Joosen und Bart Vanbrabant. A survey of system configuration tools. *Proceedings of the 24th Large Installations Systems Administration (LISA) conference (San Jose, CA, USA, 11/2010), Usenix Association*, 2010.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Dissertation, University of California, Irvine, 2000.

[FLRU13] J.L. Fernandes, I.C. Lopes, J.J.P.C. Rodrigues und S. Ullah. Performance evaluation of RESTful web services and AMQP protocol. In *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*, Seiten 810–815, Juli 2013.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar und Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework, April 2007.

[Git14] Git-SCM. Git. http://git-scm.com/, 2014.

[Hos13] Ben Hosmer. Getting Started with Salt Stack: The Other Configuration Management System Built with Python. *Linux Journal*, Januar 2013.

[KD14] Michael Klishin und Chris Duncan. The AMQP 0-9-1 Model Explained, 2014.

[Mit14] Mittwald CM Service GmbH & Co. KG. Reseller Webhosting für Agenturen & Freelancer. https://www.mittwald.de/, 2014.

[MSPP10] Gregory Marsh, Ajay P Sampat, Sreeram Potluri und Dhabaleswar K Panda. Scaling Advanced Message Queuing Protocol (AMQP) Architecture with Broker Federation and InfiniBand. Bericht, 2010.

[OAS12] OASIS. OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html#toc, Oktober 2012. OASIS-Standard.

[OAS14] OASIS. AMQP Architecture. https://www.amqp.org/product/architecture, 2014.

[Piv14a] Pivotal Software, Inc. RabbitMQ: Distributed RabbitMQ brokers. http://www.rabbitmq.com/distributed.html, 2014.

[Piv14b] Pivotal Software, Inc. RabbitMQ: Messaging that just works, 2014.

[Pup14a] Puppet Labs, Inc. MCollective: Programmatic Execution of Systems Administrations Actions. http://puppetlabs.com/mcollective, 2014.

[Pup14b] Puppet Labs, Inc. Puppet Enterprise: IT Automation Software for System Administrators. http://puppetlabs.com/puppet/puppet-enterprise, 2014.

[Sal14] SaltStack, Inc. Salt Documentation, Release 2014.1.1. https://media.readthedocs.org/pdf/salt/latest/salt.pdf, März 2014.

[Til11] Stefan Tilkov. *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien*. dpunkt.verlag, Heidelberg, 2. Auflage, 2011.

[TvS07] Andrew S. Tanenbaum und Maarten van Steen. *Verteilte Systeme: Prinzipien und Paradigmen*. Pearson, 2. Auflage, November 2007.

[Vin06] S. Vinoski. Advanced Message Queuing Protocol. *IEEE Internet Computing*, 10(6):87–89, November 2006.

[VJ13] Bart Vanbrabant und Wouter Joosen. A framework for integrated configuration management tools. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, Seiten 534–540. IEEE, 2013.

[YZF+11] Jiayuan Yue, Zhuo Zhang, Junhong Fu, Shengqing Lu, Xinmin Li und Yunfu Shen. Extensible architecture for high-throughput task processing based on hybrid cloud infrastructure. In *2011 International Conference on Electronics, Communications and Control (ICECC)*, Seiten 1452–1455, September 2011.

[ZGW+13] Sai Zeng, Shang Guo, Fred Wu, Constantin Adam, Long Wang, Cashchakanithara Venugopal, Rajeev Puri und Ramesh Palakodeti. Universal Script Wrapper – An innovative solution to manage endpoints in large and heterogeneous environment. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, Seiten 916–919. IEEE, 2013.