面向云服务的 DevOps 知识获取与应用①

陈咏秋, 张 斌, 徐明珠, 顾永生

(江苏电力信息技术有限公司, 南京 210024)

摘 要: DevOps 作为一种新兴范型能够实现开发和 IT 运维之间的高度协同, 从而在完成高频率部署的同时, 提高生产环境的可靠性、稳定性、弹性和安全性. DevOps 与云计算一起能够实现资源的按需供给. DevOps 制品和云服务的规模不断增长, 大量的 DevOps 知识分散在不同的社区和来源中, 没有得到有效的组织、管理和使用, 如何针对大量可选的 DevOps 方法和工具进行有效的决策和选择成为亟待解决的问题. 针对这一问题, 提出了一套完整的 DevOps 知识管理方法. 方法首先针对一组可访问的知识源进行多种方式的知识获取、组织、转换和存储; 然后提出了 DevOps 知识分类方法, 并设计实现了 DevOps 知识库原型系统; 最后基于谓词逻辑提出了 DevOps 需求的描述方法, 并展示了基于需求的 DevOps 知识库的使用.

关键词: DevOps; 云计算; 知识管理; 知识库

Methodology of Capturing and Using DevOps Knowledge for Cloud Services

CHEN Yong-Qiu, ZHANG Bin, XU Ming-Zhu, GU Yong-Sheng

(Jiangsu Electric Power Information Technology Co., Ltd., Nanjing 210024, China)

Abstract: DevOps is an emerging paradigm to achieve the highly collaboration between system development and operations in order to enable high frequency of software deployment and improve the reliability, stability, elastic and security of production environment. DevOps is typically combined with Cloud computing to realize rapid, on-demand provisioning of underlying resources. Today, an ever-growing amount of DevOps tools, reusable artifacts and Cloud services are available to implement DevOps automation, and a huge number of DevOps knowledge scatters between different communities and sources. As a result, how to make an appropriate decision and select the most suitable method and tools during application design and deployment has become a big challenge. To address this issue, we propose an approach to manage and utilize DevOps knowledge systematically. The approach firstly captures, links, transforms and stores DevOps knowledge from multiple resources in various ways. Then the approach proposes a set of DevOps knowledge taxonomy and implements a knowledgebase prototype. Finally, the approach describes application of DevOps requirements based on predicate logic, and shows how this knowledgebase is utilized.

Key words: DevOps; Cloud computing; knowledge management; knowledgebase

随着互联网信息技术的快速发展, 软件和服务的生命周期迭代间隔不断缩短, 用户都期望能够在第一时间获得新版系统的最新功能特性, 也希望系统缺陷能够在最短时间内被修复. 因此, 应用系统的快速交付和持续更新能力已成为衡量软件厂商与服务提供商竞争力的重要标准之一, 有效缩短应用发布周期是满足用户期望并提高自身竞争优势的关键[1].

作为一种新兴范型, DevOps (Development 和 Operations 的组合)能够消除开发人员和运维人员之间的隔阂[2-4], 实现开发和 IT 运维之间的高度协同, 从而在完成高频率部署的同时, 提高生产环境的可靠性、稳定性、弹性和安全性.

DevOps 是一组过程、方法与系统的统称, 用于促进开发(应用程序/软件工程)、技术运营和质量保障

① 收稿时间:2015-12-07;收到修改稿时间:2016-01-29 [doi: 10.15888/j.cnki.csa.005318]

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(QA)部门之间的沟通、协作与整合. 它的出现是由于软件行业日益清晰地认识到: 为了按时交付软件产品和服务, 开发和运营工作必须紧密合作. 因此, 开发与运维之间的紧密协作以及端到端的过程自动化(如全自动化的部署流水线[1])是 DevOps 实现持续交付的有效方法.

在 DevOps 的各个环节中都存在着大量的、各种各样的工具、系统、可重用制品和服务来支持开发和运维的协同, 其中具有代表性的有配置管理框架 Chef[6]、持续集成服务器 Jenkins[14]和容器虚拟化技术 Docker[15]等. Chef 采用基于 Ruby 的领域描述语言来创建和执行系统运维脚本—Chef cookbooks, 而这些脚本是执行应用系统软件栈组件自动化部署的基础. Docker 容器镜像使组件具有较好的隔离性和可重用性. 其他的 DevOps 工具及其相应制品还包括: Juju[16]和 Juju charms[17], Puppet[7,8]和 Puppet modules[18]等.

DevOps 通常和云计算[9]相结合, 以自服务的方式实现资源的按需供给, 如虚拟化服务器和存储资源等. 云计算提供方(如 Amazon[19])提供了不同形式的接口(包括: 图形化接口、命令行接口和 API)来供给和管理云计算资源, 尤其是基于 API 和命令行接口能够通过编程方式实现 DevOps 自动化方法和云计算资源的集成, 并且适用于公有云、私有云以及混合云的不同场景. 例如, Amazon 的云计算 API 能够实现虚拟服务器的供给, 然后通过执行 Chef cookbooks 或者是 Juju charms 实现在虚拟服务器上部署应用软件栈. 部分云提计算供方还提供了诸如中间件服务(如: 运行环境即服务、数据库即服务)和自动化运维服务, 这些服务可以作为底层基础设施服务的补充.

DevOps 和云服务的出现和发展变化十分迅速, 存在数量众多、类型多样的 DevOps 相关工具和方法, 并且很多工具在功能方面十分相似, 由此给选择最恰当的方法和工具并将其组合在一起来实现面向特定应用系统的 DevOps 自动化带来困难. 另一方面, 丰富的 DevOps 知识分散在互联网上(如开源工具社区等), 并没有得到有效的管理和使用. 因此, 提供有效的 DevOps 知识管理方法是实现 DevOps 自动化和协作的必要前提, 基于系统化的 DevOps 知识管理为系统设计开发和运维人员提供恰当的 DevOps 工具和方法集合, 从而最终实现应用和服务的持续交付.

针对上述问题, 本文首先提出一套系统化的 DevOps 知识管理方法; 然后面向 DevOps 的知识管理需求对 DevOps 知识库原型系统进行了设计和实现; 最后, 提出了面向应用 DevOps 需求的形式化描述方法, 并实现基于形式化表达式的 DevOps 知识库查询.

1 问题分析

WordPress[20]是当前流行的开源博客应用系统, 基于其三层架构, WordPress 的部署和运维需求包括: 1) 5.0 或更高版本的 MySQL 数据库服务器; 2) 5.2.4 或更高版本的 PHP 运行时环境; 3)Web 服务器, 可以是 Apache HTTP Server 或者 Nginx. 为了支持 WordPress 新版本的持续交付, 上述需求必须在系统的 DevOps 中加以考虑.

图 1 WordPress 应用的中间件部署可选项

WordPress 的多层体系架构如图 1 所示, 其中不同类型的中间件组件可以具体的采用不同的可选技术和实现, 如图 1 中标注所示. 当选择 Amazon 的虚拟机镜像(Amazon Machine Image, AMI)来提供底层基础设施, 并安装 LAMP(Linux+Apache+MySQL)软件栈来支持 WordPress 的部署和运行时, 所有资源都局限于一台虚拟机实例, 使得应用的可扩展性收到限制. 因此, WordPress 的部署运维可以使用开源配置管理工具 Chef, 通过自动化脚本的执行把 MySQL 和 PHP 应用分别部署在两台不同的虚拟机上. 当 WordPress 部署运维还需要提高 MySQL 数据库服务器的可扩展能力时, 可以使用 Juju 提供的 MySQL charm 创建主从(master-slave)结构的 MySQL 集群, 实现应用数据的读写分离以及在多个从数据库实例间的负载均衡. 但是 Juju charm 的使用必须满足一个前提约束, 即 Juju charms 只能够部署在 Ubuntu 操作系统上.

对 WordPress 的部署和运维需求进一步扩展, 如果需要使 WordPress 具有根据负载进行弹性伸缩的能力, 那么可以通过提供数据库即服务(database-as-a

© 中国科学院软件研究所 http://www.c-s-a.org.cn

sercie)和运行环境即服务(runtime-as-a-service)[10]来实现. 例如, Amazon Elastic Beanstalk 和 Google App Engine 提供了 PHP 运行环境, 并对用户屏蔽底层基础设施环境. Amazon Relational Database Service (RDS) 可以作为 WordPress 的 MySQL 数据库服务来满足 WordPress 对数据库的需求. 但是云服务提供商的服务在系统可配置性方面受到限制. 例如, 独立安装的 MySQL 数据库能够根据用户和实际情况进行任意调整与配置, 而提供商的数据库服务则必须使用预先设置好的参数配置.

综上所述, 即使是 WordPress 这类较为简单的应用系统, 虽然只有少数几个部署和运维需求, 仍然存在大量的部署运维可选方案. 另一方面, 不同的 DevOps 方案优缺点各不相同, 导致应用设计和运维人员难以做出方案决策, 从而简洁有效的部署和运维他们的应用系统. 因此, 本文需要解决的主要问题是"如何系统的获取、关联和使用 DevOps 知识, 将其作为应用设计和部署时方法决策的基础".

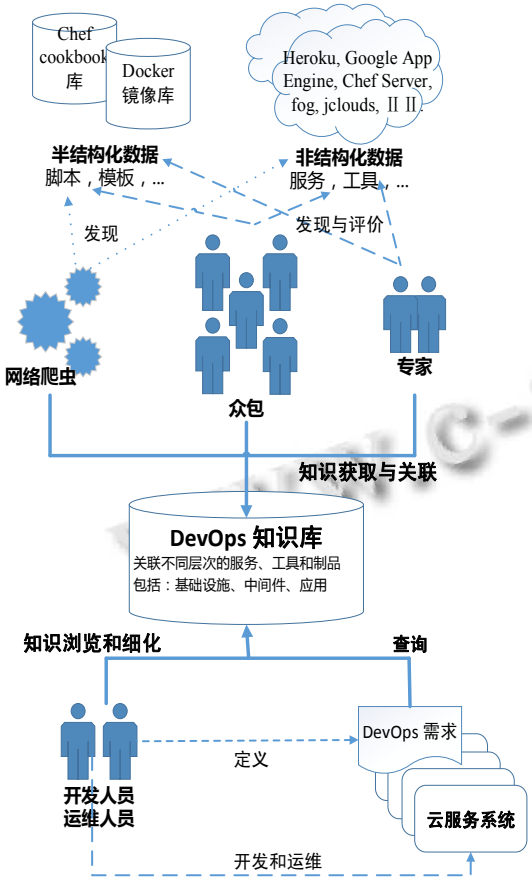

图 2 DevOps 知识管理方法

2 DevOps知识管理

应用系统的开发和运维对于基础设施和中间件的选择都存在大量不同的可选方案. 随着 DevOps 的兴起和发展, 丰富的 DevOps 知识以非结构化和半结构化数据的形式分散在互联网上, 具有不同的信息来源. 例如, 包括 Chef cookbook repository, Puppet forge, Docker Hub, Amazon Web Service' Marketplace 在内的不同公共制品库可以对外提供如脚本、模板和镜像等类型的可重用制品来支持应用系统的运维. 上述类型的公共制品库对其中的制品信息进行了元数据标注和描述(例如: 资源依赖、制品类型、输入/输出、评价信息等), 能够基于网络爬虫自动化的获取这些半结构化的知识、信息和数据.

非结构化数据的自动发现和获取较为困难, 而诸如文档分类[11]的自然语言处理技术存在结果不准确的问题, 因此本方法采用人工方式对非结构化数据(包括: 文档、DevOps 工具、云资源和服务等)进行抽取、整理和评价. 人工方式又可以分为领域专家方式和众包方式(crowdsourcing approach)[12]. 领域专家的优势在于有丰富的知识背景和经验积累, 因此对于抽取和评价的 DevOps 知识具有较高的可信度和准确度, 而众包方式则是以 DevOps 工具和制品相关的开源社区(如 Chef、Puppet、Juju 和 Docker)为基础的 DevOps 知识的收集.

系统化的 DevOps 知识管理方法如图 2 所示, 将基于爬虫、专家和众包方式获得的 DevOps 知识进行关联、细化, 并存储在 DevOps 知识库中. DevOps 知识库能够以多个分布式存储的方式组织和管理不同层面的资源, 包括: 基础设施层的资源供给库、中间件层的部署脚本和应用栈的模板等. 同时, DevOps 知识的发现、获取可以是一个持续的、迭代的过程, 如图 3 所示.

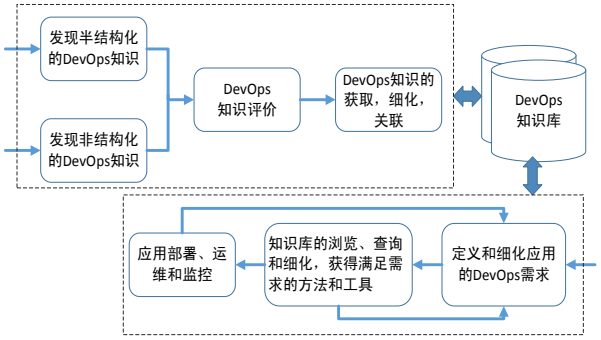

图 3 DevOps 知识的管理和应用

© 中国科学院软件研究所 http://www.c-s-a.org.cn

在知识使用方面, 如图 2 和图 3 中所示, 开发和运维人员针对某个应用提出 DevOps 需求, 从 DevOps 知识库中查询出能够满足这些需求的可选方法. 通过更新和增加新的 DevOps 知识信息(例如增加新的中间件组件部署脚本), 整个知识库可以不断更新存储内容和扩展知识规模. 除此之外, 通过对应用系统的运行时监控, 可以根据系统运行时状态和存在的问题对相应的 DevOps 需求进行细化和调整. 例如, 当系统中某个中间件存在运行时异常, 那么相关的 DevOps 需求也必须随之发生变化.

因此, 系统化的 DevOps 知识管理方法能够在应用的整个生命周期中促进 DevOps 知识的持续性、迭代式的积累、组织和使用.

3 DevOps知识库

DevOps 知识库是 DevOps 知识管理系统的核心组件. 基于知识库的协同并不局限于开发和运维人员, 还包括了基于专家、爬虫和众包方式协同的 DevOps 知识发现、获取和评价. 本节着重讨论 DevOps 知识库的概念结构, 从技术上来讲, DevOps 知识库可以由多个分布式的知识存储组成. 例如, 其中的公共知识库可以由开源社区维护, 这些存储主要涉及某些诸如 Chef cookbooks 和 Docker 镜像的制品. 私有知识库能够由公司或者部门维护, 主要包括某些特定资源或者不希望对外公开的数据信息. 因此, 实际的知识库是有多个公共和私有知识库组成.

3.1 知识分类

知识分类和关联是实现 DevOps 知识系统化存储管理和使用的必要前提, 本文提出了一组 DevOps 领域知识分类来对当前已有的 DevOps 工具、制品和服务等进行系统的类型划分和关联.

当前 DevOps 涉及的知识类型主要包括: 中间件、基础设施、服务提供方和 DevOps 自动化工具, 本文将上述几种类型作为知识分类中的抽象类型, 这些抽象类型可以包括和划分为多个子类型, 例如在中间件类型中可以包括: 运行环境、Web 服务器等, 如图 4 所示. 每个具体的工具、可重用软件制品和服务等则作为具体的 DevOps 实现与一个或多个抽象类型关联. 图 4 所示的中间件分类以云服务提供方和 DevOps 工具提供方的中间件分类为依据, 如 Heroku, Google, IBM bluemix 和 chef 等.

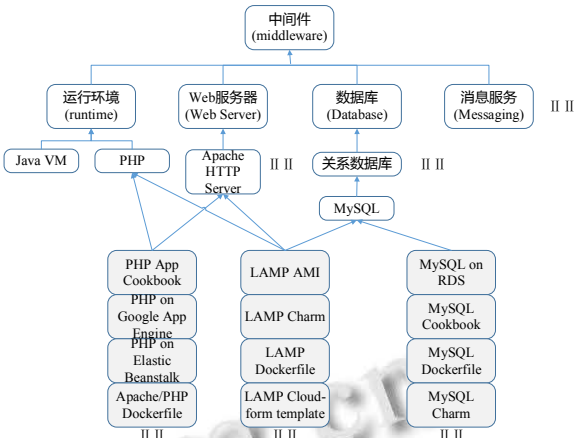

图 4 中间件分类

3.2 DevOps 知识库原型实现

基于第 2 节提出的 DevOps 知识分类和管理方法, 本文设计实现了 DevOps 知识库的原型, 如图 5 所示. 原型系统从 Google App Engine 和 Amazon Web Service 提供的文档和特征描述中获取非结构化的数据, 从 Chef Cookbook 库和 Juju Charm 库中自动发现和获取半结构化数据. 每条获取的知识信息采用单个 YAML 文件的方式存储在 Git 库中, 并基于 3.1 节建立的知识分类方法对每条数据进行相应的类型标记和关联.

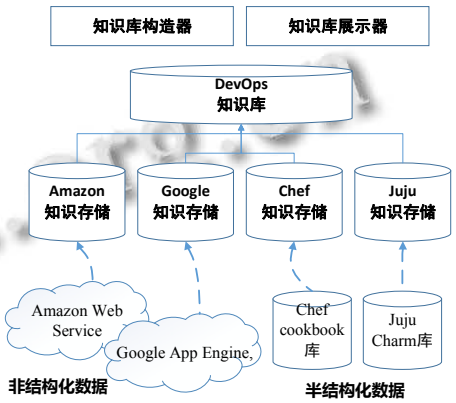

图 5 DevOps 知识库原型实现

原型系统包括一个基于 Node.js 实现的知识库构造器, 通过读取所有知识库内容并采用分级的结构化数据库存储的方式实现知识库的创建和合并. 当前的原型系统包括了大约 4000 个中间件类型的具体实现, 包括: 1430 个 Chef 脚本, 2190 个 Puppet 脚本和 278 个 Juju charms; 除此之外还包括了基础设施、服务提供方和中间件类型的其他众多实现.

原型系统还包括一个基于 Node.js 的知识库展示

© 中国科学院软件研究所 http://www.c-s-a.org.cn

器来对 DevOps 知识内容进行不同形式的呈现, 如 JSON、XML、YAML 等, 从而使知识库能够更好的应用于不同场景. 知识库构建器和知识库展示器均能够通过编程方式或者命令行接口来加以使用.

4 DevOps需求描述

为了实现对 DevOps 知识库的查询, 获取满足 DevOps 需求的方法和工具, 本文提出了基于布尔表达式的应用 DevOps 需求描述方法. 布尔表达式通过谓词逻辑的定义和组合描述应用需求, 并进行 DevOps 知识库的查询. 本文分别从 DevOps 知识涉及的实体、属性和属性值等方面进行表达式的定义.

① ε 表示 DevOps 知识分类包括的所有实体的域, 谓词 $P_{requires}$: ε->{true, false}为每个实体(包括抽象实体和具体实现)赋予一个布尔值. 当给定的实体是一个具体实现(或至少存在一个实现子类)时, $P_{requires}$ 值为 true, 否则为 false.

② ρ 表示所有属性的域, υ 表示所有属性值的域; 谓词 $P_{propertyEq}$: ε×ρ×υ->{true, false}为每个实体、属性和属性值的组合赋予一个布尔值, 如果给定实体具有当前属性的当前值(或者存在子类实体具有当前属性的当前值), 则 $P_{propertyEq}$ 值为 true, 否则为 false.

③ 谓词 $P_{propertyEqGr}$: ε×ρ×υ->{true, false}与 $P_{propertyEq}$ 相似, 但是只有当属性值大于等于当前给定值时, 其值才为 true. $P_{propertyEqGr}$ 可以用来表示版本依赖语义, 例如需要某一特定版本或更高版本的软件.

$P_{propertyEqGr}$ 与 $P_{propertyEq}$ 的语义包含了 $P_{requires}$ 的语义, 这是由于当对实体的某个属性取值存在要求时, 那么对应的实体本身也是需要的. 例如, $P_{requires}$('Middleware/DB/.../MySQL')表达了对 MySQL 数据库的需求, 当对数据库的版本有特定要求时, 表达式进一步细化为:

$$P_{propertyEqGr}(\text{'Middleware/DB/.../MySQL'}, \text{'versions'}, \text{'5.0'})$$

除上述应用相关的 DevOps 需求外, 布尔表达式还可以描述应用无关的约束. 例如, 1)MySQL 必须运行在 Ubuntu 虚拟机上; 2)不允许操作 Amazon 上的任何组件; 3)不允许使用 Chef 进行部署, 诸如此类的约束都可以通过表达式表示为附加性的需求.

附加需求可以通过自定义的附加谓词表达. 例如, 谓词 $P_{excludes}$: ε->{true, false}表达对特定实体是否出现的约束: 不允许在任何应用栈中包含 Amazon 的表达式可以表达为 $P_{excludes}$('Provider/Amazon').

应用系统完整的 DevOps 需求可以通过多个谓词表达式的组合实现. 通过使用整体的布尔表达式来进行 DevOps 知识库的查询, 从而获得满足应用相关需求和应用无关需求的 DevOps 方法、工具、实现等. 本文采用标准化的 Web 服务策略框架(WS-Policy)[13]将表达式表示为策略. 最后, 合并的表达式转换为 WS-Policy 标准形式来实现表达式的处理和查询使用.

需要说明的是, Web 服务策略框架仅仅是作为本文方法实现的技术手段之一, 同样可以采用其他诸如约束求解器、规则引擎(例如 java drools)等技术手段加以实现. 在后续工作中, 我们将尝试采用多种技术端进行系统的实现, 并从功能覆盖、表达能力和执行效率等多方面对不同的技术实现加以分析比较.

以第 1 节中的 WordPress 为应用案例, 其最小需求集合可以表示为如下:

$$WordPress_{minimum} = P_{propertyEqGr}(\text{'Middleware/DB/.../MySQL'}, \text{'versions'}, \text{'5.0'}) \wedge P_{propertyEqGr}(\text{'Middleware/Runtime/PHP'}, \text{'versions'}, \text{'5.2.4'}) \wedge P_{requires}(\text{'Middleware/Web Server'})$$

基于 $WordPress_{minimum}$ 可以在 DevOps 知识库中查找是否存在合适的实现来支持 WordPress 的运维, 并通过从知识库中获得的实现构成多个可选的运维策略. 基于 $WordPress_{minimum}$ 从知识库原型系统中找到的实现包括:

① LAMP AMI (middleware) 运行在 Amazon EC2 (provider)上;

② LAMP Charm (middleware) 运行在 Amazon EC2 (provider)的 ubuntu (infrastructure)操作系统上;

③ PHP 应用脚本(middleware)部署在 Amazon EC2 (provider)的 ubuntu (infrastructure)操作系统上, 同时 MySQL charm (middleware)部署在 Amazon EC2 (provider)的 ubuntu (infrastructure)操作系统上;

④ PHP on Elastic Beanstalk (middleware) 部署在 Amazon Elastic Beanstalk (provider)上, 而且 MySQL on RDS (middleware) 部署在 Amazon RDS (provider)上;

⑤ PHP on Google App Engine (middleware) 部署在 Google App Engine (provider)上, MySQL charm (middleware) 部署在 Amazon EC2 (provider)的 ubuntu

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(infrastructure)上.

上述为 WordPress 部署和运维可选的一些具有代表性的方案. 下一步可以通过增加附加约束的方法对可选方案进一步细化和过滤, 例如要求 MySQL 的实现具有可扩展能力.

$WordPress_{minimum}=$

$P_{propertyEqGr}(\text{'Middleware/DB/.../MySQL','versions','5.0'})$

$\wedge$

$P_{propertyEq}(\text{'Middleware/DB/.../MySQL','scaling','master-slave'}) \wedge \ldots\ldots$

类似的, 表达式可以进一步增加约束来进行 DevOps 方案的细化, 例如增加运行时环境弹性能力需求 $P_{propertyEq}(\text{'Middleware/Runtime/PHP','elastic','ture'})$ 和对服务提供方的约束 $P_{excludes}(\text{'Provider/Amazon'})$.

5 结语

当前 DevOps 作为新兴范型是实现有效的、无缝衔接的软件自动化管理的有效途径. DevOps 与云计算一起能够实现基础设施资源的快速按需供给. 当前规模不断增涨的可重用 DevOps 制品和云服务使应用设计和开发人员有大量机会进行 DevOps 的尝试和实践. 但是, 由于大量可选 DevOps 方法和工具的存在使得如何进行有效的决策和选择成为亟待解决的问题, 而且这些知识分散在不同的社区和来源中. 针对这一问题, 本文提出了一套完整的 DevOps 知识管理方法. 方法首先针对一组可访问的知识源进行多种方式的知识获取、组织、转换和存储; 然后提出了 DevOps 知识分类方法, 并进行了 DevOps 知识库原型系统的设计和实现; 最后基于谓词逻辑提出了 DevOps 需求的描述机制以及基于 WS-Policy 实现需求的转换和使用. 未来工作将进一步改进 DevOps 知识管理方法, 包括: 提供更多的 DevOps 知识库信息, 以及基于自动化爬虫的知识评价方法.

参考文献

1 Humble J, Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010.

2 Humble J, Molesky J. Why enterprises must adopt DevOps to enable continuous delivery. Cutter IT Journal, 2011,24.

3 Walls M. Building a DevOps Culture. O'Reilly Media, Inc., 2013.

4 Wettinger J, Breitenbücher U, Leymann F. DevOpSlang – Bridging the Gap Between Development and Operations. Proc. of the 3rd European Conference on Service- Oriented and Cloud Computing (ESOCC), 2014.

5 Hüttermann M. DevOps for Developers. Apress, 2012.

6 Nelson-Smith S. Test-Driven Infrastructure with Chef. O' Reilly Media, Inc., 2013.

7 Turnbull J, McCune J. Pro Puppet. Apress, 2011.

8 Uphill T. Mastering Puppet. Packt Publishing Ltd., 2014.

9 Mell P, Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology, 2011.

10 Kächele S, Hauck FJ, Spann C, Domaschka J. Beyond IaaS and PaaS: An extended cloud taxonomy for computation. Storage and Networking, 2013.

11 Trinkle P. An Introduction to Unsupervised Document Classification, 2009.

12 Dustdar S, Bhattacharya K. The social compute unit. Internet Computing, IEEE, 2011, 15(3): 64–69.

13 W3C, Web Services Policy Framework (WS-Policy), Version 1.5, 2007.

14 Jenkins, http://jenkins-ci.org.

15 Docker, http://www.docker.com.

16 Juju. https://juju.ubuntu.com.

17 Juju charms. https://jujucharms.com.

18 Puppet modules, https://forge.puppetlabs.com.

19 Amazon EC2. http://aws.amazon.com/ documentation/ ec2

20 WordPress. http://wordpress.org.

© 中国科学院软件研究所 http://www.c-s-a.org.cn