Sublimated Configuration of Infrastructure as a Service Deployments
MING: *A Model- and View-Based Approach for Cloud Datacenters*

Ta'id Holmes
Infrastructure Cloud, Deutsche Telekom Technik GmbH, Darmstadt, Germany

Keywords: Cloud, Code Generation, Configuration, Datacenter, Deployment, DSL, IaaS, JuJu, MBE, Metamodel, OpenStack.

Abstract: For establishing the basic cloud service model, a datacenter (DC) needs to deploy an infrastructure as a service (IaaS) solution. The planning, setup, implementation, and operation of DCs – involving hard- and software – comprises multiple activities. At least, software-related aspects such as IaaS deployment can be automated. Yet, in the forefront of an automated installation extensive configurations need to take place. These configurations often relate to the design and characteristics of the respective DC. Using existing deployment technologies, however, information from various aspects are scattered and tangled. For avoiding respective drawbacks and resulting adverse effects, MING (明) – a model-based approach – is presented. It decouples configuration from automated deployment technologies. This way, various further benefits of model-based engineering are leveraged such as separation of concerns through view-based models, platform independent representation of information, and the utilization of existing deployment technologies through code generation.

1 INTRODUCTION

The cloud computing paradigm continues to change the way end-users consume services. At the same time service providers adapt (*cloudify*) software services (cf. (Andrikopoulos et al., 2013)) for profiting from the benefits of cloud computing. Thus, cloud computing penetrates all levels from DCs to end-users. On an infrastructure level cloud computing offers (virtualized) hardware resources and network capabilities as IaaS (cf. (Mell and Grance, 2011)). This impacts the design and deployment of DCs.

For this reason the planning, implementation, and operation of DCs has changed. Above all, cloud DCs distinguish themselves from traditional DCs by having an IaaS solution deployed. The IaaS solution establishes the correspondent service model while managing the DC resources. These resources comprise computational power from central processing units (CPUs), random-access memory (RAM), storage in the form of objects stores or block devices from hard drive disks (HDDs) and solid-state drives (SSDs), and network interfaces. All of these need to be registered and managed by the IaaS solution.

OpenStack [1] is a popular IaaS solution with a big community and support across industries. For easing the installation and deployment of OpenStack various automated deployment technologies and tools exist. Often they are provided from an operating system (OS) vendor as a kind of value proposition.

For realizing installation of bare machines and the deployment of the IaaS solution, currently, various information needs to be aggregated in configurations by experts who are familiar with the technologies. Often the information relates to different aspects and is scattered and tangled and needs to be kept consistent. Changes in the design or characteristics of a DC may impact the configuration fundamentally: e.g., networking, number of availability zones in a DC, or dedication of a certain node to some aggregate.

Ideally, it would be possible to describe the various aspects of a cloud DC (i.e., the hardware, the networking) and its deployment (i.e., the services) conceptually in a domain-specific language (DSL) so that from such information as contained in the respective views the automated DC deployment can take place. While several deployment tools exist (that in fact can all be made use of following the model-based approach) there is no technology agnostic datamodel for specifying a DC deployment that is understood by tools.

Therefore, MING (明), a model-based approach, is proposed: It permits the model- and view-based description of DCs and respective IaaS deployments by

[1] http://openstack.org

Holmes, T.
Sublimated Configuration of Infrastructure as a Service Deployments - MING: A Model- and View-Based Approach for Cloud Datacenters.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 2*, pages 308-313
ISBN: 978-989-758-182-3

means of a platform-independent model (PIM). This way, separation of concerns (SoC) is realized supporting different stakeholders. Also, integration with existing tools is realized using code generation.

The remainder of this paper is structured as follows: The following section presents some further background by explaining tasks when deploying a cloud DC. Prior to presenting the MING approach in Section 4, Section 3 relates to the state of the art. Next, Section 5 presents some details of the current prototype. Section 6 discusses on the benefits, risks, and limitations and Section 7 concludes the paper.

2 DATACENTER DEPLOYMENT

The planning, setup, implementation, and operation of a DC comprises multiple activities involving hard- and software. Prior to focusing on the latter, i.e., the automated software installation and deployment, this section first looks at the structure of DCs.

2.1 Structure of Datacenters

Figure 1 depicts a simple metamodel for DCs (that is also part of the MING metamodel, cf. Figure 2). A DATACENTER comprises one or several AVAILABILITYZONES. These may be fire compartments that are separated from each others. Each AVAILABILITYZONE contains RACKS for mounting equipment such as routers and servers (NODE). A server comprises network interface controllers (NICs), CPU, RAM, and storage in form of HDDs and/or SSDs. Each NIC has a unique media access control (MAC) address. For networking (cf. Layer 3 of the Open Systems Interconnection (OSI) reference model (Zimmermann, 1980)) a NIC will be configured at some stage with an Internet Protocol (Cerf and Khan, 1974) (IP) address, e.g., using Dynamic Host Configuration Protocol (Droms, 1997) (DHCP). Within a NETWORK (i.e., IP4 and NETMASK) a NIC has a particular NETWORKID (e.g, the last byte of the address).

INSTALLATION nodes are used for bootstrapping the DC deployment. Generally it is possible to group servers into different categories: STORAGE nodes contain a large amount of storage capacity while COMPUTE nodes have high computational power. NETWORK nodes may comprise fiber optical NICs for high bit rates. Finally, MANAGEMENT nodes are dedicated for hosting IaaS services (see also Section 2.2).

Intelligent Platform Management Interface (IPMI) may provide an administrative access to the servers through a dedicated network. Besides, all servers may

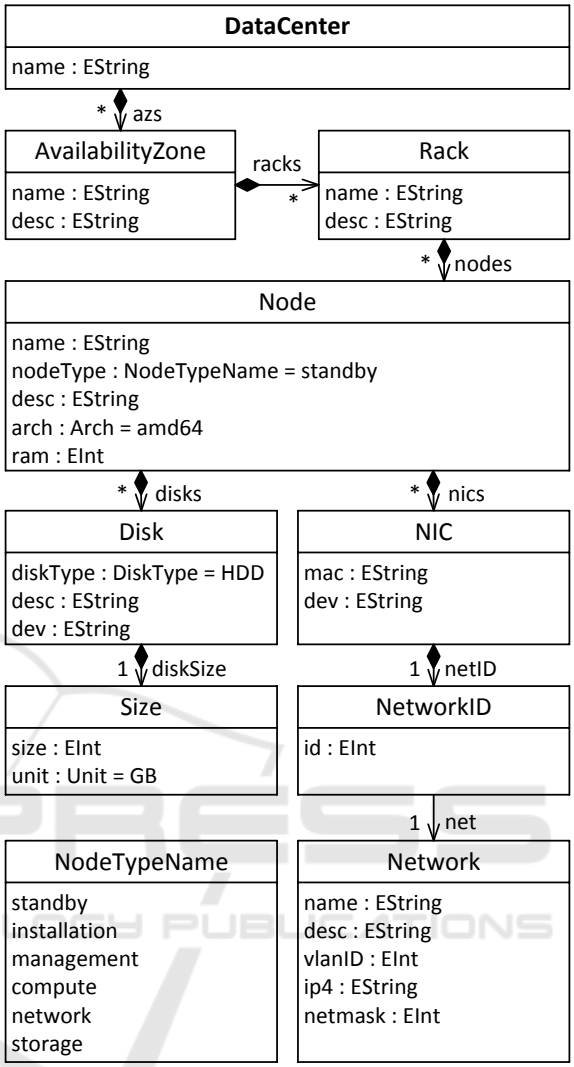

Figure 1: A Datacenter Metamodel.

be part of multiple physical or virtual (VLANID) networks. For example, storage nodes may have a backend network for replication in addition to a frontend network for the data.

2.2 Software Installation

Given a DC with a completed physical setup including networking and cabling, installation of the bare machines can take place (see also Section 3.1). That is, on each node a base OS is installed over the net, e.g., using IPMI and Preboot Execution Environment (Intel Corporation, 1999) (PXE) together with a DHCP server. Yet, some information needs to be collected beforehand and placed at the DHCP server such as the MAC addresses of the NICs. This information may be discovered automatically.

Next, IaaS services can be installed (see also Section 3.2). STORAGE servers may deploy Ceph (Weil et al., 2006), a distributed object store. COMPUTE nodes run OpenStack Compute (Nova). OpenStack Networking (Neutron) is used for NETWORK nodes. Finally, MANAGEMENT nodes host services such as OpenStack Identity Service (Keystone), Nova cloud controller, OpenStack Orchestration (Heat), and OpenStack Dashboard (Horizon).

Having briefly introduced the deployment of cloud DCs, the following section discusses different existing deployment tools for realizing the installation.

3 STATE OF THE ART

Prior to presenting the MING approach let us first have a look at the state of the art. At the end of this section, a current shortcoming of the existing deployment technologies is summarized for positioning the contribution of this paper. As outlined in the previous section there are two distinct phases in the deployment of DCs: bare machine and IaaS installation with some of the tools focusing on the former and others on the latter.

3.1 Bare Machine Installation

The following selection of tools and projects focus on installing a base OS on bare machines (cf. (Chandrasekar and Gibson, 2014) for the evaluation of some frameworks).

Cobbler [2] is a lightweight build and provisioning system for the deployment of physical and virtual machines. Objects and variables are used for configuring the provisioning. These are then applied in templates, e.g., for generating preseed files. This way, i.e., through templates, Cobbler also integrates with Kickstart. Generally, integration with existing tools and configuration management (CM) systems is encouraged. In addition to a command-line interface (CLI) there is a also a web user interface (UI). Cobbler is currently used by Compass and Fuel (see Section 3.2).

FAI [3], with a particular focus on unattended automated installations, builds – as Cobbler – on top of technologies such as DHCP, Trivial File Transfer Protocol (Sollins, 1992) (TFTP), and PXE. Originally focusing on Debian-based distributions, FAI has been adopted for CentOS. It realizes profiles in addition to a class concept that can help to describe complex setups.

Ironic [4] is used for the provisioning of physical machines within OpenStack. Thus, in contrast to the other tools of this category, it is not a self-contained system. Its functionality is used by TripleO and will also be relied on by Fuel.

MAAS [5] is used for the provisioning of Ubuntu in combination with JuJu and Charms (see Section 3.2). Similar to Cobbler and FAI, a MAAS server acts as a DHCP server for the provisioning of machines. Configuration such as MAC to IP mapping can be done in a JuJu YAML Ain't Markup Language (YAML) file (see also Section 5).

3.2 OpenStack Installation

The automated provisioning, configuration, and installation of services is addressed by CM systems. Thus, after each node has been installed with an OS, the installation and configuration of IaaS services can be realized using CM. For the deployment of OpenStack there is a variety of existing tools:

Compass [6] supports different CM systems through a plugin architecture. By establishing abstraction layers, it also decouples resource discovery and bare metal installation. Besides, it facilitates operations support system (OSS) integration.

Crowbar [7] is a project that relies on the Chef CM system for the deployment of applications such as OpenStack or Hadoop. In contrast to other solutions of this category it does not presume but also realizes bare metal installation and comes with a web UI.

Fuel [8] offers a web UI frontend for the deployment of OpenStack in addition to a CLI. Cobbler is currently used under the hood, yet, migration to Ironic is intended. Puppet is used for CM. Some features comprise the automated discovery of nodes and the possibility to perform pre-deployment checks.

JuJu [9] is an orchestration technology that is also used for MAAS. As with MAAS, also the deployment of OpenStack is specified in form of an orchestration in a YAML file. Charms, classified by (Wettinger et al., 2014) as environment-centric artifacts, deploy the actual OpenStack services.

Packstack [10] provides Puppet modules for OpenStack projects. Using Puppet for CM, the various OpenStack services can be deployed. Thus, some

[2] http://cobbler.github.io

[3] Fully Automatic Installation (Gärtner et al., 1999; Lange, 2010) (FAI). http://fai-project.org

[4] OpenStack Bare Metal Provisioning (Ironic). http://wiki.openstack.org/Ironic

[5] Metal as a Service (MAAS). http://maas.io

[6] http://wiki.openstack.org/Compass

[7] http://crowbar.github.io

[8] http://wiki.openstack.org/Fuel

[9] http://jujucharms.com

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

front-end deployment tools such as RDO (see below) make use of Packstack. Currently, distributions based on RedHat Package Manager (RPM) are supported.

RDO [11] is a web-based deployment tool based on Foreman, a Ruby on Rails application and frontend for the CM with Puppet. Therefore Packstack is used.

TripleO [12] is an exception to the CM-based solutions. Instead of relying on such, TripleO aims at realizing the functionality using OpenStack's own cloud features for facilitating installation, management, and operation. For this, a deployment cloud (a.k.a. undercloud) needs to be setup first. Using Ironic workload cloud(s) (a.k.a. overcloud(s)) are deployed. The deployment and configuration of nodes is realized using Heat. For this golden images need to be prepared. These consist of a base OS with elements on top (resembles FAI class and profile concept). During provisioning a node will configure itself using the parameters from a Heat Orchestration Template (HOT) that constitutes the deployment plan. Finally, an overcloud can be scaled using the Tuskar subproject.

3.3 Positioning and Contribution

Currently, there is neither a standard nor a common datamodel for the configuration of an OpenStack deployment in a cloud DC. As a result, none of the projects exposes its configuration in a form that can be used by other projects. This however would be interesting in order to evaluate different frameworks and avoid tool dependencies. TripleO – aiming at avoiding any third party dependency for the deployment of OpenStack – is a particular case. It is using HOT for realizing the CM. This way, it decouples the configuration from the automated deployment. It may be argued that HOT is an established format that other tools could implement. Yet, this is not feasible, as it dictates orchestration through Heat. Not only Heat but also JuJu is an orchestration technology. In both cases, therefore, configuration needs to be expressed in a particular syntax and way by experts leading to the problems mentioned such as scattering and tangling.

MING in contrast truly decouples configuration from automated deployment technologies. It declarativly permits the view-based modeling of DCs and their deployments, facilitating SoC. As a result, stakeholders that are not familiar with the used deployment technologies and/or other concerns of the deployment are supported as well. Similar in spirit with Cobbler MING integrates with arbitrary tools and frameworks (as also envisioned by Compass) through code generation.

[11] http://rdoproject.org

[12] http://wiki.openstack.org/TripleO

4 ABSTRACTING FROM TECHNOLOGIES

MING aims at a tool agnostic, declarative specification of configuration for realizing an IaaS deployment in a cloud DC from bare machines. For this, configurations from existing provisioning and deployment technologies have been sublimated using reverse model-based engineering (MBE). That is, models are established through abstraction. Given a valid deployment plan in form of MAAS and OpenStack JuJu files, DC specific values and repetitive code has been identified in a first step.

For capturing respective information in models, a conceptual metamodel has been derived and templates have been created in a next step. Finally, integration with the target technologies was realized using code generation. That is, the same code was generated using the MBE approach. As a result, a conceptual modeling layer has been established with sublimated configuration in form of models.

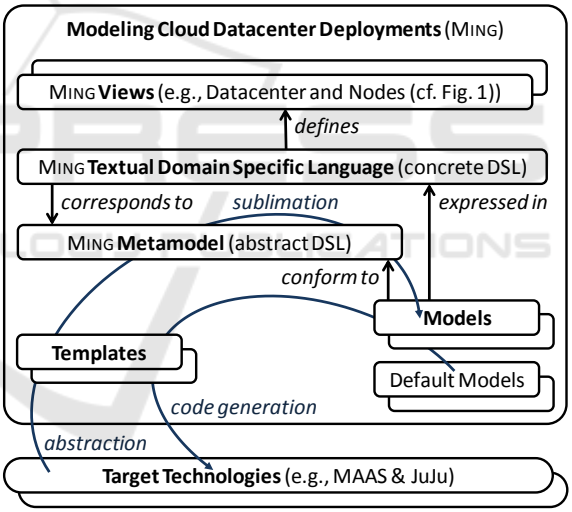

Figure 2: Overview of the MING Framework.

Figure 2 depicts the MING framework. As an interface for populating, expressing, and representing models, a textual DSL has been defined for the MING metamodel (i.e., abstract DSL). In order to support SoC, distinct views permit the expression of different aspects. One view, e.g., covers the DC related information as depicted in the metamodel shown in Figure 1. Other views capture networking, node assignments (e.g., to an aggregate), OpenStack specific configuration, and credentials.

For supporting convention over configuration, defaults can be expressed in models too that are applied to models in case of missing configuration. Finally,

model transformation processes the (resulting) models and generates code using templates.

5 IMPLEMENTATION

For realizing the MING prototype, the Eclipse Modeling Framework (EMF) [13] was chosen as a modeling foundation. Xtext served for defining the DSL and its views and for obtaining a respective editor. Finally, model transformations were implemented in Xtend.

For a better understanding of the approach and in order to present some details, two artifacts (i.e., a code generation and a DSL view) are explained.

```
nodes:
  «FOR az : dc.azs»
  «FOR rack : az.racks»
  «FOR node : rack.nodes»
  - name: «model.deployment.name»-«node.name»
    «IF node.nodeType == NodeTypeName.CN»
    tags: «getComputeAggregate(zones, node, az)»
    «ELSEIF node.nodeType == NodeTypeName.SN»
    tags: storage-«getCephPool(zones, node).name»
    «ELSEIF node.nodeType == NodeTypeName.MN»
    tags: api
    «ELSEIF node.nodeType == NodeTypeName.NN»
    tags: gateway-«getGatewayZone(zones, node).name»
    «ELSE»
    tags: standby
    «ENDIF»
    architecture: «node.arch»/generic
    mac_addresses:
    «FOR nic : getNICs(node)»
      - «nic.mac»
    «ENDFOR»
    power:
      type: ipmi
      address: «getIPMI(node)»
      user: «model.credentialsIPMI.username»
      pass: «model.credentialsIPMI.password»
      driver: LAN_2_0
    «enrichWithIPs(node)»
    «FOR nic : getNICs(node).filter[it.ip4 != null]»
    sticky_ip_address:
      mac_address: «nic.mac»
      requested_address: «nic.ip4»
    «ENDFOR»
  «ENDFOR»
  «ENDFOR»
  «ENDFOR»
```

Figure 3: Code Generation for MAAS `nodes` with Xtend.

Figure 3 depicts an excerpt from the model-to-text (M2T) transformation for generating a MAAS configuration file. Three FOR loops iterate over all DC's availability zones, racks, and finally nodes. As a result an entry is generated for every node containing all of its MAC addresses and assigned IP addresses. The latter are specified in a `sticky_ip_address` section. Code generation assures the consistency between the MAC addresses.

Please note that the same model can be processed by a different template for supporting a different target technology.

[13] http://eclipse.org/modeling/emf

```
IaaS Deployment SongThrush @ MingDC8
OpenStack Liberty on Trusty

Compute
      liveMigration
      mtu 9000

Ceph
      osdFormat btrfs
      osdReformat

Percona
      maxConnections 12345

Neutron
      externalBridge "br-ext"
      overlayNetType "gre vxlan"
      l3_ha
      l3_agents 2–4
      mtu 9000

Heat
      workers 8
      hiddenTags "generated"
```

Figure 4: Deployment of OpenStack – DSL View.

For configuring OpenStack various variables exist for the different services. Using the DSL editor a user profits from code completion, syntax highlighting, and validation. Figure 4 shows an example configuration view for the deployment of OpenStack services at a DC using a referenced OS image for the bare machine installation and a certain version of OpenStack.

Together with the other views (e.g., an instance of the metamodel as shown in Figure 1) and the default models information is complete for model transformation to take place. The current prototype supports generation of JuJu YAML files for MAAS and OpenStack. For the sublimated configuration, the ratio between the size of MING models and YAML code for MAAS and JuJu yielded 27%. That is, the models in MING are nearly four times more compact than the corresponding code as generated by the templates.

6 DISCUSSION

The model-based approach enables the utilization of diverse technologies. Given availability of respective templates, this enables evaluation of different deployment tools. That is, from the same models configurations are generated using the templates. This in turn fosters a common datamodel for establishing a standard for configuring an IaaS deployment from bare machines. With the separation of the models in distinct views further benefits can be leveraged:

Discovery of nodes and their components automatically yields a certain view. Credentials as stored in another view are generated initially if not set for a certain deployment. In both cases parts of the overall configuration are provided and the DSL user only needs to focus on the other aspects of a deployment.

For a given DC it is possible to specify different

deployments. That is, while the physical setup (i.e., availability zones, racks, and nodes) as captured in one view does not change, views related to the deployment such as the assignment of nodes or the deployment and configuration of OpenStack services may be different. Yet, only a part of the overall configuration is adapted and existing views can be reused avoiding software clones. This way, deployments can be tested and the differences between them can be described precisely by comparing two models.

In case a different DC shall be deployed similarly, it is possible to reuse views such as the configuration of OpenStack services. This eases the deployment of multiple DCs with a tested configuration.

The possibility to specify default configuration options in models permits custom user-defined defaults. The fact that these are then applied on the views has two major advantages: It simplifies the models by moving default configuration options out of the views making the files more compact. Also, it simplifies code generation by only processing the resulting model and makes it independent from any (user-defined) defaults.

Regardless of the benefits of this model-based approach, it is always possible to continue work with the output. Thus, MING does not introduce any dependency in regard to the underlaying automation.

Not all fine-grained configuration options of the bare machine provisioning or IaaS solution are reflected in the MING metamodel. Thus, in case these shall be lifted to the modeling layer, the metamodel and the views need to be extended. In case multiple technologies are supported using code generation, certain features of one technology may not be supported by alternatives. For example, the deployment of IaaS services may be realized using Kernel-based Virtual-Machine (KVM) or Linux Containers (LXC). In case such a configuration option is specified but not supported by a technology a fallback may be realized during code generation. For early feedback this can be addressed through validators in the DSL. That is, when selecting an option that is not supported by some technologies a warning is displayed in the DSL editor.

7 CONCLUSION

In this paper MING, a model-based approach, has been presented for the tool agnostic, declarative configuration of IaaS deployments in cloud DCs.

A textual DSL permits to describe such deployments from different view-points. Through code generation particular technologies for the automated installation and deployment of a base OS and OpenStack as an IaaS solution are leveraged.

ACKNOWLEDGMENTS

The author would like to thank the extended Infrastructure Cloud team and in particular Tobias Brausen and Thomas Oswald for valuable feedback and comments and Mike Machado for proofreading.

REFERENCES

Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2013). How to Adapt Applications for the Cloud Environment. *Computing*, 95:493–535.

Cerf, V. G. and Khan, R. E. (1974). A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22:637–648.

Chandrasekar, A. and Gibson, G. (2014). A comparative study of baremetal provisioning frameworks. Technical Report CMU-PDL-14-109, Parallel Data Lab, Carnegie Mellon University.

Droms, R. (1997). Dynamic Host Configuration Protocol. RFC 2131, The Internet Engineering Task Force.

Gärtner, M., Lange, T., and Rühmkorf, J. (1999). The fully automatic installation of a Linux cluster. Technical Report 379, Computer Science Department, University of Cologne.

Intel Corporation (1999). Preboot Execution Environment (PXE) Specification Version 2.1. http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf. [accessed in March 2016].

Lange, T. (2010). 10 Jahre FAI Projekt. Technical Report 603, Computer Science Department, University of Cologne.

Mell, P. M. and Grance, T. (2011). The NIST Definition of Cloud Computing. Technical Report SP 800-145, National Institute of Standards & Technology.

Sollins, K. R. (1992). The TFTP Protocol (Revision 2). RFC 1350, The Internet Engineering Task Force.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. (2006). Ceph: A scalable, high-performance distributed file system. In Bershad, B. N. and Mogul, J. C., editors, *7th Symposium on Operating Systems Design and Implementation*, pages 307–320. USENIX Association.

Wettinger, J., Breitenbücher, U., and Leymann, F. (2014). Standards-Based DevOps Automation and Integration Using TOSCA. In *7th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 59–68. IEEE.

Zimmermann, H. (1980). OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432.