云应用部署配置模型①

姬 源, 黄育松, 谢 冬, 王向东

(贵州电网公司 电力调度控制中心, 贵阳 550002)

摘 要: 云计算已经成为广泛使用的计算范型, 越来越多的大规模分布式系统已经或正在向云平台部署和迁移. 用户在部署和管理维护应用系统时通常需要管理底层基础设施资源细节, 或者使用平台提供方的应用部署和管理服务, 前者使得应用部署和运行时管理易于出错且费时费力, 而后者则降低了系统管理的灵活性, 很难满足用户的个性化需求. 针对这一问题, 本文提出了一种高层抽象模型来描述云应用的部署配置和管理需求. 需求模型采用声明式机制定义期望的系统状态, 而无需描述实现目标状态所需的执行步骤和细节. 本文基于开源云计算平台 OpenStack 和自动化配置管理工具 Puppet 进行了原型实现, 通过一个应用案例验证模型的有效性.

关键词: 云计算; 需求模型; 配置管理; 部署

Deployment and Configuration Model of Cloud-Based Applications

JI Yuan, HUANG Yu-Song, XIE Dong, WANG Xiang-Dong

(Guizhou Electric Power Grid Dispatching and Control Center, Guiyang 550002, China)

Abstract: Cloud computing has been a popular computing paradigm, and more and more large-scale distributed systems are migrating or have been migrated to cloud platforms. When deploying and managing IT systems in cloud environments, system owners have to manage the low-level details of the infrastructures, or leverage the services of platforms. Managing low-level details is time consuming and error-prone, and leveraging the services is less flexible and more difficult to tailor to user requirements. To address these issues, we propose a high-level abstraction model for cloud-based systems to specify their requirements. The requirement model describes the expected system states in a declarative way, without introducing the low-level steps and details about how to reach the states. We implemented a prototype based on open source cloud platform OpenStack and configuration management tool Puppet, and the effectiveness of the requirement model is evaluated through a case study.

Key words: cloud computing; requirement model; configuration management; deployment

随着智能电网技术的不断发展, 电力系统中数据和信息呈爆炸性增长, 为系统的运行和数据分析带来巨大挑战. 电力系统现有的硬件设施和计算能力已难以适应未来电力系统在线分析和实时控制所要求的计算能力和存储要求. 迫切需要引入新的计算技术, 为智能电网提供有效的技术支撑[1]. 通过建立智能电网云计算平台, 可以有效整合系统中的资源, 为潮流计算、状态估计、安全分析、采集监控等提供强大的并行计算与海量存储能力支持.

云计算作为一种新计算范型正在影响着 IT 系统的设计、部署和使用方式[2]. 当前云计算服务分为三大类: 基础设施及服务(Infrastructure as a Service, IaaS)、平台即服务(Platform as a Service, PaaS)和软件即服务(Software as a Service, PaaS).

包括 Amazon Web Service (AWS)[3]和 OpenStack[4]在内的 IaaS 允许用户访问平台提供的虚拟化基础设施资源 (如 CPU, 存储, 带宽等), 但是用户部署和管理应用系统时必须面对所有的软件堆栈和系统的底层细节, 因此这种完全的灵活性使用户在部署和配置应用是易于出错[5,6]. 另一方面, SaaS 为最终用户提供预先

① 收稿时间:2015-08-25;收到修改稿时间:2015-10-19

配置好的应用服务, 如 SalesForce[7]. 用户通常只能够从一组预先定义的模板中进行选择, 而这些模板很难覆盖所有用户的不同需求. 作为处于SaaS和IaaS之间的PaaS, 如Google App Engine和Windows Azure, 为用户的系统应用提供了运行支撑环境, 包括操作系统以及所需的各种中间件和服务, 使得用户能够在 PaaS上完成系统的部署和管理. 但是当前已有的PaaS平台在对用户部署和管理应用方面提供的支持仍然有限, 同样具有平台相关性强, 难以满足用户需求的问题.

当前, 在云计算环境下缺少能够完整描述 IT 系统目标和功能的高层抽象模型, 用以辅助应用系统的部署和长期运行维护. 尽管当前已有工作在云应用抽象建模方面进行了探索, 但是还没有能够将用户需求与运行平台以及应用系统的实现细节完全分离, 因而也使得这些方法的使用场景和适用范围受到了限制, 难以达到适用于多种云计算平台的灵活性, 能够满足用户对系统长期运行维护的目标.

对于有效的云应用系统高层抽象模型来讲, 它应该具备以下几个特点:

① 模型必须能够明确的声明用户需求, 而不是通过一系列的实现细节来描述如何满足这些需求.

② 模型应该能够针对不同的云计算设施平台将描述需求和目标自动化的体现在系统的部署和维护管理中.

③ 能够辅助系统运行维护和管理, 包括对应用实例的弹性伸缩和应用系统的更新等.

针对上述问题, 本文研究并提出了一种云应用部署配置模型, 该模型采用声明式的方法刻画云应用的部署拓扑结构, 实现用户需求与运行平台以及应用系统的实现细节的分离. 在此基础上, 本文还基于开源云计算平台 OpenStack 和自动化配置管理工具 Puppet 进行了原型实现, 并通过一个应用案例初步验证了模型的有效性.

本文第 1 节提出面向云应用系统的部署配置模型, 对其语法结构和模型进行介绍. 基于该模型, 第 2 节介绍系统原型实现, 描述了原型的整体架构, 并在第3节的案例中对模型的有效性进行实验评估. 第 4 节介绍本文的相关工作. 第 5 节对本文的工作进行总结, 并讨论了未来的工作方向.

1 部署配置模型

1.1 YRML

本文采用一种领域相关的语言来进行目标系统的高层需求建模, 即 YRML (YAML-based Requirement Modeling Language). YRML 的语法结构主要基于YAML [8]. YAML 是一种支持任意名值对(key-value)描述形式的语言. YRML 的抽象语法结构如图 1 所示.

```
ReqModel ::= goals => GoalSpec+
             roles => RoleSpec+

GoalSpec ::= goalID =>
                [name => string]
                roles => roleID+

RoleSpec ::= roleID =>
                [name => string]
                [min => integer]
                [exports => ChannelSpec+]
                [imports => ImportSpec+]
                implementations => ImplementationSpec+

ChannelSpec ::= chanelID =>
                (type => channelTypeID, ChannelAttr*)

ChannelAttr ::= attributeID => value
ImportSpec ::= roleID => channelID+
ImplementationSpec ::= implementationID = value
```

图 1 YRML 语法结构

ReqModel 表示目标系统的整个需求模型, 系统需求模型主要描述了系统目标(*goals*)和系统角色(*roles*).

目标(*goals*)是对系统高层的、总体的业务功能的简单描述, 便于快速掌握系统的设计和实现目的. 例如, 对于博客系统的目标可以表示为 *blog website*. 每个目标系统的需求模型中至少具有一个目标.

每个系统目标(*goals*)需要通过一个或多个角色(*roles*)实现. 角色(*roles*)定义了实现系统目标的逻辑单元, 每个角色在其中分别承担不同的业务功能. 对应到具体的应用系统中, 角色可以是构成系统的组件和其他软件, 例如系统包括的数据库、Web 应用、消息代理等. 为了提供良好的系统性能和服务质量, 云环境下的应用系统通常采用集群方式提供服务, 因此系统中的每个角色可以代表一组提供相同功能的服务实例, 因此在需求模型中以一个角色来表示具有相同功能的所有实例集合. 对于每个角色的定义主要包括: 角色名(*name*)、最少实例数(*min*)、对外提供的接口(*exports*)和需要依赖的接口(*imports*)以及角色实现(*implementation*).

接口(*imports* 和 *exports*)是目标系统中各个角色之间

交互的基础, 角色之间可以相互依赖. 每个角色对外提供的接口通过信道(*Channelspec*)加以定义, 信道定义了接口的类型(*type*), 其中包括了接口相关的 0 个或多个属性(*ChannelAttr*). 例如, 一个 Web 应用 webapp 可以对外提供一个 HTTP 类型的接口, 其接口描述可以声明这是一个 TCP 端口. 对于当前角色依赖的其他角色的接口(*import*)则通过信道(*ImportSpec*)声明, 其中通过声明被依赖角色的相关接口(*channelID*)进行描述.

每个角色的实现(*implementations*)定义的不是该角色的具体代码实现, 而是定义了对于当前角色进行部署和维护管理的相关操作. 角色实现(*implementation*)值的具体表示形式依赖于系统所采用的配置管理工具(Configuration Management Tool, CMT), 如 Puppet [9]、Chef [10]等. 结合选定的 CMT 工具, 角色实现将采用相应的语言对该角色的部署配置和维护管理操作进行定义, 从而基于 CMT 工具实现对目标系统和角色的自动配置和管理.

1.2 部署配置模型

本方法的核心思想是对 IT 系统的功能和结构进行抽象建模, 并用来支持实际系统的部署配置和管理. 如图 2 所示, 整个部署配置模型由两部分组成, 即需求模型(requirement model)和系统模型(system model).

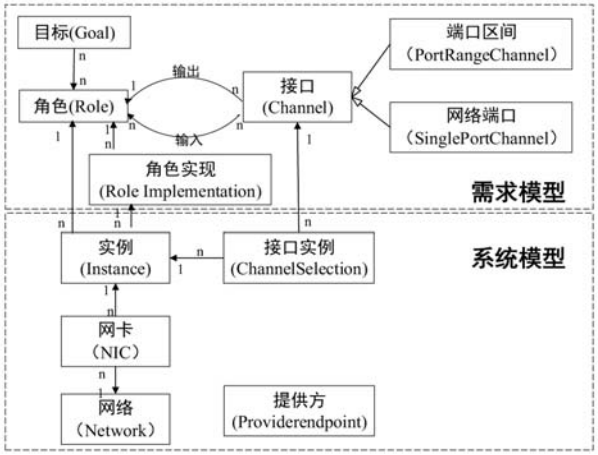

图 2 部署配置模型

图 2 是基于实体关系图的部署配置模型, 模型元素实体之间的有向线段表示实体间的关联, 模型同时还描述了不同实体之间实例数量的对应关系(包括: 1:1, 1:n, n:n). 例如, 角色与目标之间是多对多的关系, 表示每个目标可以由多个角色共同完成, 同时每个角色也可以服务于多个目标.

需求模型用来刻画用户的需求和对目标系统的高层抽象信息, 具体系统的细节信息则通过系统模型加以维护. 当通过需求模型实现对某个应用系统的部署创建后, 其对应的系统实例信息则能够通过对应的系统模型表示. 需求模型和系统模型之间保持一致, 当系统在运行过程由于需求变化而发生系统改变(如: 升级、弹性伸缩、配置改变等)时, 系统模型能够产生相应的变化并确保模型信息与实际系统的一致. 系统模型是对基于云的 IT 系统的一个全局映射, 使系统管理人员能够通过构成系统的多个角色实现对系统具体结构和信息的掌握. 同时, 系统模型实现高层抽象(即需求模型)到具体系统实现之间的衔接.

由图 2 可以看到, 需求模型和系统模型之间主要通过角色(*Role*)和实例(*Instance*)以及接口(*Channel*)和接口实例(*ChannelSelection*)实现衔接和映射. 实例表示承担相应角色的虚拟机及运行其上的软件和服务, 每个角色可以有多个实例, 而每个实例只能属于一个角色. 每个实例通过接口实例访问当前角色对外提供或依赖的接口. 接口实例是需求模型中接口的具体实现, 可以是一个指定的网络端口(*SinglePortChannel*), 或者是一个端口区间(*PortRangeChannel*).

在实际应用中, 我们基于 YRML 对目标系统的部署配置模型进行刻画, 每个目标系统的部署配置模型满足图 2 所定义的实体类型及其之间的关联.

2 系统实现

以部署配置模型 YRML 为基础, 本文实现了一个基于 Ruby 语言的原型系统(Prototype of Concept, PoC), 该系统的总体结构如图 3 所示.

PoC 以开源 IaaS 平台 OpenStack 作为云应用的运行支撑环境. 同时, PoC 提供了一个针对云应用的部署配置及需求建模环境, 采用 YRML 进行应用建模. 对于云应用的 YRML 模型, PoC 通过编译器模块将声明式的模型信息编译转换为能够被 CMT 工具执行的脚步. 最后, PoC 基于 CMT 工具 Puppet 实现系统到云平台的自动部署配置和管理维护.

基于需求模型, 编译器确定目标系统中每个角色的实例个数、角色实现描述、IP 地址和接口等. 编译器基于类型检查机制来确保需求模型结构良好, (例如, 只有在当前角色对外提供了某个接口, 该接口才能够被其他角色引用). 然后, 编译器将需求模型转换为具体的执行操作, 包括: 云平台虚拟机的创建, 以及应

用系统的安装、配置和运行时管理的一系列操作. 在具体执行过程中, PoC 通过 Fog(Ruby 的一个云服务类库)调用 OpenStack 的 API 来提供基础设施资源, 包括虚拟机实例、网络等. 虚拟机创建后, PoC 使用 Puppet 完成应用系统实例的部署和配置.

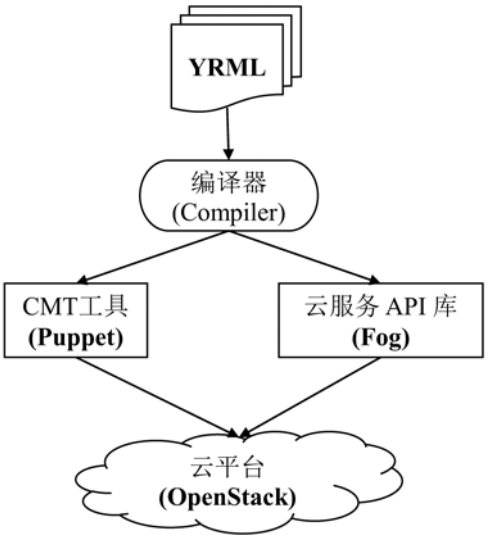

图 3 PoC 总体结构

PoC 对外提供 REST(Representational State Transfer)API [11], 客户端主要包括基于命令行的接口(Command Line Interface, CLI), 用来对应用系统进行部署配置和维护管理.

3 应用案例

本节主要基于一个典型的三层架构 Web 系统 Blogging Website 来初步分析评价本文所提出的需求模型在云环境中应用的有效性. 整个系统架构如图 4 所示.

Blogging Website 主要提供博客服务, 通过 Varnish [12]实现负载均衡, 将来自多个客户端的请求分发给 Web 应用 WordPress [13]集群中的多个实例, 每个实例通过访问后端的 MySQL 数据库实现博文的管理.

该应用的 YRML 模型描述如图 5 所示. 其中, 整个目标系统具有一个目标, 即 wordpress, 该目标的名称为 Wordpress blog. 为了实现这一目标, 系统的 YRML 模型声明了 3 个角色, 即: weblb、webapp 和 db, 分别对应构成系统的三类软件系统 Varnish Load Balancer、WordPress Web Application 和 MySQL Database. 在系统的 YRML 模型中, 三类角色通过接口(imports 和 exports)来描述之间的相互依赖关系, 例如 weblb 使用 webapp 的 http 接口(图 4 中 17, 18 行), 而 webapp 则使用了 db 的 querying 接口(图 4 中 27, 28 行). 特别的, 应用系统 WordPress Web Application 对应的角色 webapp 由于在系统部署中采用多实例的集群方式来保障整个系统的执行性能, 因此 YRML 模型对 webapp 声明了其最小实例个数(图 4 中 21 行), 即 webapp 至少具有 2 个或者更多的实例, 系统可以根据运行时的实际情况通过弹性扩展的方式增加和减少 webapp 的实例数.

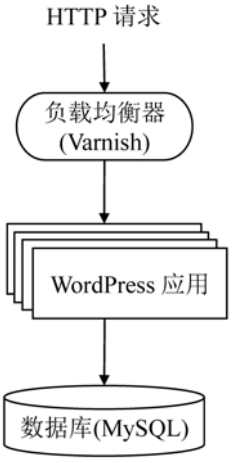

图 4 Blogging Website 系统结构

```
1.  goals:
2.    wordpress:
3.      name: Wordpress blog
4.      roles:
5.        - db
6.        - webapp
7.        - weblb

8.  roles:
9.    weblb:
10.     name: Web application load balance
11.     is_public: true
12.     implementations:
13.       default:
14.         profiles: role::weblb::default
15.     exports:
16.       http: {type: single_port, protocole: tcp, number: 80}
17.     imports:
18.       webapp: http

19. webapp:
20.   name: Web application
21.   min: 2
22.   implementations:
23.     default:
24.       profile: role::weblb::default
25.   exports:
26.     http: {type: single_port, protocol: tcp}
27.   import:
28.     db: querying

29. db:
30.   name: MySQL database
31.   implementations:
32.     default:
33.       profile: role::weblb::default
34.   exports:
35.     querying: {type: single_port, protocol: tcp}
```

图 5 Blogging Website 的需求模型

为了基于 YRML 模型实现应用到云计算平台 OpenStack 的自动部署配置和管理. PoC 基于 Puppet 脚本完成对每个角色实现(implementation)的定义. 在 YRML 模型中, 采用与普通编程代码类似的方式声明每个角色实现所对应的实现类, 如图 5 中 14、24 和 33 行所示. 角色的实现脚本作为单独文件存储, 与 YRML 分离, 达到抽象需求描述与具体实现细节分离的目标, 也能够实现对多种 CMT 工具的使用和集成. 以角色 weblb 实现为例, 图 6 展示的是其对应的基于 Puppet 语法的脚本实现, 在实际应用中同样可以采用其他 CMT 工具进行实现(Puppet 脚本语法不作为本文内容, 可参看相关文献).

```
1.  class role::weblb::default {
2.      class { “profile::varnish”:
3.                  listen_port => ’80’
4.              }
5.      file { “default.vcl”:
6.          ensure  => file,
7.          content =>
8.            template (“role/weblb-varnish/default.vcl.erb”),
9.          path    =>  “/etc/varnish/default.vcl”,
10.         owner  =>  root,
11.         group   =>  root,
12.         mode   =>  644,
13.         require =>  Package [“varnish”],
14.         notify   =>  Exec[“reload-varnish”],
15. }
```

图 6 角色 weblb 的 puppet 实现脚本

基于 PoC 系统, 本文还通过在运行时改变 Web 应用系统 Blogging Website 的 YRML 模型来实现系统运行时的弹性伸缩操作, 主要体现在对 WordPress Web Application 实例的动态增减方面, 该操作主要是通过改变 YRML 中角色 webapp 的实例数完成. 如果通过增加和减少应用模型描述中 webapp 的实例个数能够成功触发实际系统中实例个数的变化, 那么对外表现出来的变化是系统性能将会随着集群中实例的变化而提高或者降低, 能够从侧面验证本文所提出模型对于支持应用系统需求描述和部署配置执行的有效性. 实验步骤如下:

① 在基于 OpenStack 搭建的私有云环境下, 根据图 5 描述的部署配置脚本进行 Blogging Website 的初始部署, 整个系统包括 1 个负载均衡节点(weblb), 1 个 MySQL 节点(db)和 3 个 WordPress Web Application 节点(webapp, 运行在 tomcat 上), 其中每个节点都运行在一个单独的虚拟机上.

② 以上述部署配置作为基准测试环境, 通过压力测试工具 JMeter 进行并发负载的生成(500 并发), 并观察系统的响应时间和吞吐率, 以 10 次测试的平均值作为测试基准.

③ 增加系统部署配置模型中 webapp 的实例个数, 增加一个 webapp 实例. 然后, PoC 通过 Fog 获取一个虚拟机实例, 并触发 Puppet 工具实际完成 webapp 新实例的部署和配置. 然后与第 2 步相似, 采用 JMeter 以 500 并发不断对系统产生负载, 观察此时的应用吞吐率.

④ 减少系统部署配置模型中 webapp 的实例个数(减少至 2 个实例). 然后与第 2 步相似, 采用 JMeter 以 500 并发不断对系统产生负载, 观察此时的应用吞吐率.

实验结果如表 1 所示, 其中分别记录了案例系统 Blogging Website 的性能测试基准结果(baseline)以及通过 YRML 和 PoC 分别增、减一个集群实例时的测试结果. 整个验证过程能够保证案例系统的正常运行, 同时其性能变化也符合集群实例变化应该导致的变化趋势. 因此, 在一定程度上能够验证本文部署配置描述模型以及原型系统 PoC 的有效性.

表 1 Blogging Website 系统性能变化情况

	平均响应时间(ms)	吞吐率(requests/sec.)
测试基准	456.13	89.49
增加实例	418.76	96.63
减少实例	458.89	86.18

在本文方法的可扩展性和适用性方面, 本文通过 YRML 模型实现了高层抽象描述与底层细节的分离, 仅仅通过在模型中对角色对应的实现类进行声明来完成对底层部署和配置执行细节的引用. 因此, 在具体实现层面, 本方法可以集成不同的 CMT 执行供给, 例如出 Puppet 之外的 Chef、Ansible 和 Salt Stack 等, 仅仅需要将模型中声明的实现类与具体的执行脚本相关联即可. 例如, 在集成 Chef 时, 可以将本文图 5 中 14、24 和 33 行实现类的声明与 Chef 对应的实现脚本关联, 而此时 weblb 实现类对应的脚本将不是图 6 所对应的 Puppet 脚本, 而是采用 Chef 对应的脚本语言和语法所描述的具体执行细节.

另一方面, 本文的原型系统 PoC 在与底层 IaaS 平台集成时采用了基于 IaaS 所提供的 API 方式实现, 本文实验中集成了开源系统 OpenStack 提供的 API 库,

对于其他云计算开源系统或者云计算提供方(如 AWS、阿里云等)同样可以通过提供的 API 和服务完成虚拟资源的申请, 因此能够广泛适用于主流的 IaaS 云计算环境.

4 相关工作

本文研究的云应用部署配置模型主要应用在云计算环境中 PaaS 对应用系统的部署配置和管理, 与本文最为相关的领域主要有应用部署模型、云计算 PaaS 平台、配置管理方面.

在模型方面, 当前面向云应用的模型包括: AWS CloudFormation、OpenStack Heat、Terraform, 这些应用模型主要针对 IaaS 层资源的创建与管理, 基于配置文件创建 IaaS 层资源, 依旧无法分离用户部署需求和部署的实现细节.

当前不同的 PaaS 提供方也在不同程度上提供了系统抽象描述机制, 如 Windows Azure definition schema[14] 和 Google AppEngine YAML-based specifications[15]都使用户能够定义自己的云应用系统, 但是这些方法与 YRML 在实现目标方面有所不同. 上述方法仅仅针对用户的应用系统提供描述机制, 而由于没有对云平台本身的组件进行建模, 因此无法描述包括底层基础设施资源在内的完整的系统. 同时这些抽象描述方法具有平台相关性, 依赖于特定的 PaaS 平台, 无法将用户需求与平台细节分离, 难以适用于不同的云环境.

在 CMT 方面, 当前广泛适用的配置管理工具有 Chef[10]和 Puppet[9], 两者工作机制和原理相似. 以 Puppet 为例, 其基于 C/S 架构, 通过安装在服务器结点上的代理(agent)实现对该结点上系统的安装、配置和其他管理操作, 相应的管理操作通过中心控制结点(master)向各个目标结点分发, 这些操作任务以声明式的脚本语言描述, 其中定义了对相关系统所期望达到的目标状态. 本文实现的 PoC 基于 Puppet 实现应用系统的实际安装配置和管理操作, 但是本文提出 YRML 模型并不依赖于某个特定 CMT, 如 Puppet. 这是由于 YRML 中对于每个角色的抽象实现描述与具体的实现方法分离(如本文案例中图 4 和图 5 所示), 因此在系统实现中可以根据具体实现技术采用的不同的角色实现描述方法.

5 结束语

云计算平台能够为智能电网提供有效的技术支撑. 在云计算环境下的应用部署和运行维护中, 将用户需求与实现细节分离有助于提高 IT 系统的运维效率和灵活性. 本文提出了一个描述用户高层抽象需求的 YRML 模型用以辅助实现云应用系统的部署维护建模. 基于开源云平台 OpenStack 和 CMT 工具 Puppet, 本文实现了一个原型系统 PoC, 并通过 YRML 模型对一个系统案例进行高层抽象需求建模, 完成系统向目标云平台的初始部署以及运行时管理, 该案例和实验结果验证了 YRML 模型的有效性. 未来工作将通过更多的应用案例和实验, 并结合不同的云计算环境, 进一步验证 YRML 的广泛适用性, 以及对应用在不同云平台部署和迁移的灵活性和可移植性的支持.

参考文献

1 潘睿,刘俊勇,郭晓鸣.电力系统云计算初探.四川电力技术, 2010,33(3):71−76.

2 Armbust M, Fox A, Griffith R, et al. A view of cloud computing. Communications of the ACM. 2010. 50–58.

3 AWS. Amazon Web Service. http://aws.amazon.com/.

4 OpenStack. http://www. openstack.org.

5 Balduzzi M, Zaddach J, Balzarotti D, Kirda E, Loureiro S. A security analysis of Amazon's elastic compute cloud service. Proc. of the 27th Annual ACM Symposium on Applied Computing (SAC). 2012. 1427–1434.

6 Bugiel S, Nurnberger S, Poppelman T, Sadeghi A, Schneider T. AmazonIA: When elasticity snaps back. Proc. of the 18th ACM Conference on Computer and Communications Security (CCS). 2011. 389–400.

7 Salesforce. http://www.salseforce.com.

8 YAML. http://yaml.org/.

9 Puppet Labs. What is Puppet. http://puppetlabs.com/puppet/what-is-puppet.

10 Opscode. Chef. http://www.opscode.com/chef/.

11 Fielding RT. Architectural styles and the design of network-based software architectures [PhD Thesis]. Irvine: University of California, 2000.

12 Varnish Cache Website. About. https://www.varnish-cache.org/about.

13 WordPress Website. WordPress. https://wordpress.org/.

14 Windows Azure Service Definition Schema (.csdef). http://msdn.microsoft.com/en-us/library/windowsazure/ee758711.aspx.

15 Google Developers. https://developers.google.com/appengine/docs/python/config/appconfig.