A Configuration Crawler for Virtual Appliances in Compute Clouds

Michael Menzel, Markus Klems, Hoàng Anh Lê, Stefan Tai
Karlsruhe Institute of Technology (KIT),
Institute for Applied Informatics and Formal Description Methods (AIFB),
Karlsruhe, Germany
{menzel,klems,tai}@kit.edu
hoang.le@student.kit.edu

***Abstract*—Compute clouds are pools of virtual machines that are shared in a multi-tenant environment by multiple users. The virtual machine images are stored in one or more repositories and are pre-configured with an operating system. Users of the compute cloud can upload their own images or install and configure additional software on top of existing basic virtual machines. Today, the Amazon Elastic Compute Cloud (EC2) counts more than 35,000 publicly available virtual machine images. We observe, however, that the meta-data that describes the virtual machine images is of poor quality and does not cover vital information such as operating system configurations or software package installations. The sprawl of poorly documented virtual machine images poses a hurdle to sharing and re-use among members of the compute cloud community. We present a method that allows collecting software-related meta-data in compute clouds through appliance introspection. Moreover, we show how applications in the domains of selection and configuration management benefit from rich meta-data and interact with the method. The method has been implemented as an automated tool, the crawler, that collects configuration data of virtual machine images in public compute clouds and evaluated our approach by crawling Amazon EC2.**

***Keywords*-Compute Cloud; Virtual Machine Image; Software Libraries; Configuration Management; System Management**

I. Introduction

A compute cloud is a pool of virtual machines that is shared in a multi-tenant environment by multiple users. Each user can manage virtual machines in an isolated part of the compute cloud via Web service calls over the Internet. Compute cloud users can start one or more virtual machine instances from a virtual machine image. The images are stored in a repository and are pre-configured with an operating system. A cloud user can start a virtual machine instance, install and configure necessary software packages, and save the instance back to the repository as a (new) virtual machine image. Over time, more and more images are created and shared by users. Today, the Amazon Elastic Compute Cloud (EC2) counts more than 30,000 virtual machine images in different image repositories (Amazon Machine Images (AMIs) with type "machine") . We observe, however, that the meta-data that describes the virtual machine images is of poor quality and does not cover vital information, and particularly lacks information about software package installations.

Virtual machine (VM) images that have been configured for a specific purpose are called *virtual appliances* [1], for example Web server appliances or database server appliances [2]. Compute cloud users face a variety of options when settling for an adequate virtual machine to deploy their software. Virtual machine setups range from basic virtual machine images that only contain an operating system, to virtual appliances containing complex, fully-fledged software stacks. While basic VM images offer more customizability, however, imply more effort, fully fledged virtual appliances tend to restrain modifications, but ease and speed up deployments.

We believe that the sprawl of poorly documented VM images poses a hurdle to sharing and re-using images. Without proper documentation of image configurations, a cloud user cannot judge if an image fits his requirements, nor is it possible to compare images with one another. With rich meta-data cloud users can base evaluations of images on collected meta-data within a selection process. In an adaptive approach the meta-data collection method allows to influence and customize how and what vital information shall be collected.

Additionally, collected meta-data on software libraries allows to derive a configuration description of a VM image that allows to either rebuild an image, or transfer and reuse parts of a configuration to a new VM image. Thus, a description of the ingredients of an image becomes important to cloud users and providers.

Our paper proposes an adaptable method with the following features for collecting and using virtual machine image meta-data: (i) a discovery mechanism, (ii) a crawler, (iii) and a configuration data model. Furthermore, our paper shows applications that benefit and interact with the crawler method and collected meta-data.

II. Virtual Appliances

A virtual appliance is a virtual machine image that has been optimally configured for a particular purpose, e.g., database server appliance. The concept of virtual appliances,

hence, depends on the existence of an platform that allows uploading and sharing virtual machine images and on the existence of individuals or organizations that provide the service of configuring and packaging virtual appliances. The platform for sharing virtual machine images is today fulfilled by compute clouds, such as Amazon Elastic Compute Cloud (EC2) or Rackspace Cloud Servers. Currently, there is no widely accepted standard format for a virtual machine image. For this reason, interoperability between compute clouds is limited.

A. Communities and Markets

We observe different models of virtual machine image provision in compute clouds: (1) centralized packaging and management of bare operating system images (e.g., Rackspace), (2) centralized provision of a wide range of virtual machine images, from bare operating system to complex software stacks (e.g., GoGrid), and (3) decentralized provision (e.g., Amazaon EC2). We can further observe that in decentralized provision models many individuals as well as profit and non-profit organizations provide services to package virtual machine images and upload them to repositories of compute Clouds where they can be utilized by Cloud users. Individuals such as Eric Hammond [3], who assembles a variety of Ubuntu Linux AMIs, and organizations like Bitnami [4], rPath [5], and Turnkey Linux [6], specialized on building reliable AMIs with complex software stacks, create an abundance of instantly deployable virtual machine images and, in particular, virtual appliances. Commercial software vendors, including IBM and Oracle, package AMIs and either release them for free or with commercial licenses. Amazon's API comprises fee or license payment services for AMIs (charged by hour or for long-term subscriptions). The Cloud Market [7] holds and maintains a database of 31,270 AMIs.

In a first attempt to evaluate the zoo of virtual machine images on Amazon EC2, we collect and analyze the ownership of publicly accessible AMIs. The results are aggregated in Table I. We focus on the AMI repository that is associated with the EC2 us-east-1 region which is the oldest region and counts the largest number of AMIs (see Figure 1). For comparison, we state AMI ownership numbers of all (currently 7) regions. However, we believe that the global AMI ownership number is less meaningful because some owners just copy the same AMI into different regions. Each copied AMI counts as an individual AMI although it brings no additional value except from being available in a different region.

The statistics that we calculated show that Bitnami is by far the largest AMI owner with more than 1600 AMIs which accounts for nearly 15% of all AMIs in this region. Bitnami is followed by rPath, a company that offers deployment and management software, and CloVR, a company that specializes on genome sequencing applications. Ranks four and five are held by Canonical and Alestic, the major maintainers of EC2 Ubuntu AMIs. In the us-east-1 region, there are 1,796 different owners who own a total number of 10,265 AMIs. In all regions, there are 2,207 different owners who own a total number of 32,944 AMIs.

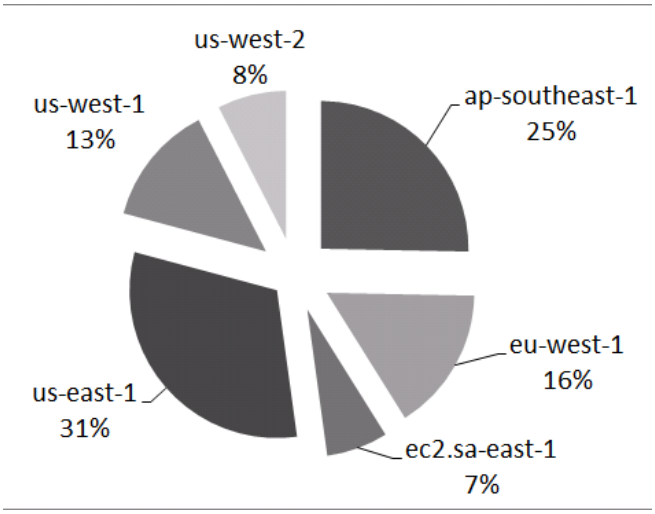
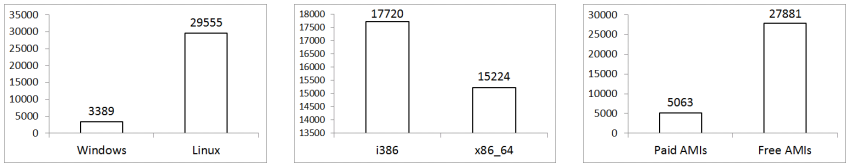


Figure 1. Number of AMIs per Amazon EC2 Region.

Rank	Owner	images	% images
1	Bitnami	1648	14.92%
2	rPath	868	7.86%
3	CloVR	655	5.93%
4	Canonical	584	5.29%
5	Alestic	397	3.60%
6	AWS Marketplace	292	2.64%
7	Turnkey Linux	210	1.90%
8	Zeus Technology	200	1.81%
9	RightScale	189	1.71%
10	CloudSwitch	181	1.64%
11	Scalr	161	1.46%
12	Elastic Bamboo	153	1.39%
13	Amazon-1	110	1.00%
14	Red Hat	100	0.91%
15	Openbravo	98	0.89%
16	Amazon (Windows)	97	0.88%
17	EnterpriseDB	60	0.54%
18	CohesiveFT	58	0.53%
19	Ensembl	58	0.53%
20	Jumpbox	53	0.48%

Table I
THE TOP 20 AMI OWNERS BY #IMAGES IN US-EAST-1 REGION.

Most of the EC2 AMIs are Linux-based and free of commercial software licenses (Figure 2). In the Amazon EC2 region us-east-1, approximately 9,600 out of 11,000 AMIs have no commercial software license. Only 5 out of the top 20 AMI owners offer paid AMIs with commercial software, namely AWS Marketplace (100% commercial AMIs), Turnkey Linux (59%), Zeus Technology (100%), EnterpriseDB (100%), and CohesiveFT (74%). The use of free and open source software likely supports sharing and re-use of virtual machine images.



(a) Operating System (b) Architecture (c) Software License

Figure 2. Attributes of AMIs in all regions (global).

Attribute	AWS	RS	GG
ID	✓	✓	✓
Name	✓	✓	✓
Timestamp Created	×	✓	✓
Timestamp Updated	×	✓	✓
Status Accessibility (Public, Non-Public)	✓	×	✓
Status Availablity (Active, Saving, Deleting)	✓	✓	✓
Status Progress (Modification Progress)	×	✓	×
Operating System	×	×	✓
Price	✓	×	✓
Geo Location	✓	×	×
File Location	✓	×	✓
Owner	✓	×	✓
Architecture	✓	×	×
Category/Type	×	×	✓
Description Text	✓	×	✓
Hypervisor	✓	×	×
Disk Volumes	✓	×	×

Table II
META-DATA ATTRIBUTES ACCESSIBLE VIA COMPUTE CLOUD APIS.

B. Virtual Machine Image Meta-Data

Compute cloud providers usually offer mechanisms to annotate virtual machine images with meta-data and make this information accessible as a service. However, information on operating system configurations and software package installations of virtual machine images is not provided by any of the large compute cloud providers. In decentralized provision models, virtual machine image repositories of compute cloud providers are filled by a multitude of users who form a community. The contributors do not properly describe their images with meta-data – neither in the compute cloud repository nor on their own websites.

We examine the meta-data attributes exposed through service APIs of different compute cloud providers, namely Amazon Web Services (AWS), Rackspace (RS) and GoGrid (GG). Table II shows the different meta-data attributes used by each of the three providers.

C. Virtual Appliance Introspection

In analogy to Virtual Machine Introspection (VMI) [8], we define *Virtual Appliance Introspection* (VAI) as a capability to obtain knowledge of operating system configurations and software package installations of a virtual machine image that serves as an appliance. In the following section, we show a method in support of basic introspection capabilities. Our method is used to collect meta-data information of virtual appliances and store the collected information in a central database for data analytics and configuration management.

III. THE CONFIGURATION CRAWLER METHOD

In the following sections, we describe the components of our automated, adaptable configuration crawler method for VM images and, in particular, appliances with complex configurations. The method guides through the components subsequently and applies the steps described in each component. The method is divided into three major components that work in concert and support different tasks. The discoverer component prepares task queue of VM images introspection jobs for later parallel data collection. The crawler distributes the tasks to multiple crawler instances that collect the requested meta-data from inside a launched virtal machine and upload their results to the data management component. In the data management component crawling results are structured in a model and persisted in a database management system which provides interfaces to make use of the results in applications (see IV).

A. The Discoverer

The *Discoverer* takes a list of *repositories* and *filters* as inputs. A repository is a uniquely identified location where virtual machine meta-data of a compute cloud provider can be accessed. For example, Amazon EC2 offers an API that allows fetching a list of all AMIs from EC2 repositories in different regions. The Discoverer outputs a *discovery list* for each repository, i.e. a list of ids of virtual machines that have been found in the repository and match the filter criteria.

Filters are provider-specific or even repository-specific since different compute clouds likely provide differently named and detailed meta-data. There are two types of negative filters: (1) user-defined K.O. criteria, and (2) user-defined block-lists. K.O. criteria remove unwanted VM images from the discovery list, such as images with commercial software licenses. Then the ids in the block-list are matched with the discover list. Each match is removed from the discovery list, for example, ids of images that have already been introspected. Users can also define a positive filter, i.e. requirements that need to be fulfilled, for example, a specific operating system.

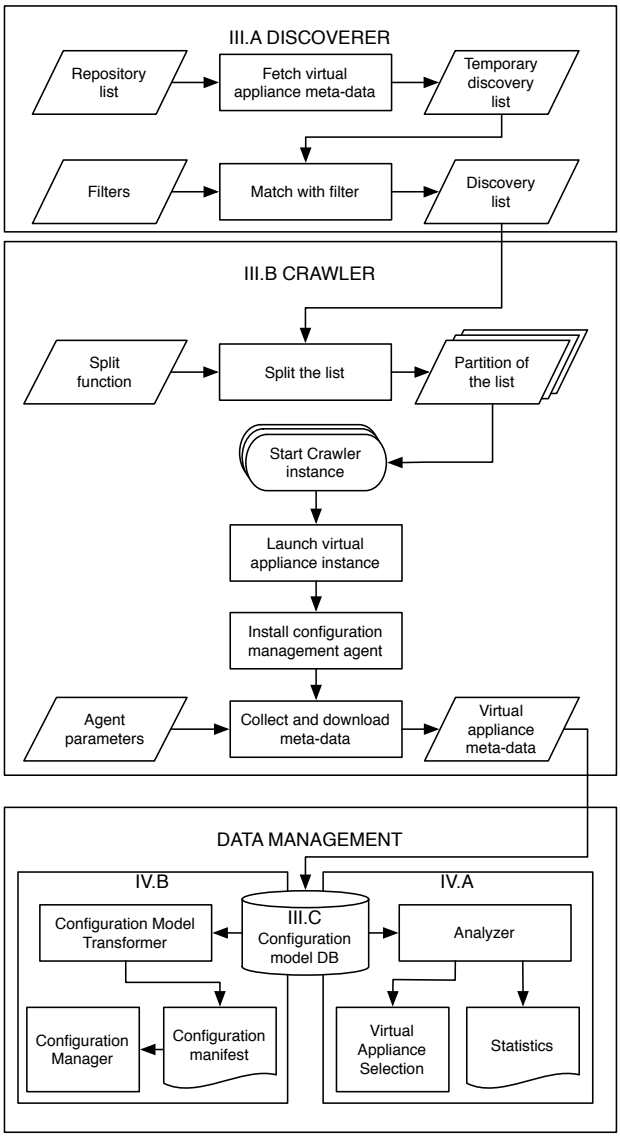


Figure 3. The Configuration Crawler Method.

B. The Crawler

The *Crawler* collects meta-data from a subset of the virtual appliances represented by VM image ids in the discovery list. Since the discovery list can be long, the user might want to narrow down the list. We suggest two approaches to reduce the size of the discovery list. The user either selects the desired images by their id or description, or the user inputs the desired coverage, either as an absolute number of VM images or as a fraction of the discovery list between 0 and 100%. In the latter case, simple random sampling is used to select the images.

Additionally, a stated cost budget restrains the total number of introspections to a maximum amount of costs. The cost calculation needs to align with the actual costs generated at the Cloud provider with cost models differing in means of preciseness or billing subjects causing different costs per introspection, e.g., Amazon bills by the hour. While the cost budget would not be exceeded by an additional virtual machine instantiation the crawler proceeds with processing the discovery list.

The discovery list can be processed in parallel by multiple Crawler instances. If the user provides a split function to the Crawler, the discovery list is segmented into multiple discovery list partitions. Each discovery list partition is processed by a separate Crawler instance. A Crawler instance can be a separate thread or process on the same server, or a separate (virtual) server.

Each Crawler instance runs the following sequential process for each VM image in the partition of the discovery list:

1) Launch an instance of the VM image (if possible)
2) Upload and install a configuration management agent on the running instance
3) Execute the agent with parameters, collect the requested meta-data and download the meta-data to a central database

Step 1 requires an API to a compute service in order to launch an instance of a VM image. The second step requires a channel to upload the agent software onto the running virtual machine instance. For Linux operating system instances, the agent software bundle is uploaded with ssh. Next, an installer script checks the operating system (Linux distribution) and decides which agent software to install. Of course, existing agents of configuration management software packages could be used in this procedure, e.g., Puppet's facter[1]. In the third step, the agent is executed and collects meta-data. All collected data is retrieved from the agent by the crawler which, again, uploads and stores it in a central database. For the third step a user can provide agent parameters that determine (1) what meta-data to collect or (2) how to collect meta-data. By giving a list of attributes a user can define what data to collect from the instantiated virtual machines. A user can, furthermore, inject a custom script to be executed by the agent that collects (additional) meta-data. Virtual machine instances contain different operating systems and execution environments, and, thus, custom scripts must be provided in multiple versions or made platform-independent.

The crawler is kept as simple as possible to reduce failures that could emerge due to incompatibility of the agent software running on different operating systems. Therefore, the data that is uploaded into the central database has the native data format (e.g., JSON or XML) and structure used by the agent. However, in a subsequent step, data needs to be transformed and uploaded to a data management system that allows complex queries and analytics. Since every agent

[1]http://projects.puppetlabs.com/projects/facter

produces different output formats we refer to data integration approaches for transformation of the data into the proposed data model.

C. The Configuration Data Model

As a basis for future applications the following data model shall provide a structured, queriable source of information about introspected virtual appliance configurations. In our model, depicted in Figure 4, we consider the superset of attributes and information available through both, the compute cloud APIs and our crawler (see Table II). While Virtual Machine Image objects hold meta-data directly connected with a virtual machine, Software and ProgrammingLanguage objects hold information on an appliance's configuration retrieved with the crawler. Both, Software and ProgrammingLanguage, contain an AttributeMap attribute that allows to store arbitrary additional information as key-value pairs. Also, all data is typed as String to support all sorts of data formats as the attribute "Version", for instance, might be a number "3.0" or a non-numeric text "3.0-milestone".

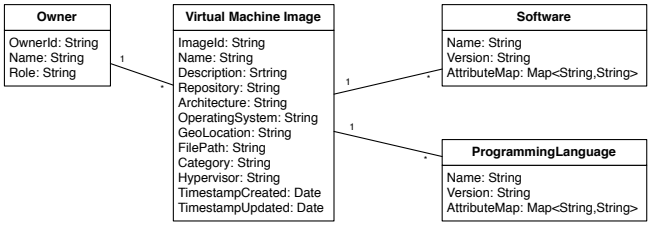


Figure 4. The Configuration Data Model.

IV. Applications

The configuration models that have been collected by the Crawler and are stored in a central configuration model database are valuable input for a broad spectrum of software development and system administration applications. On the other side, applications can interact and apply the configuration crawler method. We address multiple application scenarios and show how the data that has been collected with our method and the crawler as an instrument adds value to the public compute cloud communities.

A. Configuration Analytics

The data collected by the Crawler can be used for configuration analytics and search. We describe how virtual machine candidates can be identified when filtering by requirements based on analytics. Furthermore, we give insights into Amazon's virtual machine database fed by the community. Therefore, we conducted an analysis on attribute clusters in publicly available AMIs based on recent crawling results.

Table III
CloudGenius And Additional Requirement Types

Value Type	Req. Type	Boolean Expression
Numerical	Max	$\chi(\alpha) < v_r$
Numerical	Min	$\chi(\alpha) > v_r$
Non-numerical	Equals	$\chi(\beta) = s$
Non-numerical	OneOf	$\chi(\beta) \in S$
Non-numerical	Includes	$\{\chi(\beta)_1, ..., \chi(\beta)_n\} \ni s$

1) Virtual Appliance Selection.: With specific details, a list of appliance candidates can be identified that are the best fit for given requirements. In particular, the problem space of virtual appliance selection [9] can be addressed and improved with very detailed requirement definitions. This potentially results in more accurate filtering and higher qualified virtual appliance candidates. We discuss how configuration models that have been collected with the Crawler can be used in the CloudGenius framework [9]. Also, how the CloudGenius framework can profit from employing the crawler method.

In the CloudGenius framework, an essential step is the selection of virtual appliances and images that suit an engineer's preferences in combination with a compute service. To filter out ineligible appliances according to their configuration, we propose to include configuration meta-data in the framework's requirement definition phase (see requirement types in Table III). When integrating the collected configuration meta-data into the CloudGenius framework, additional information on the images can be accessed to set constraints, e.g., Java version ≥ 5, Region $=$ "US".

With the configuration models, an additional requirement type needs to be introduced to the CloudGenius framework. When requiring a certain software package or programming language to be pre-setup in the appliance, requirement constraints must be defined with an "$\ni$" operator. In detail, the operator is for non-numerical value types. The operator queries the existence of a value in a set of values related to the appliance: $\{\chi(\beta)_1, ..., \chi(\beta)_n\} \ni s$.

For complex requirement definitions, analysis capabilities should rather be based on a declarative approach. The requirement definitions are imported into a database system and can be queried, for example with SQL.

The CloudGenius framework can also actively employ the crawler method for collecting certain information benefitial in a virtual machine image selection. Therefore, within a migration process an engineer must provide the input for the Crawler, the mandatory repository list and split function to consider and optionally budget, filter functions and agent parameters. A default filter function might be provided by the framework. With agent parameters the engineer can actively determine which meta-data to collect that is vital for her virtual machine decision.

2) Basic AMI Analysis.: We use the Crawler to collect configurations from 869 out of 981 Ubuntu AMIs provided

Ubuntu package name	Installation count
perl	863
python	710
python2.6	247
python2.7	263
ruby1.8	399
openjdk-6-jdk	0
build-essential	153
openssh-server	869
vim-tiny	843
vim	502
subversion	23
git-core	8

Table IV
EXAMPLE: SOFTWARE PACKAGE COUNT (CANONICAL AND ALESTIC.COM).

	AMI Owner	AMI count
Topic 0	Canonical	202 (38.5%)
Topic 1	Canonical	322 (61.5%)
Topic 0	Bitnami	159 (89.7%)
Topic 1	Bitnami	38 (19.3%)

Table V
LATENT DIRICHLET ALLOCATION (LDA) REPORT FOR 2 TOPICS AND 100 MINTOKEN.

by Alestic.com and Canonical. The remaining 112 AMI could not be accessed by the Crawler due to ssh connections failures (using either the root or the ubuntu user). Table IV shows examples of software package count queries.

3) Document Cluster Analysis.: We use the Latent Dirichlet Allocation (LDA) [10], a document clustering technique, as a means to discover topics of similar virtual machine images. For instance, a Web server virtual appliance would belong into a different topic as a database machine image, which again, would belong into a different topic as a bare Ubuntu machine image. Using LDA, an Ubuntu-based Web server appliance would be modeled as a mixture of different topics, involving Ubuntu packages and Web server packages.

The LDA is implemented with the LingPipe Java library[2]. We parse a collection of configuration models that the Crawler collected from the AMI owners Canonical and Bitnami. The *number of topics* must be specified, as well as the *minimum token count* (mintoken), and other input parameters. We experiment with both parameters, starting with 2 categories and 100 mintoken. The LDA outputs a report summarized in Table V. The Canonical AMIs are nearly evenly distributed across both categories whereas the Bitnami AMIs are much stronger associated with Topic 0, which therefore could be labeled "Bitnami Topic". The results indicate that the Canonical Ubuntu images are less specialized than the Bitnami images and appliances (as expected).

[2]http://alias-i.com/lingpipe

B. Configuration Management

The configuration models that have been collected by the Crawler can be transformed into input files for a declarative configuration management tool, such as Puppet [11]. Listing 1 shows an excerpt from a configuration model that the crawler has collected. The configuration model is transformed into a Puppet manifest as shown in Listing 2.

Listing 1. Crawled configuration model of an Ubuntu 11.04 Natty instance

```
...
"python2.7": {
  "version": "2.7.1-5ubuntu2",
  "description": "An interactive high-level
      object-oriented language (version 2.7) "
},
"python2.7-minimal": {
  "version": "2.7.1-5ubuntu2",
  "description": "A minimal subset of the
      Python language (version 2.7) "
},
"readline-common": {
  "version": "6.2-0ubuntu1",
  "description": "GNU readline and history
      libraries, common files "
},
"rsync": {
  "version": "3.0.7-2ubuntu3",
  "description": "fast remote file copy program
       (like rcp) "
},
...
```

Listing 2. Generated Puppet manifest file

```
...
package { 'python2.7':
  ensure   => '2.7.1-5ubuntu2',
}
package { 'python2.7-minimal':
  ensure   => '2.7.1-5ubuntu2',
}
package { 'readline-common':
  ensure => '6.2-0ubuntu1',
}
package{ 'rsync':
  ensure => '3.0.7-2ubuntu3',
}
...
```

The Puppet manifest can be applied to a different virtual machine image and thus transfer (parts of) the configuration from a source image to a target image. This could be useful for migrating the configuration of an image from one compute cloud into another one, for re-creating similar images, etc. From our experience, the process works fairly well but cannot be fully automated, as there are always some incompatibilities between package names and package version names, particularly between different operating systems. For example, transferring the configuration from a base Ubuntu 11.04 AMI (ami-87c31aee) to another Ubuntu 11.04 AMI with Ruby Passenger Rails3 (ami-bfb473d6) resulted in 10 errors (package could not be installed) and in 33 package updates while 339 packages remained untouched.

We believe that a semi-automated process could provide useful assistance to system administrators. For example, a system administrator could migrate the python and ruby packages and libs from one image to another one (or more).

V. Implementation

We have implemented the configuration crawler method as a software tool for crawling the Amazon Elastic Compute Cloud (EC2) [12]. Ruby scripts implement the Discoverer and Crawler components, and MongoDB servers as the configuration model database for JSON configuration documents.

The Discoverer accesses the EC2 API and retrieves a list of all EC2 region endpoints (AMI repositories), i.e., "ec2.us-east-1.amazonaws.com", "ec2.us-west-1.amazonaws.com", etc. In a next step, the Discoverer accesses the EC2 API to query the meta-data of all AMIs that match given filter criteria. We excluded all paid AMIs, since some AMIs charge substantial monthly fees billed with the first instance launch. We further restrict the list to Linux operating system AMIs by the most prominent AMI owners (see Table I). The Discoverer saves the reduced AMIs into a text file, named discovery list.

The Crawler requires a few input files: (1) the discovery list, (2) a filter definition file, and (3) a split function script. Given the input, the Crawler implementation runs a fully automated procedure and fills the database with configuration documents.

The Crawler starts multiple Crawler instances in parallel, each of which launches an EC2 instance. Each Crawler instance connects via ssh to its EC2 instance and uploads the agent software. A shell script installer downloads and installs a suitable agent. For introspection of all Linux machines, a shell script serves as basic agent software. If the operating system is suitable, the installer furthermore downloads and installs an Ohai[3] agent to collect additional operating system and software package information. After the shell script and Ohai agents have completed their introspection tasks, the Crawler downloads all collected data from the instance and terminates it. The collected meta-data is then uploaded by the Crawler to Amazon S3 and later transferred into the MongoDB database.

We also provide a Web frontend to the virtual machine image meta-data we collected with our crawler implementation [13]. The Web frontend lists available images and allows text searches and filtering by software libraries. The full configuration of a virtual machine image is displayed as a list of software libraries and version numbers.

VI. Related Work

Virtual Machine Introspection (VMI) [8] is a topic of security research. Different from our approach, the VMI framework can directly access the Virtual Machine Monitor (VMM). We assume that the compute cloud provider does not allow access to the VMM. Moreover, different from VMI, we do not attempt to discover security holes. Instead, we want to deliver information on software installations and configurations of virtual appliances. However, the technical approach that we used for introspection could easily be used for other purposes, as well.

Filepp et al. [14] have developed an virtual appliance configuration approach that installs missing software on appliances and finds an appliance which requires lowest effort. The approach uses a repository of appliance configurations that has been build based on the Galapagos system [15] that only detects few well-known enterprise software packages [16]. The notion of agents and automated crawling with the purpose to provide a configuration model database for a variety of applications is missing in the approach. Dastjerdi et al. [17] propose a file format to describe virtual units that comprise virtual machine images or appliances. Compute cloud providers are able to advertise machine images in their repositories in the format and, thus, allow a requirements matchmaking. The approach, however, requires providers or contributors to maintain the meta-data.

In contrast to our approach Lutterkort and McLoughlin [18] see meta-data of virtual machine images to be bundled together with its files. Marginal and distributed meta-data proposed by the approach makes it very costly to conduct an analysis.

Other approaches in the field of meta-data of virtual machine images have addressed somehow related issues. Reimer et al. [19] have recognized a sprawl of virtual machine images and introduced a semantic-based approach to describe the contents of an image. However, contents of an image are described by hash values of files tying the approach to detection of contents by identical files only. Also, this approach requires the administrator of images to adopt a certain format. Similarily, Kecskemeti et al. [20] propose an approach to decompose virtual appliances on a file level to optimize storage and transfer over networks, however, lack an identification of software actually included in an appliance. Wilson and Matthew [21] create virtual appliances from multiple software packages with configuration management technologies but void keeping the configuration of appliances in a meta-data database. Liu [22] shows how configuration meta-data allows an automated configuration of appliances, but aims at configuration settings only and misses support for rebuilding virtual appliances as a reason of trust or migration.

VII. Conclusions & Future Work

Building and sharing software stacks as VM images in compute clouds is gaining popularity and has already lead to an abundance of images that are available in public compute clouds. We find that the meta-data that describes the publicly

[3]http://wiki.opscode.com/display/chef/Ohai

available images lacks important information that software developers and system administrators need for selecting the appliance that fits their requirements.

Our work presents a method to automate the process of collecting additional meta-data about VM image configurations. The method can be applied in both private or public compute clouds. We evaluate the method by crawling public VM images in Amazon's Elastic Compute Cloud (EC2), however, plan to extend our work to a broader spectrum of compute cloud providers. The data that we collect with our Crawler is stored in a database and can be used by applications that we envision to benefit from the configuration model data and adaptability of the method. These applications include VM image selection and search, as well as statistic analysis (for example for market research or appliance categorization). Moreover, generating configuration manifests from models in the configuration model database would allow cloning or migrating images.

The presented method is open to extensions and more complex applications. With extensions to the configuration data model, more detailed analytics could be conducted, e.g., geo-location of image repositories would allow geographic querying and matching methods. Moreover, transforming version attributes from text to numbers, in the configuration data model or during analytics, could enable advanced querying possibilities. With further research and development of configuration management agents, a wider operating system support for extended configuration discovery might be possible. Advanced agents might be able to also include configuration settings of software packages, e.g., configuration files in "/etc" and other file paths, into the introspection method.

Furthermore, future work aims at introducing an extended logic for cost budget limitations and parametrization into the crawler method. Further parametrization helps to restrict a crawling process to the collection of particular meta-data attributes from a pre-ordered list of virtual appliances to minimize the introspection effort and adapt the method to specific applications. With cost budgets and parametrization an optimization logic must consider cloud service price models and align virtual machine runtimes, e.g., collect as much of the requested meta-data as possible, shutdown a machine introspection right before an additionally billed hour and repair broken results if any.

We plan to extend our Crawler implementation to support multiple public compute services and develop the presented applications to more advanced software prototypes. Moreover, we publish our Crawler implementation as open source software, hoping that the open source community uses it as a basis for future research and development. With the presented method and implementation at hand, we expect the emergence and evolution of more compute cloud crawlers along with public configuration model databases.

Acknowledgments.: The work presented in this paper was partially funded by the German Federal Ministry of Education and Research (BMBF) in the context of the Software-Cluster project EMERGENT (software-cluster.org) under grant number "01IC10S01". The authors assume responsibility for the content.

We thank Amazon Web Services for the research grant that supported this work.

REFERENCES

[1] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," in *Proceedings of the Seventeenth Large Installation Systems Administration Conference (LISA 2003)*, October 2003.

[2] A. Aboulnaga, K. Salem, A. A. Soror, U. F. Minhas, P. Kokosielis, and S. Kamath, "Deploying database appliances in the cloud," *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 13–20, 2009.

[3] Eric Hammond, http://www.alestic.com, accessed 25th April 2012. [Online]. Available: http://www.alestic.com

[4] Bitnami, http://www.bitnami.org, accessed 25th April 2012. [Online]. Available: http://www.bitnami.org

[5] rPath, http://www.rpath.com, accessed 25th April 2012. [Online]. Available: http://www.rpath.com

[6] Turnkey Linux, http://www.turnkeylinux.org, accessed 25th April 2012. [Online]. Available: http://www.turnkeylinux.org

[7] The Cloudmarket, http://www.thecloudmarket.com, accessed 25th April 2012. [Online]. Available: http://www.thecloudmarket.com

[8] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *NDSS*, 2003.

[9] M. Menzel and R. Ranjan, "CloudGenius: Decision Support for Web Server Cloud Migration," in *Proceedings of the 21st International Conference on World Wide Web*. New York, NY, USA: ACM, 2012. [Online]. Available: http://arxiv.org/abs/1203.3997

[10] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[11] Puppetlabs, "Puppet," http://projects.puppetlabs.com/projects/puppet, accessed 25th April 2012. [Online]. Available: http://projects.puppetlabs.com/projects/puppet

[12] "The Crawler Implementation," Github Repository. [Online]. Available: https://github.com/myownthemepark/ami-crawler

[13] "The Crawler Ride," Web application, 2012. [Online]. Available: http://crawlerride.appspot.com

[14] R. Filepp, L. Shwartz, C. Ward, R. Kearney, K. Cheng, C. Young, and Y. Ghosheh, "Image selection as a service for cloud computing environments," in *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*, dec. 2010, pp. 1 –8.

[15] K. Magoutis, M. Devarakonda, N. Joukov, and N. G. Vogl, "Galapagos: Model-driven discovery of end-to-end application-storage relationships in distributed systems," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 367 –377, july 2008.

[16] IBM, "Tivoli application dependency discovery manager," http://www-01.ibm.com/software/tivoli/products/taddm/, accessed 25th April 2012.

[17] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya, "An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, IEEE Computer Society. Ieee, 2010, pp. 104–112. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5493487

[18] D. Lutterkort and M. McLoughlin, "Manageable virtual appliances," *Linux Symposium*, 2007. [Online]. Available: http://ols.fedoraproject.org/OLS/Reprints-2007/OLS2007-Proceedings-V1.pdf#page=293

[19] D. Reimer, A. Thomas, G. Ammons, T. Mummert, B. Alpern, and V. Bala, "Opening black boxes: using semantic information to combat virtual machine image sprawl," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '08. New York, NY, USA: ACM, 2008, pp. 111–120. [Online]. Available: http://doi.acm.org/10.1145/1346256.1346272

[20] G. Kecskemeti, G. Terstyánszky, P. Kacsuk, and Z. Németh, "An approach for virtual appliance distribution for service deployment," *Future Generation Comp. Syst.*, vol. 27, no. 3, pp. 280–289, 2011.

[21] M. S. Wilson, "Constructing and managing appliances for cloud deployments from repositories of reusable components," in *Proceedings of the 2009 conference on Hot topics in cloud computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855549

[22] H. Liu, "Rapid application configuration in amazon cloud using configurable virtual appliances," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11. New York, NY, USA: ACM, 2011, pp. 147–154. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982221