Modularization of Software as a Service Products: A Case Study of the Configuration Management Tool Puppet

Johannes Hintsch, Carsten Görling, and Klaus Turowski
{johannes.hintsch|carsten.goerling|klaus.turowski}@ovgu.de
Faculty of Computer Science, Otto von Guericke University
P.O. Box 4120, 39106 Magdeburg, Germany
http://mrcc.ovgu.de

***Abstract*—With the increasing adoption of business models and technologies related to cloud computing, the business of software as a service is a growing market. Distributed computing environments with multiple hosts often require complex configuration. In order to increase operational efficiency, configuration management tools use the concept of modules, which decrease repetition in the work of system operators. These tools can be combined with ERP systems to automate the order-to-cash processes for production of software as a service products using bill of materials to represent the products in the ERP systems. In this research, the configuration management tool Puppet is studied to investigate how the concept of bill of materials maps to the module concept of Puppet by using the examples of two Puppet modules for Hadoop and OpenStack. The usage of Puppet modules in ERP system controlled and automated order-to-cash processes for complex software as a service products such as Hadoop and OpenStack can be validated, but areas of future work are also identified.**

***Keywords*—Configuration Management, Bill of Materials, Puppet, Hadoop, OpenStack, ERP, Software as a Service.**

I. Introduction

Since the rise of cloud computing, IT services have taken center stage for transforming the ways in which software is developed, operated, and consumed [1]. The challenges of offering IT services do not lie in managing the development of complex, possibly monolithic, software systems. Offering IT service has similarities to producing physical goods [2]. This analogy between IT service production and physical goods production is investigated by business informatics researchers under the topic "industrialization of IT". These investigations aim at increasing efficiency and effectiveness of IT service production (cf. Walter et. al [2]).

To research which characteristics of the industrialization of IT are already implemented in practice, Becker et al. [3] conducted a case study. They found that characteristics like process standardization, modularization, and quality management are widely adapted, but that further potential for improvement lies in the areas of sourcing, automation, process standardization, and modularization.

This study's investigation is conducted within the IT service management domain of configuration management. The IT Infrastructure Library (ITIL) [4, p. 328] defines service asset and configuration management as the following.

> *The process responsible for ensuring that the assets required to deliver services are properly controlled, and that accurate and reliable information about those assets is available when and where it is needed. This information includes details of how the assets have been configured and the relationships between assets.*

Configuration management in IT can be compared to variant production because the configurations of the whole IT system landscape are managed in order to consistently meet the individual purposes of the services that the landscape offers (cf. [5]). Configuration management tools support this process on an operational level. They automate the configuration of large IT landscapes. Configurations can be modularized into reusable configuration modules. Furthermore, they standardize by giving a common language for configuration.

Motivated by the findings of Becker et al., an automated order-to-cash process for software as a service (SaaS) products was conceptualized by Hintsch et al. [5]. The order-to-cash process is supported by an Enterprise Resource Planning (ERP) System. Aiming at the growing SaaS market [6], with this process model arbitrary complex SaaS products can automatically be provisioned based on customers' individual requirements, achieving high efficiency and consistent quality across deployments. The model uses the concept of bill of materials (BOM) to specify SaaS products in the ERP system. These BOMs are transformed to configuration specifications that configuration management tools can process to provision the SaaS products' applications. The process' general feasibility was demonstrated by the implementation of a prototype using the popular [7] configuration management tool Puppet. However, the capabilities of the module concept of Puppet were not sufficiently evaluated. The evaluation's significance should be extended because, for the aforementioned prototype, only the illustrative example of provisioning an open source content management system as a SaaS product was used. Also the validity of the analogy between Puppet's modules and bill of materials should be further researched. Although Cfengine and Chef are also frequently mentioned as popular tools [7], in this study, as the three tools' modularization concepts are comparable [8], only Puppet is studied. The leading research question is:

Research question: Is the module concept of the configuration

978-1-4673-8005-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ES.2015.25

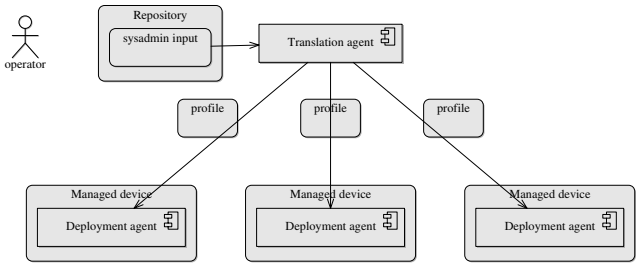

Fig. 1. Configuration management tool architecture [8]

management tool Puppet adequate in order to implement provisioning of SaaS products that are represented by bill of materials and that have a degree of complexity that goes beyond that of illustrative examples?

The paper is structured as follows. In section II the research background is described. It contains an overview of how configuration management tools work, a summary of the order-to-cash process concept, and other work in this field. In section III Puppet and related tools are presented. This helps to understand the study of the Puppet modules for Hadoop and OpenStack in section IV, which is performed to answer the research question. In section V the paper is summarized and a conclusion drawn. An outlook on future work is given in section VI.

II. Research Background

This section starts with a description of the paper's central topic of configuration management tools.

A. Configuration Management Tools

Delaet et al. [8] present a comparison framework for configuration management tools. They also define the tools' architectural essence that is displayed in figure 1. A configuration management tool provides the administrator with an interface, which is used to specify the configuration of the managed devices. These specifications are stored in a repository. Device-specific profiles, which represent the configuration specifications, are generated and the deployment agents of the managed devices configure the device as specified.

The comparison of eleven configuration management tools with the comparison framework by Delaet et al. [8] yields, among others, the following relevant results: among open source tools like Puppet, several proprietary tools exist. These proprietary tools, unlike their open source counterparts, do not allow configuration specification in a declarative fashion. However, Delaet et al. [8] report that declarative specifications are more robust than the paradigm of imperative scripting used by proprietary tools. Delaet et al. [8] also call for the introduction or usage of higher-level abstractions when specifying configurations. In an example, they map end-to-end requirements, such as configuring enough mail servers in order to guarantee a certain response time, to concrete configurations for instances. Wettinger et al. [9] address this problem with the aim of improving the manageability of cloud services. They integrate the configuration management tool Chef and the Topology and Orchestration Specification for Cloud Applications (TOSCA) by OASIS. Their concept allows the modeling of a topology of cloud applications. Combined with configuration management tools that implement applications on single instances, this realizes the demand for higher-level abstraction.

Besides Puppet, frequently mentioned open source tools are Chef and Cfengine [7]. However, Schaefer et al. [10] favor Puppet because they report that it is more feature rich than Cfengine and more mature than Chef.

Configuration management tools are not the only option to automate or modularize configuration of a large system landscape. This may also be achieved by writing custom scripts that automate configuration tasks [11]. However, an advantage is the common language for configuration specifications provided by these tools, which make the specifications sharable for common configuration tasks. One key aspect of open source software is that it becomes more reliable if a lot of people use it and review the code. The same is true for frequently used configuration specifications [12]. Managing large system landscapes can also be achieved by configuring virtual machines as templates that are reused. However, centrally updating these machines is not covered by this approach and additional concepts and tools would be needed. A similar concept to that of preconfigured virtual machines is that of containers such as Docker[1] (cf. [13]. Docker containers contain applications that can be transferred between machines without having to reconfigure them on each machine. Other than virtual machines, they do not contain an operating system, but only the applications and their dependencies necessary to run the contained applications. The processes of one container run strictly isolated from other containers. The containers may be recombined to form larger applications. However, this combination also has to be specified.

In the next subsection the order-to-cash process for SaaS products is presented, which motivated the work presented in this paper.

B. Order-to-cash Process for SaaS Products

An IT product is composed of one or more IT services, and IT services can be organized in a service model. IT services consist of preliminary services. These preliminary services use resources, which are either human, information, application, or infrastructure as service (IaaS) resources [5]. A preliminary service has possibly multiple bills of configurations, which consist of application configuration models and instance configuration models. The term "bill of configurations" is used instead of bill of materials in order to illustrate the centrality of configuration for the concept. The instance configuration model specifies characteristics of IaaS resources that are used by the preliminary service. The application configuration model contains application configuration options such as a configured access right or a heap size in an application. An application configuration model can be described by Puppet modules. The model was evaluated with a prototype implementation.

The order-to-cash process is presented in figure 2. For each process step, the software components that were used in the prototype are also displayed. SAP ERP was used to generally support the order-to-cash process. A bill of materials is

[1] https://www.docker.com/

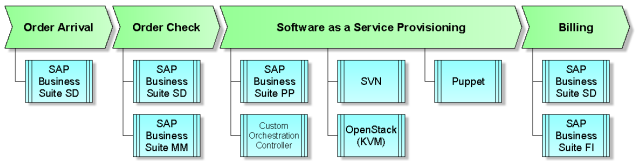

Fig. 2. Order-to-cash process with software components used in the prototype

sent from the ERP system's production module to a custom orchestration controller. This orchestration controller gets the application and instance configuration models from a repository and prompts the IaaS provider to spawn an instance and configures the application using Puppet, thus provisioning the SaaS product. The process presented in [5] does only focus on the provisioning step, not the further steps of the service lifecycle such as service operation. This will also be the limitation of this paper. OpenStack was used as a IaaS software. In ITIL terminology a resource would be an asset.

The following section presents previous research done by other authors on configuration management tools.

C. Previous Research on Configuration Management Tools

Several authors [11][14][15] describe grid computing scenarios in which configuration management tools are used. The contribution of this work is argued to be the concept for automating the management of grid computing landscapes. These concepts include bootstrapping physical machines with network boot mechanisms, automatically assigning IP addresses via the DHCP protocol, and configuring the application stack using the configuration management tool.

Meyer et al. [12] propose an automated quality assurance service for configuration specifications. In the context of this paper this would mean quality assurance for Puppet modules. The authors describe the problem that configuration specifications are deployed to different environments where, for example, package names may be different and therefore, these configuration specifications would not correctly function in different environments. For publicly available configuration specifications, oftentimes no quality measures other than download numbers, user ratings and comments, or currentness exist. Therefore, the authors propose a service that spawns virtual machines via IaaS and tests the configuration specifications by configuring the virtual machines. A report would then be published which can for instance be used in a catalog for configuration specifications.

Meyer et al. [12] also list quality issues for configuration specifications. Errors often occur because of operating system heterogeneity. They point out that configuration specifications, which are publicly available, often have hidden dependencies that are not documented. Furthermore, they report that configuration management tools are often updated, and that backward compatibility of configuration specifications is not always guaranteed.

Wettinger et al. [13] identify a problem with idempotent configuration specifications. Idempotent configuration specifications are a concept of configuration management tools that specify configuration declaratively. Such specifications cause the system to be transferred into a specified state, no matter how often the configuration management tool executes the specification. Designing such specifications can be challenging because a system, which shall be configured, can be in a state that the original author does not anticipate during design of the specification. Thus, side effects may occur that lead to errors. Such errors can be, for example, unavailable package resources that lead to an incomplete installation and require manual intervention. The authors propose a method to undo actions that were erroneous so that when the configuration specification is applied again, no manual intervention is necessary.

In the following section, an overview of the configuration management tool Puppet is given, and tools that are commonly associated with it are presented.

III. Puppet and its Periphery

The first subsection describes Puppet's modules and architecture. Compared with variant production of physical goods that uses bill of materials to define product composition, variant production of SaaS products requires means to test the combination of different software components to avoid side-effects. Therefore, the second subsection describes Puppet's periphery that can assist in managing this complexity.

A. Puppet Modules and Puppet's Architecture

Configuration specifications are written in scripts so that configuration management tools may process these scripts. For Puppet, such scripts are called manifests. These manifests can be put together to form a Puppet module and abstract away the complexity of bulky configuration tasks, such as configuring a MySQL database.

A module has a standard manifest, which is called *init.pp*. In this manifest, the interface of the module is specified. Listing 1 shows the *init.pp* of a module that configures the text editor vim by defining the content of the vim's configuration file *.vimrc*, which resides in a user's home directory. Class and module in the *init.pp* always have the same name.

Listing 1. A sample *init.pp* manifest of module *vimrc*

```
#Content of init.pp of module [authorname]-vimrc:
class vim ($username = undef, $userdir = '/home') {
  package { 'vim': ensure => present }

  file { "$userdir/$username/.vimrc":
    source => 'puppet:///modules/vim/vimrc',
    ensure => present,
    owner  => $username,
    group  => $username,
    mode   => '644',
  }
}

#The following invokes the configuration of vim for
#user jsmith on the node foohost in the site.pp by
#parameterizing it with the desired username.
#The home directory's path, because it is not
#explicitly specified, defaults to /home
node 'foohost' {
  class { 'vim':
    username => 'jsmith',
  }
}
```

Puppet's domain specific language (DSL) for specifying configurations defines *resources*. The listing 1, for instance, specifies with the resource *package* that the package of vim shall be installed. With *file*, it specifies the file *.vimrc*.

The standard directory layout of Puppet modules contains the directories *manifests*, *files*, *templates*, *lib*, *facts.d*, *tests*, and

spec [16, Chapter: Module Fundamentals].
The directory *manifests* contains the *init.pp* and additional manifests that can be used to structure more complex modules. Files that, for instance, contain application data and do not change if the module is parametrized differently, can be stored in the *files* directory. Configuration files whose content changes when the module is parametrized are specified by *templates* which may themselves be parametrized. The *lib* directory contains plug-ins that offer functionality that goes beyond the standard features of Puppet. Puppet also defines *facts*. These are values such as a computer's host name or operating system name. The directory *facts.d* contains executables that may be written in an arbitrary programming language to construct custom facts. Examples illustrating the usage of a module are contained in the directory *tests*. The directory *spec* contains tests, comparable to unit tests, that are written to ensure the function of the module's plug-ins. [16, Chapter: Module Fundamentals]
The architecture of Puppet is similar to the generalization by Delaet et al. [8] (figure 1). The Puppet master contains all Puppet modules and a manifest called *site.pp* specifies for each node of the managed system landscape how it shall be configured. For this, the Puppet agent, which is responsible for configuring its node, sends *facts* about its node to the Puppet master. The master responds by sending a compiled catalog that specifies how the node should be configured. The agent then responds to the master by reporting about the status of the configuration.
In the next subsection, the periphery of Puppet is described, which is also necessary to understand the content of section IV.

B. Puppet's Periphery

Developing Puppet modules can involve programming, in particular when complex configuration tasks shall be performed by idempotent scripts and custom plug-ins are developed. However, modules that are only specified using Puppet's DSL also need to be tested. The development of modules can be structured along process steps of a software lifecycle: requirements analysis, design, development, testing, and operation.
When a module is developed by a single administrator, it is then likely that the requirements analysis step is performed implicitly because he is the only stakeholder involved. It is also likely that most modules grow and are developed iteratively, but with the growing number of involved stakeholders, managing requirements becomes a task that needs to be performed explicitly as, for example, practiced in agile software development methodologies [17]. The design and development phase also correspond to software development, for instance module manifests are version controlled the same way as normal code [7].
For testing, things are different. For plug-ins that extend Puppet's standard functionality, known approaches such as unit testing are suitable. But when testing Puppet modules that specify the configuration of nodes traditional unit testing approaches are unfeasible. Here, virtualization is key. A tool that is commonly used by Puppet is Vagrant. Vagrant is a wrapper tool that abstracts the functionality of specific desktop virtualization software products as those offered by VMware[2], but also that of IaaS platforms. A file is used to specify the characteristics of a virtual machine such as CPU, memory, and networking. It also specifies an image of a virtual machine that is preconfigured. These images are called Vagrant boxes. The file that is called *Vagrantfile* may be kept in a code versioning repository along with the Puppet module's code. Vagrant may be used to set up approximations of complex system landscapes on a developer's desktop computer or laptop that can structurally be identical to production environments. Vagrant is commonly used to test the functionality of Puppet modules, which usually involves multiple virtual machines. Vagrant does not alter the virtual machine image so that if something goes wrong during configuration, it is very easy to go back to the predefined state. Complex configurations defined by SaaS products' BOMs can be tested this way, ruling out configuration issues of different variants of a SaaS product. Puppet may configure physical machines or virtual machines. As the main area of Puppet's application lies in managing large numbers of nodes, particularly in cloud computing scenarios, IaaS systems play a central role in the operation step. Puppet offers integration[3] with the well-known EC2 from Amazon, but is also integrated[4] with the open source cloud computing software platform OpenStack.
In development, Puppet is closely related to the paradigm of DevOps [7] because it enables professionals from development and operation departments to work with the same configuration of an application under development. DevOps has a strong relation to the practice of continuous delivery [18]. The relationships between DevOps and Puppet's periphery are illustrated by the concept map [19] in figure 3. The next section describes the study of two publicly available Puppet modules.

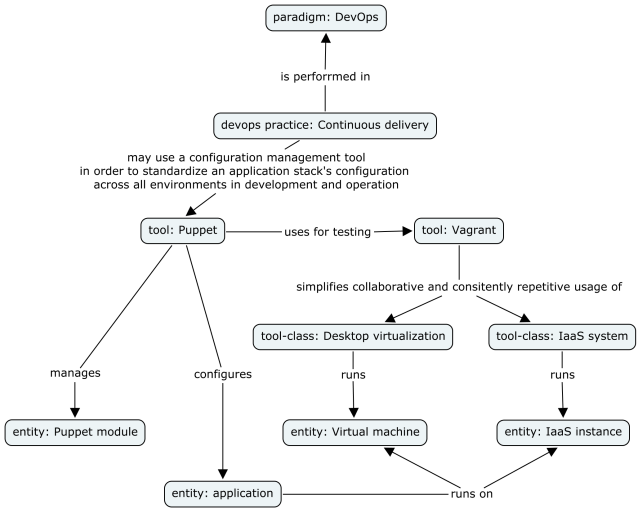

Fig. 3. Concept map of Puppet's periphery

IV. Study

To answer the paper's research question if the module concept of Puppet is adequate to implement provisioning of SaaS products that are represented by bill of materials and

[2]http://www.vagrantup.com/vmware

[3]https://puppetlabs.com/solutions/ec2

[4]https://wiki.openstack.org/wiki/Heat

that have a degree of complexity that goes beyond that of illustrative examples, a case study of Puppet is conducted. The following subsections report on this study.

A. Methodology

The methodology is aligned with the guidelines proposed by Eisenhardt [20] and Yin [21]. The following steps were performed: *(1)* definition of research question, *(2)* case selection, *(3)* definition of unit of analysis, *(4)* data collection, *(5)* linking of data to propositions, and *(6)* interpretation of findings.
The research question *(1)* was defined in the introduction of this paper. The selected case *(2)* is the configuration management tool Puppet. Puppet was selected because it is used in the automated order-to-cash process' prototype [5] and it is reported to be a better alternative than the two other popular open source configuration management tools Chef and Cfengine [10]. As unit of analysis, *(3)* modules that are publicly available through the Puppet module database puppet-forge[5] were selected. Specifically, Hadoop and OpenStack were chosen. Hadoop[6] allows to distributively process large data sets across computer clusters. In order to provision such an installation of Hadoop, all the nodes of a cluster have to be configured. The same is true for OpenStack, where nodes, which, for example, are responsible for computing, storage, and networking, have to be configured. Configuring large quantities of nodes is a common task for configuration management tools [11][14][15]. Therefore, Puppet modules that configure and provision these tools were selected. For Hadoop, out of the modules that were on available puppet-forge a module was selected because it had most recently been updated. The OpenStack Puppet module was selected because it is provided directly by the vendor of Puppet.
The collection of data *(4)* in late February 2015 was performed by using the modules to deploy and configure Hadoop and OpenStack in test environments. The researcher performing the configurations downloaded the modules, performed the configuration, and took notes during the process. The collected data is presented in the next subsection, followed by a presentation of the fifth *(5)* step of the case study, and the interpretation of the findings *(6)* in the final subsection.

B. Puppet Modules for Hadoop and OpenStack

Table I displays characteristics of the modules.
A module is unambiguously identified on puppet-forge by the identifier that consists of the name of the creator followed by a hyphen which is then followed by the module name and by its version. While the module for OpenStack was first released in June 2012, the Hadoop module is relatively young, having first been released on puppet-forge in January 2015.
Both modules are designed to support the configuration of the two applications on a variety of operating systems that are all Linux distributions. For Hadoop, package sources that, for instance, are provided by the Hadoop distributor Cloudera, have to be added to the package manager of the operating system. Additionally, a java development kit (JDK) has to be installed in order for Hadoop to function. Both modules are documented and provide samples that illustrate how the modules are to be used.

[5]https://forge.puppetlabs.com/
[6]http://hadoop.apache.org/

TABLE I. THE TESTED PUPPET MODULES

Module	Hadoop	OpenStack
Identifier	cesnet-hadoop	puppetlabs-openstack
Version	0.9.5	5.0.2
Releases	6	20
1st release date	Jun. 13th 2012	Jan. 20th 2015
Supported OS	Fedora 21, Debian 7/wheezy (Cloudera distribution), Ubuntu 14.04	RedHat 7 variants RHEL, CentOS, Scientific Linux), Ubuntu 14.04
Requirements	Hadoop repositories, JDK	a minimum of two network interfaces for each node
Documentation	yes	yes
Samples	yes	yes
# of req. modules	2	32
Max. depen. level	1	6
Total size	2.5 MB	17 MB
# of downloads	778	16,658
Date of last update	Feb. 15th 2015	Jan. 27th 2015
St. code anal. score	4.8/5	5/5
Vendor qlty. label	no	no
Alt. avail. modules	6	7
Deployment types	single-node, multi-node	single-node, multi-node
Multiple runs req.	yes[1)]	no

Remark: data is from 10th of March 2015
1) Necessary because distributed file system nodes have to run before other Hadoop services can be started.

While Hadoop only has two modules on which it depends, OpenStack depends on 32 modules. These dependency modules are also available over puppet-forge and are installed by Puppet's module package system automatically when installing the desired module. A collapsed view of the dependency tree is shown in listing 2. The deepest level of this dependency tree is level six. Although Hadoop is architecturally divided into different components the module is not divided into "submodules" resembling these components. For OpenStack, the application is also architecturally divided into different components. However, there are core components and components that are optional. Every OpenStack component has its own Puppet module.
The listing 2 also testifies to the practice of reusing modules that are available through puppet-forge. For instance, a module for Apache the web server as well as one for the MySQL database are reused. Both modules were created by Puppet-Labs. Modules that were created by other community members, such as a module for installing the run-time environment of the programming language Erlang from Erlang's official package repositories on a range of Linux distributions, are also used, highlighting reusability across organizations. The size in megabyte (MB) of the modules may attest to the different complexities of the applications they configure.
Different criteria exist for choosing a module. The number of downloads by other users, the date of the last update, which may indicate if the module is actively maintained, a static code analysis score, and a quality label[7] by PuppetLabs help users decide whether or not they want to further investigate if a module fits their needs. This is useful when considering that for Hadoop six and for OpenStack seven alternative modules can be downloaded.
Both selected modules allow to deploy the components of the applications on multiple nodes or on one node. A single-node deployment may be useful when testing out the application.
Puppet currently does not provide a mechanism to express

[7]https://forge.puppetlabs.com/approved

inter-node dependencies. For configuring Hadoop two configuration runs have to be performed, as described in the footnote of table I.

Listing 2. Dependencies of the module *puppetlabs-openstack, v5.0.2*

```
+--\AC puppetlabs-openstack (v5.0.2)
  +--\AC stackforge-ceilometer (v5.0.0)
  | |-- puppetlabs-inifile (v1.2.0)
  | +--\AC stackforge-keystone (v5.0.0)
  |   +--\AC puppetlabs-apache (v1.3.0)
  |   | -- puppetlabs-stdlib (v4.5.1)
  |   | -- puppetlabs-concat (v1.2.0)
  |   +--\AC stackforge-openstacklib (v5.0.0)
  |     -- aimonb-aviator (v0.5.1)
  |     -- puppetlabs-mysql (v2.3.1)
  |     +--\AC puppetlabs-rabbitmq (v3.1.0)
  |       -- puppetlabs-apt (v1.7.0)
  |       +--\AC garethr-erlang (v0.3.0)
  |         -- stahnma-epel (v1.0.2)
  --\AC stackforge-cinder (v5.0.0)
  [eight further tree nodes omitted]
```

Both modules were successfully used to deploy and configure Hadoop and OpenStack in virtualized environments that were set up using Vagrant and the desktop virtualization software VirtualBox[8]. The Hadoop module installed without problems.

Initially, a Windows-operated computer was used to install OpenStack on. However, the Vagrant-enabled examples that were included in the OpenStack Puppet module used additional tools and scripts that were designed to be run on a Linux or Mac operating system. Emulating Linux produced compatibility issues. Thus, it was decided to use a derivative of Ubuntu Linux[9] instead. A second problem that arose was that the example provided with the module used a Vagrant box that can only be run with the non-free VMware desktop virtualization software products. Therefore, Vagrant boxes with different operating systems were tried instead. However, this lead to several errors that included a misconfigured or outdated Puppet installation or missing package sources. Using the information of supporting operating systems, this problem could be solved. The OpenStack deployment and configuration could then be performed. However, Puppet consistently reported an error regarding connection issues for Neutron, OpenStack's network provider, during the configuration. This seems to be a problem that is known because a software product, which is frequently mentioned in forums where issues of OpenStack configurations are discussed, is Packstack[10]. This is yet another tool that wraps around the Puppet modules to deploy and configure OpenStack.

In the next section, the collected data is linked to propositions that are derived from the research question in order to answer it.

C. Linking of Data to Propositions

Table II shows the developed propositions. The concept for the automated order-to-cash process is introduced to increase efficiency and maintain high quality across deployments of possibly heterogeneous complex SaaS products. As Puppet was used in the prototype, *P1* was formulated. The second proposition (*P2*) states that it is possible to use Puppet modules for producing SaaS products, supported by ERP systems, like physical goods.

[8]https://www.virtualbox.org/

[9]http://www.ubuntu.com/

[10]https://wiki.openstack.org/wiki/Packstack

TABLE II. PROPOSITIONS

Proposition	Description
P1	Puppet modules increase efficiency and achieve consistent quality through modularization and standardization.
P2	Puppet modules correspond to the production-related concepts of bill of materials, work plan, and equipment.

TABLE III. BENEFITS AND CHALLENGES REGARDING PROPOSITION 1

Type[1)]	Name	Description
+	reuse	Puppet modules enable the reuse of configuration specifications. Puppet-forge is a platform to exchange these modules.
-	linux	Modules only support Linux distributions although e.g. Hadoop supports Windows.
-	package	Packages of applications have to be accessible over the operating system's package management system. Puppet only tells the package manager to install a desired package. Methods exist to add application data to modules or load binaries from remote sources. This, as an example, is necessary when installing Oracle's distribution of the JDK, which is not available in Ubuntu's package repositories.
-	unix	Modules and associated examples are usually written on Linux or Mac. This makes testing on Windows difficult as convenience scripts often require a UNIX environment such as a UNIX shell.
-	non-free	Puppet and most modules on puppet-forge are Apache licensed, thus enabling for-free usage. However, proprietary software, in this case from VMware, is required by the OpenStack module's example. This restricts the use of the example as-is to non-free usage scenarios, in which additional proprietary software is required.
-	complexity	Modules introduce a new layer of software complexity. This complexity has to be managed. Modules may contain bugs or they may have incompatibilities.

1) Type of observation: + (benefit), - (challenge)

Observations of benefits and challenges of using Puppet modules are described in table III (*P1*). Table IV shows the mapping between an excerpt of the production concepts with the terms of the domain discussed in this paper. This mapping is only valid for provisioning SaaS products [5] not for their operation, also it assumes that one SaaS product has exclusive access to its resources.

BOM is mapped to bill of configurations, preliminary service, and IT service. The same way a BOM structures the composition of components of a physical good, the three entities structure the composition of an IT product. A work plan can be seen as a configuration model that combines and parametrizes different configuration specifications (Puppet modules). The relation between equipment and resources is obvious as, for example, the machines that are one kind of equipment are mappable to the physical servers that are one kind of resource. However, if bill of materials is linked to bill of configurations, what are then materials or inputs in SaaS product production? The findings are interpreted in the following subsection.

TABLE IV. MAPPING OF PRODUCTION CONCEPTS

Prod. of Physical Products	Prod. of SaaS Products
Bill of materials	Bill of configurations, preliminary service, IT service
Work plan	Configuration model
Equipment	Resources

This is an excerpt of a mapping shown in [5].

D. Interpretation of Findings

The findings regarding *P1* testify to the benefits and challenges of Puppet and its module concept as described in table III. Puppet automates, which already increases efficiency and addressing the research question of this paper, the module concept leads to the *reuse-benefit*. Collaboratively using and improving modules may increase the efficiency not only of one administrator or one organization, but of the community of professionals of a whole industry.

However, challenges were also identified. The vendor of Puppet should carefully consider the *non-free-challenge*. We argue that for-free software that equals the quality of traditional non-free software can lead to a high dissemination of for-free solutions because users can use the software without purchasing a license and even improve the software, as is the case with Linux. For vendors it appears to be difficult to find a balance between attracting new users and developers through making available for-free and open source software while still operating with a profitable business model. This difficult trade-off can also be seen with PuppetLabs' involvement with VMware, which restricts some features to VMware customers. Even though this might be seen as a problem by users, it also can be seen as a positive aspect for organizations that use VMware and can integrate Puppet with more ease into their VMware-enabled system landscape.

Although Puppet modules help to increase efficiency, the *complexity-challenge* has to be considered. As Puppet modules are software by themselves, bugs that come with software may result in errors, instability, or security risks.

The *linux-*, *unix-*, and *unix-challenge* can be seen as specific to Puppet, but the *reuse-benefit* and the *complexity-challenge* should be applicable to Cfengine or Chef also.

Addressing *P2*, the question arose what materials correspond to. Production can be described as a process that transforms input to output (cf. Gutenberg [22]). Inputs are production factors such as material, equipment, and manpower. Outputs are goods and services, or summarized products. According to Zarnekow [23, pp. 87-106], in IT service production, inputs can be classified into external and internal production factors. Internal production factors are information, whereas external production factors can be seen as the user input who uses an IT service. Internal and external production factors are transformed to a desired output (the user using the IT service) by a running application service.

An application service is composed of resources such as IaaS resources and applications. Now, considering the question what materials might be in this context, it seems reasonable to map material to the internal production factor of configuration information. A revised version of the mapping of production concepts from table IV is shown in table V. Configuration information is formalized and made executable through configuration specifications. The previously heterogeneously and distributively stored configuration information is made explicit in the form of Puppet modules. We drop the term configuration specification and subsequently use configuration module, in line with Puppet modules. Configuration modules are aggregated and structured in bill of configuration modules (BOCM), as illustrated by the OpenStack Puppet module's dependency tree of listing 2. A work plan is equivalent to parameterizing and aggregating configuration modules to individual applications (cf. listing 1).

TABLE V. REVISED MAPPING OF PRODUCTION CONCEPTS

Prod. of Physical Products	Prod. of SaaS Products
Material	*Configuration information*
no related concept	*Configuration module*
Bill of materials	*Bill of configuration modules*, preliminary service, IT service
Work plan	*Parametrized and aggregated configuration modules*
Equipment	Resources

Added or changed concepts are written in italics.

In physical goods production, parts such as screws, but also complexer parts, are standardized, making their reuse in production and in engineering with computer aided design software possible. Assembling software like this has long been a motivation for software engineers. However, software artifacts are very heterogeneous. This is in part because software engineering is a creative process, while market aspects also play a role. Assembling software based on preexisting modules is possible in ecosystems such as the object-oriented Java-world. However, for operating complex IT landscapes, technology stacks become heterogeneous and assembling them is not as easy.

While service-oriented architectures address this problem on an architectural level, tools like Puppet address this issue at a lower level. Taking again the analogy to physical goods production, a Puppet module wraps around a software application and provides the standard interface that the software did not have before. Puppet does this by providing a common language for configuration, concepts, and a common architecture aligned with the operating systems that Puppet supports.

Component markets for a specific programming language or even a technology stack above operating systems appear unfeasible, as technology stacks may change over time or from organization to organization. But, it can be assumed that the building blocks of a system landscape such as networks, nodes, and operating system provided core concepts such as users, files, packages, services, and others will remain constant for some time.

The next section draws conclusions, and gives an outlook on future work.

V. CONCLUSION

In this paper, we used the analogy between IT service production and physical goods production to better understand Puppet modules we used in a prototype. The prototype demonstrated the feasibility of an automated order-to-cash-process for SaaS products. Puppet modules are adequate to implement provisioning of SaaS products that are represented by bill of materials and that have a degree of complexity that goes beyond that of illustrative examples. However, these modules come with challenges such as an additional complexity layer. Configuration management is a part of IT service management and IT service management has to be embedded in the company's general management framework. The same is true for configuration management tools that have to be integrated into the architecture of an organization's system landscape. We believe that the integration of configuration management tools with ERP systems holds potential. However, in future work, some issues should be addressed. Possible paths of future work are outlined in the paper's final section.

VI. Future work

As this paper presents research in progress, in future work, the mapping to physical goods production will be extended to the operation phase of SaaS products, and also to include SaaS products that share resources.

Other paths of future research include the following. As only Puppet was analyzed in this case study and other popular configuration management tools such as Cfengine and Chef have similar modularization concepts as Puppet [8], they should still be analyzed to further generalize the findings.

As written in the paper, configuration management tools are not the only means to manage system landscape with multiple nodes. An interesting concept is that of containers that isolate their processes from other processes of the host system. The tool Docker, which was mentioned in subsection II-A, implements the container concept. Assembling containers that contain preconfigured applications is a concept that is similar to physical good production. In future work, it should be investigated whether or not this is a viable alternative to using configuration modules.

Also, future work should follow the proposed concept by Meyer et al. [12]. They propose an automated quality assurance service for configuration modules, which addresses the *complexity-challenge* of configuration modules that is more prominent than in variant production of physical goods. All possible combinations of configurations should be tested because this would reduce problems when users apply a module in their specific environment. However, this is unfeasible as an application with x layers and y alternatives per layer already leads to y^x configuration alternatives. It would therefore be useful to have means to reduce the space of all possible configuration alternatives to a subset of technically and logically possible alternatives. In this reduced space, clusters could be identified, and, for each cluster, a representative candidate could be identified and afterwards tested. Reducing the number of candidates and finding the mentioned candidates is necessary in order to perform the testing with an acceptable amount of time and resources. The fully formalized configuration information of a configuration tool-enabled system landscape would be a data source that could be exploited for this approach.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] S. M. Walter, T. Böhmann, and H. Krcmar, "Industrialization of IT - Foundations, Features, and Characteristics of a Trend (original German title: Industrialisierung der IT - Grundlagen, Merkmale und Ausprägungen eines Trends)," *HMD – Praxis der Wirtschaftsinformatik*, vol. 256, 2007.

[3] J. Becker, J. Poeppelbuss, D. Venker, and L. Schwarze, "Industrialization of IT Services: Application of Industrial Concepts and their Implications from the Perspective of IT Service Providers (original German title: Industrialisierung von IT-Dienstleistungen: Anwendung industrieller Konzepte und deren Auswirkungen aus Sicht von IT-Dienstleistern)," in *10. Internationale Tagung Wirtschaftsinformatik (WI2011)*, A. Bernstein and G. Schwalbe, Eds. AIS Electronic Library, 2011, pp. 345–354.

[4] S. Rance, *ITIL – Service Transition*. The Stationery Office, 2011.

[5] J. Hintsch, H. Schrödl, H.-J. Scheruhn, and K. Turowski, "Industrialization in Cloud Computing with Enterprise Systems: Order-to-Cash Automation for SaaS Products," in *Proceedings der 12. Internationalen Tagung Wirtschaftsinformatik (WI 2015)*, O. Thomas and F. Teuteberg, Eds. http://wi2015.uni-osnabrueck.de, 2015, pp. 61–75.

[6] E. Anderson, L.-l. Lam, C. Eschinger, S. Cournoyer, J. M. Correia, L. F. Wurster, R. Contu, F. Biscotti, V. K. Liu, T. Eid, C. Pang, H. H. Swinehart, M. Yeates, G. Petri, and W. Bell, "Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 4Q12 Update," Website, 2013, accessed: 31.07.2014. [Online]. Available: https://www.gartner.com/doc/2332215/forecast-overview-public-cloud-services

[7] D. Spinellis, "Don't Install Software by Hand," *IEEE Software*, vol. 29, no. 4, pp. 86–87, 2012.

[8] T. Delaet, W. Joosen, and B. Van Brabant, "A Survey of System Configuration Tools," in *24th Large Installation System Administration conference (LISA'10)*, R. van Drunen, Ed. USENIX, 2010.

[9] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, and T. Spatzier, "Integrating Configuration Management with Model-driven Cloud Management based on TOSCA," in *3rd International Conference on Cloud Computing and Service Science (CLOSER)*. SciTePress, 2013, pp. 437–446.

[10] A. Schaefer, M. Reichenbach, and D. Fey, "Continuous Integration and Automation for Devops," in *IAENG Transactions on Engineering Technologies*, ser. Lecture Notes in Electrical Engineering, H. K. Kim, S.-I. Ao, and B. B. Rieger, Eds. Springer, 2013, vol. 170, pp. 345–358.

[11] C. Magherusan-Stanciu, A. Sebestyen-Pal, E. Cebuc, G. Sebestyen-Pal, and V. Dadarlat, "Grid System Installation, Management and Monitoring Application," in *10th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2011, pp. 25–32.

[12] S. Meyer, P. Healy, T. Lynn, and J. Morrison, "Quality Assurance for Open Source Software Configuration Management," in *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2013, pp. 454–461.

[13] J. Wettinger, U. Breitenbcher, and F. Leymann, "Compensation-Based vs. Convergent Deployment Automation for Services Operated in the Cloud," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, X. Franch, A. Ghose, G. Lewis, and S. Bhiri, Eds. Springer, 2014, vol. 8831, pp. 336–350.

[14] J. Fischer, R. Knepper, M. Standish, C. A. Stewart, R. Alvord, D. Lifka, B. Hallock, and V. Hazlewood, "Methods For Creating XSEDE Compatible Clusters," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, ser. XSEDE '14. ACM, 2014, pp. 74:1–74:5.

[15] D. Pop, M. Neagul, and D. Petcu, "On Cloud Deployment of Digital Preservation Environments," in *IEEE/ACM Joint Conference on Digital Libraries (JCDL)*. IEEE, 2014, pp. 443–444.

[16] Puppet Labs, "Puppet 3.7 Reference Manual," 2015, accessed: 10.03.2015. [Online]. Available: https://docs.puppetlabs.com/puppet/latest/reference

[17] T. Dybå and T. Dingsøyr, "What Do We Know about Agile Software Development?" *Software, IEEE*, vol. 26, no. 5, pp. 6–9, 2009.

[18] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston, USA: Pearson Education, 2010.

[19] J. D. Novak and A. J. Cañas, "The Theory Underlying Concept Maps and How to Construct and Use Them," Florida Institute for Human and Machine Cognition, http://bit.ly/198T9h0, Tech. Rep. IHMC CmapTools 2006-01 Rev 01-2008, 2008.

[20] K. M. Eisenhardt, "Building Theories from Case Study Research," *Academy of Management Review*, vol. 14, no. 4, pp. 532–550, 1989.

[21] R. K. Yin, *Case Study Research: Design and Methods*, ser. Applied social research methods series. Los Angeles, USA: SAGE, 2009.

[22] E. Gutenberg, *Fundamentals of Business Economics: Production. Volume 1 (original German title: Grundlagen der Betriebswirtschaftslehre: Die Produktion. Erster Band.)*, ser. Enzyklopädie der Rechts- und Staatswissenschaft. Springer, 1983.

[23] R. Zarnekow, *Production Management of IT Services: Fundamentals, Functions, and Processes (original German title: Produktionsmanagement von IT-Dienstleistungen: Grundlagen, Aufgaben und Prozesse)*, H. Österle, R. Winter, and B. W., Eds. Heidelberg: Springer, 2007.