OpsFlow：一种面向 DevOps 的应用自动化部署引擎*

李　超[1]　花　磊[2]　宋云奎[2]
（1. 新华通讯社中国经济信息社有限公司　北京　100803）（2. 中国科学院软件研究所　北京　100190）

摘　要　当前互联网应用飞速发展，用户需求随之也快速变化，应用需要快速迭代与频繁发布，因此，自动化部署是缩短应用发布周期的关键技术之一。DevOps 旨在填补开发与运维人员间的鸿沟，实现快速自动化的应用部署。本文面向 DevOps 提出了一种高效动态的应用部署引擎 OpsFlow，整合应用各层次多种技术、服务和构件以应对各类应用组件的部署问题，从而实现高效的组件组合和自动化的应用组件部署；进而，设计了自动化部署框架，将应用部署分为构建、设计、运行等三个阶段，以自动生成 OpsFlow；最后，通过实验评价了生成 OpsFlow 的开销，并且使用微服务应用的实例研究验证了方法的有效性。

关键词　应用部署引擎；应用配置；开发运维一体化；微服务架构
中图分类号　TN830.1　　**DOI**：10. 3969/j. issn. 1672-9722. 2019. 01. 040

OpsFlow: Automatic Application Deployment Engine for DevOps

LI Chao[1]　HUA Lei[1]　SONG Yunkui[2]
（1. Xinhua News Agency China Economic Information Agency Co., Ltd., Beijing　100803）
（2. Institute of Software Chinese Academy of Sciences, Beijing　100190）

Abstract　Customers' requirements change rapidly with the development of Internet-based applications, so applications have the requirements of rapid iteration and frequent delivery. DevOps aims at bridging the gap of development and operation to implement the automatic deployment of applications for decreasing the time of releasing software. This paper proposes a method of generating a dynamic deployment engine OpsFlow for microservice-based applications, and designs a technical framework to implement OpsFlow. Finally, the overhead of generating OpsFlow with experiments is evaluated, and a case study of a microservice-based application to validate OpsFlow is given.

Key Words　application deployment engine, application configuration, DevOps, microservices architecture
Class Number　TN830.1

1　引言

近年来，互联网公司业务迅速扩展，用户需求频繁变化，传统基于单体架构的软件系统难以适应互联网技术的快速发展。因此，微服务架构应运而生，将单一应用划分成众多规模较小、功能专一的服务。其中，每个服务对具体业务逻辑进行封装构建，独立运行于生产环境中，通过服务模块之间的相互配合以对外提供服务[1]。微服务架构由于模块自治性强，可以满足互联网应用变化快、模块独立更新的需求。但是微服务模块数量众多、交互复杂，增加了运维的难度与成本，如何应对微服务快速频繁的更新与部署已成为微服务架构应用亟需解决的主要问题之一。

为了实现关注点或任务的分离，将微服务的开发测试和运行维护过程进行严格区分，使得不同项目组具有不同的目标与流程。开发人员的目标是快速将变更实现到生产环境，而运维人员的目标是保持生产环境的稳定，这成为快速频繁发布软件的障碍。开发和运维人员之间的协作通常通过手工

* **收稿日期**：2018 年 7 月 13 日，**修回日期**：2018 年 8 月 26 日
基金项目：金融信息平台“新华 08”金融财经数据云服务平台科研专项“应用系统研发运营部署一体化管控平台研究与实践”（编号：TC16037XX/01）资助。
作者简介：李超，男，硕士研究生，工程师，研究方向：系统架构、微服务和容器管理、DevOps。花磊，男，博士研究生，高级工程师，研究方向：系统架构、微服务、容器、DevOps。宋云奎，男，硕士，助理研究员，研究方向：DevOps、应用自动化部署。

进行，该过程低效且易错，因此将更改应用、增加特性和修复错误实现到生产环境中需要投入大量的人力和时间。为了快速响应用户需求以及微服务应用的不断变化，软件企业需要应对快速频繁的微服务应用发布，填补开发和运营之间的鸿沟。DevOps是应对微服务应用开发与运维的新兴技术，可以实现开发和运维两个部门间的有效协作[2]。微服务应用实例不仅需要部署到开发环境以运行测试用例或者检查更新代码，也需要不断部署和重新部署到生产环境以提供服务。由于应用会频繁部署到不同环境，实现高效自动化的应用部署至关重要。同时，应用需要从简单的开发环境到复杂的分布式生产环境无缝对接，部署逻辑需要具有可移植性。

DevOps社区提供开源工具（如Chef[3]，Puppet[4]，Ansible[5]）来支持部署过程自动化以实现软件的持续交付；在中间件与应用层，可以获取可重用的构件（如脚本、模板）以实现部署自动化；在基础设施层，云计算平台为应用按需提供底层物理资源自动化部署API。因此，本文结合各层次多种技术、服务和构件以应对各类微服务应用的部署问题，从而实现高效的微服务组件组合和自动化的微服务应用组件部署。本文面向DevOps，提出了一种高效动态的应用部署引擎OpsFlow，设计与实现了技术框架以自动生成OpsFlow，最后，通过实验评价了生成OpsFlow的开销，并且使用微服务应用的实例研究验证了方法的有效性。

2　微服务应用自动化部署

2.1　应用自动化部署引擎OpsFlow

应用部署引擎OpsFlow目标是组合多种自动化部署技术以应对异构应用与部署环境，从而实现高效自动化的微服务应用部署。OpsFlow作为可移植可定制的部署逻辑包，为用户或外部系统提供API以便于部署应用实例，同时也提供管理逻辑对已部署的某些应用组件进行扩展，具有以下功能：

1）为特定应用实例指定编程语言以生成可移植包，使得应用实例在本地主机开发环境、本地服务器测试环境、远程云计算生产环境等不同的环境中运行；

2）生成可执行构件（如脚本、编译程序）以实现“原子”部署操作，其实现形态是安装和配置软件包的部署脚本，或是通过调用云服务接口来使用虚拟资源的代码；

3）编排多个部署可执行程序以形成云应用部署模板。部署可执行程序负责执行部署计划，协调部署操作，并创建特定应用拓扑；

4）提供公开API以触发部署计划的执行或创建应用拓扑，如图1所示API可以被用户或外部系统调用，例如，高层调度器可以根据当前负载调用OpsFlow接口以创建更多应用实例。

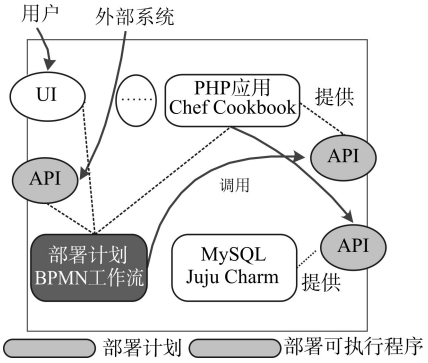

图1　OpsFlow系统架构

根据应用拓扑动态生成定制的OpsFlow引擎具有以下优点：

1）只包含实现部署操作所必需的部署可执行程序，具有较少的软件规模、资源消耗和配置操作，以及较高的性能。

2）打包所需要的部署可执行程序，以自包含方式部署特定应用，彼此间相互独立。由于不依赖于集中的中间件，避免了单一故障点，从而具有更高的可靠性。

3）使用开源生态系统提供的可重用构件（如Chef、Juju）自动生成部署计划，实现部署可执行计划的编排和执行，开发人员只需调用所需的API而无需手动开发粘连代码。

2.2　OpsFlow部署引擎生成方法

OpsFlow的动态生成方法如图2所示，分为设计、构建和运行等三个阶段，遵循通用规范使用不同技术实现。

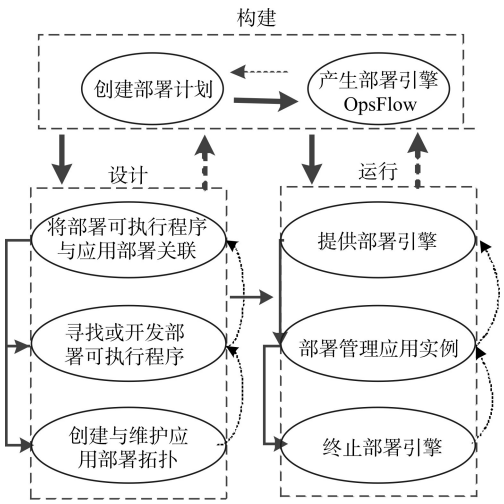

图2　OpsFlow部署引擎生成方法

1)设计阶段:创建和维护应用拓扑模型;根据基础设施、中间件和应用组件选择部署可执行程序,这些可执行程序可以定制开发,也可以使用现有构件(如Chef, Juju, fog, Jclouds);将所需的部署可执行程序关联到应用拓扑。其中,部署可执行程序,既可以独立开发,也可以重用现有开源社区提供的构件。对于现有开源部署构件(如Chef Supermarket,Docker Hub),由于每个开源社区独立维护存储库,并且提供有限的搜索能力,因此缺乏全面的知识,难以选择合适的构件,本文通过整合知识库的元模型来解决该问题。部署需求来自于应用拓扑,可以通过需求参考文件来表达部署可执行程序的需求以及与其他部署可执行程序的关联。

2)构建阶段:创建部署计划,即部署可执行程序,编排所有必需的部署可执行程序,以创建特定应用拓扑模型的实例。部署计划可以手工完成,也可以使用计划自动生成技术,如使用BPEL引擎自动生成应用部署的工作流程;在创建部署计划时,所有必需的部署可执行程序都已就绪,生成OpsFlow部署引擎。

3)运行阶段:生成可定制的OpsFlow引擎;根据建模的应用拓扑,提供API以创建部署计划,并部署与管理应用实例;如果不需要部署和管理更多的应用实例,则终止OpsFlow引擎。

2.3 OpsFlow部署引擎生成框架

OpsFlow部署引擎生成框架如图3所示,包括设计、构建和运行等三个阶段,具体包括以下几个组件。

应用拓扑模型的建模环境:建立和维护应用拓扑模型,并将所需的部署可执行程序关联到这些拓扑模型中的组件和依赖项。

部署计划产生器:动态自动生成部署计划,能够处理创建的应用拓扑模型。为了将建模环境与计划生成器分离,本文使用基于标准的建模方法(如TOSCA[10])。为了达到自定义目标而改进部署计划,则需要提供相应的部署计划开发环境,本文将Eclipse BPEL Designer作为BPEL工作流环境。

API生成器:生成部署可执行程序,隐藏和抽象实现的技术细节,创建通过API访问的部署计划,实现在不修改源码的情况下包装部署可执行程序。例如,如何调用可执行文件,如何传递输入参数,以及如何收集输出数据。如果涉及多个特定应用的部署可执行程序,手工为单个应用包装可执行文件以实现API耗时费力,那么就需要自动化创建API。由于使用多种自动化技术来部署异构应用,需要手工修改与细化部署计划。特别是,当使用工作流建模语言(如BPEL),集成各种技术非常复杂且容易出错,因此,将这些技术特点及其调用机制封装在API中,就可以使用HTTP等标准化通信协议来调用。部署计划生成器不需要了解各种技术的调用机制细节,而只需了解如何调用API以支持自动生成部署计划。

部署引擎包装器:生成自包含的OpsFlow引擎,作为可移植可执行包,在运行时部署应用实例。API生成器和OpsFlow引擎包装器都可以基于API化框架实现,比如Any2API[11]。

OpsFlow引擎运行时环境:根据OpsFlow引擎的打包格式,使用API来部署和管理应用实例。同时,通过使用可移植的虚拟化方法,可用于在不同的环境中打包和执行OpsFlow引擎。

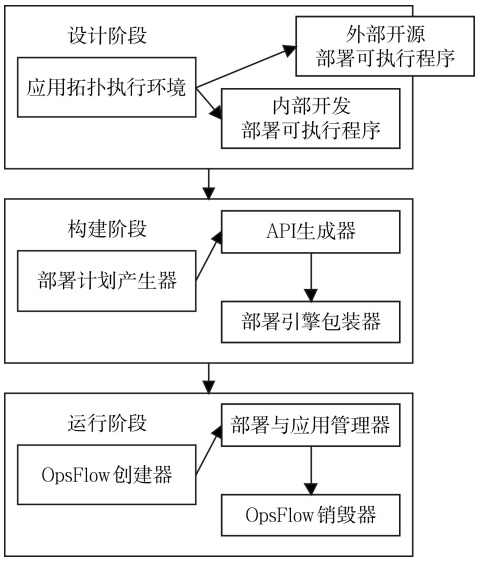

图3 OpsFlow部署引擎生成框架

2.4 OpsFlow部署引擎生成框架实现

本文基于TOSCA实现了原型系统,其中的工具链用以应对可移植性和可扩展性,涵盖了设计、构建和运行等三个阶段。在设计阶段,建模工具提供了可扩展的基于TOSCA标准的建模环境以改善可移植性。并且,使用Winery[12]实现应用拓扑建模,以管理不同类型的组件和依赖关系。

在构建阶段,本文使用OpenTOSCA作为部署计划生成器以动态为特定应用拓扑模型生成部署计划,从而满足应用拓扑的特定部署需求。对于常见的通用组件类型(如MySQL数据库和Apache服务器),部署计划生成器能够自动生成部署计划,而无需手工进行细化。

在运行阶段,基于工作流建模语言(如BPMN或BPEL)生成的部署计划,需要在构建时生成相

应的API以暴露部署可执行程序所需功能，这些功能关联到应用拓扑模型的组件和依赖项。本文使用Any2API框架生成这些API，例如，为部署可执行程序生成基于SOAP/WSDL的Web服务API，以支持在BPEL中实现的部署计划，从而调用和处理底层的可执行程序。另外，当使用脚本语言（如Python或Ruby）与相关库（rest-client）连接一起实现部署计划时，生成基于RESTFul的Web API。由于生成的部署计划也是通过调用API执行的，基于API的递归聚合可以支持众多不同的拓扑模型。最后，Docker容器作为可移植的虚拟化技术在构建时使用Dockerfile包装OpsFlow引擎，同时使用Any2API执行包装操作。

3　实例研究

3.1　实验环境

本文以典型的网上商店应用作为实例来分析应用部署所面临的问题。图4描述了网上商店应用架构，采用混合云部署方式，由左边部署在Amazon EC2的应用和右边部署在OpenStack的数据库构成，以避免将敏感数据暴露给公有云服务提供商。PHP应用部署在Apache服务器以展示用户界面并执行业务逻辑，数据库采用MySQL主/从模式以提高应用的可伸缩性与高可用性。应用由不同组件构成，采用不同部署技术。其中，Web应用使用Unix Shell脚本部署，PHP模块和Apache服务器使用Chef部署，Mysql数据库使用Juju部署。同时，在基础设施层（包括虚拟机和网络配置）使用Fog部署，与云计算平台（Amazon、OpenStack）通过API进行交互。

实现该应用部署涉及到多种类型的部署脚本和构件，需要考虑众多技术细节和差异，除了特定应用的Shell脚本外，Chef和Juju都需要特定部署引擎支持。此外，应用拓扑以多云方式部署，即Amazon作为公有云服务提供者，而OpenStack作为私有云管理平台。

通用的自动化部署框架（如OpenTOSCA[6]，Terraform[7]，Brooklyn[8]）通过引入统一的元模型进行抽象以实现不同部署可执行程序的编排，使得不同方法使用相应的元模型，但需要手工对特定部署模块进行适配开发而不能自动实现。同时，由于通用的部署引擎需要支持多种部署方法，通常是重量级模块，维护复杂度高。

3.2　开销分析

实验使用三种公开的Chef cookbooks来部署Apache服务器、PHP运行时环境和MySQL数据库，实现两个Ruby脚本在Amazon EC2云基础设施上部署虚拟服务器，使用Amazon关系数据库服务（RDS）部署MySQL数据库实例。

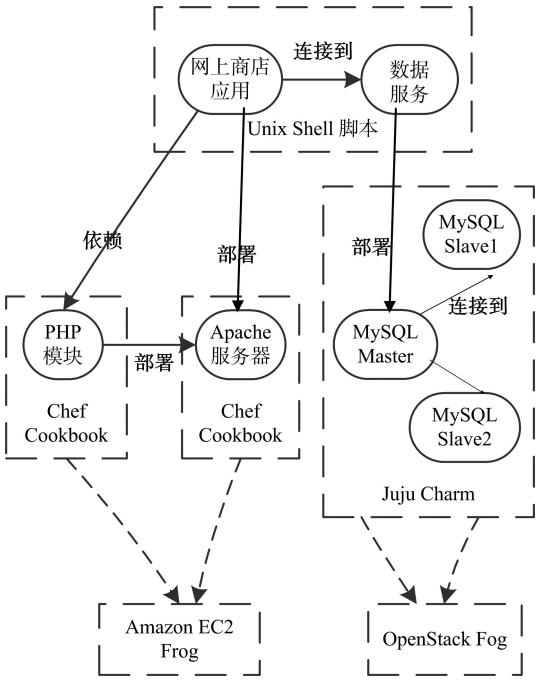

图4　部署结构图

该实验系统部署在一台虚拟机（4个2.8 GHz主频64位的虚拟CPU，4 GB内存），使用Virtual Box Hyperviso，在Debian Linux容器中完成部署可执行程序的处理和调用。本文监测容器层的所有度量，关注部署可执行程序和API实现相关的度量，执行每个部署可执行程序20次（其中，10次有API实现，10次没有API实现）。每组10次执行中的5次是初始执行（在干净的部署环境中运行，而没有执行任何操作），另外5次是后续执行。生成API实现的平均时间是构建阶段的开销，为8s~33s，其中包括检索特定部署可执行程序的依赖项。由于对生成的API进行了简化，运行时具有较少的执行时间和较小的内存开销。此外，当直接使用简单的部署可执行程序，编排组件会减少生成API实现的复杂性。因此，资源的总体消耗取决于所选的编配方式，同时可以复用API实现在不同的远程环境中运行可执行程序，以减少大型运行环境的开销。本文从设计、构建和运行等三个阶段对应用部署开销进行评估，实验结果如下：

1）设计阶段：生成API所需的时间，实验结果如图5所示；

2）构建阶段：部署可执行程序所需时间，实验结果如图6所示；

3)运行阶段:可执行程序运行所需内存,实验结果如图7所示。

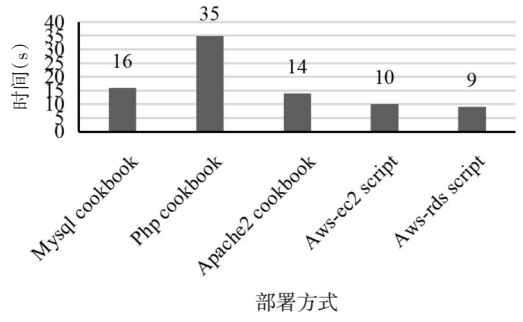

图5　生成API实现时间

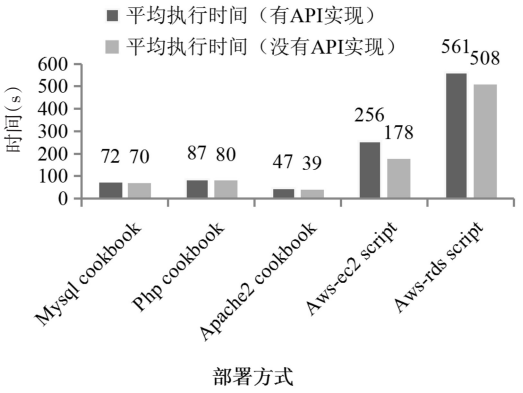

图6　部署执行时间

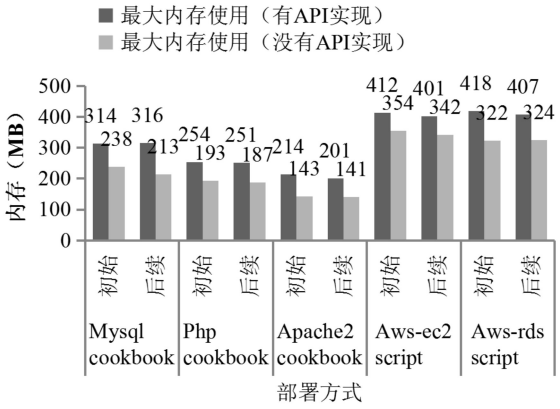

图7　运行内存使用

4　相关工作

与传统的通用部署引擎不同,OpsFlow在运行时不依赖于集中的中间件。论文[13]利用可扩展的通用部署引擎来部署和管理各种应用,通过减少生成定制引擎的步骤来减少构建时的开销。与OpsFlow相比有以下几个缺点:1)通用部署引擎提供的API并不适合所有的部署场景,需要实现多种部署技术,具有较高的复杂性,因而难以维护和扩展;2)通用部署引擎通常采用集中式中间件,存在单点失效的问题,因此需要维护其可靠性。3)使用通用部署引擎通常需要领域知识,通过二次开发来包装现有的API,从而实现多应用组件部署的编排。4)通用部署引擎不是专门针对给定的应用拓扑,因此运行开销较高。Bootware[14]首先创建部署引擎,然后用其部署应用实例,该方法使用通用部署引擎,而OpsFlow是面向特定应用动态生成的部署引擎。论文[15]将不同类型的部署可执行程序转换为基于TOSCA标准的构件,便于现有多样化开源生态系统所提供内容的复用。该方法虽然能够有效实现设计阶段和构建阶段的部署自动化,但是在运行阶段需要通用部署引擎来部署和管理应用拓扑。云提供商提供了建模和编排工具[16],同时提供了PaaS(Platform-as-a-Service)服务[17],调用该服务可以部署和管理应用实例,而不需要显式建模应用拓扑。文献[18]提出了基于Xen的云平台自动部署框架。文献[19~20]提出了PM、VM和应用等三个层次的云应用部署的描述语言。但以上方法难以实现多云或混合云部署,将应用或部分应用自动化迁移到不同的提供者非常困难。同时,API是由提供者预先定义的,并且必须通过开发自定义的粘合代码来手动包装,因此不适用于特定部署场景。

5　结语

当前互联网应用的用户需求快速变化,高效自动部署技术是缩短软件发布周期的关键。DevOps技术有助于实现快速自动化部署过程,通过不断迭代交付微服务来缩短发布周期。本文面向DevOps,提出了一种高效动态的应用部署引擎OpsFlow及其生成方法,设计并实现了技术框架以支持所提出方法,最后,实验结果表明,生成部署引擎具有较小的开销。

参考文献

[1] THONES J. Microservices[J]. IEEE Software, 2015, 32(1):116-120.

[2] HUTTERMANN M. DevOps for Developers[M]. New York:Apress, 2012.

[3] TAYLOR M, VARGO S. Learning Chef: A Guide to Configuration Management and Automation[M]. Boston: O'Reilly Media, 2014.

[4] UPHILL T. Mastering Puppet[M]. Birmingham: Packt Publishing Ltd., 2014.

[5] MOHAAN M, RAITHATHA R. Learning Ansible[M]. Birmingham: Packt Publishing Ltd., 2014.

[6] BINZ T, BREITENBUCHER U, HAUPT F, et al. Open-

(下转第247页)

dao:China University of Petroleum,2008.
[11] L.Kobbelt. Sqrt3 subdivision[J]. Proceedings of Acm Siggraph,2000,18(1):103-112.
[12] AlexVlachos, JorgPeters, ChasBoyd, JasonL, Mitchell. Curved PN-Triangle[J]. University of Florida, Symposium on Interactive 3d Graphics,2001:159-166.
[13] 郭栋梁,聂俊岚,田茂春,等. 面向移动终端的PN三角形自适应细化管线[J]. 燕山大学信息科学与工程学院,2014,26(1):30-33.
GUO Dongliang, NIE Junlan, TIAN Maochun, et al. Adaptive PN Triangle Refinement Pipeline for Mobile Terminals [J]. Information Science and Engineering, Yanshan University, 2014, 26(1):30-33.
[14] 贺星. 无预细分的GPU曲面细分算法[D]. 杭州:浙江大学,2013.
HE Xing. No Pre-Subdivision of the GPU Surface Subdivision Algorithm [D]. Hangzhou: Zhejiang University, 2013.
[15] 陈驰,吴志红. PN三角形在基于GPU曲面细分中的应用[J]. 四川大学, 2015,13(2):297-301.
CHEN Chi, WU Zhihong. Application of PN triangle in GPU based tessellation[J]. Sichuan University,2015,13(2):297-301.
[16] 李继,吴丽娟. 曲面细分模式的分类研究[J]. 沈阳师范大学学报(自然科学版),2009,54-58.
LI Ji, WU Lijuan. Classification on Surface Subdivision [J]. Shenyang Normal University (Natural Science), 2009,54-58.
[17] 刘云华. 曲面细分以及三种经典细分模式的分析研究[D]. 西安:长安大学,2011.
LIU Yunhua. Surface subdivision and three classic subdivision model analysis[D]. Xi'an: Changan University, 2011.
[18] 刘扬. 基于B样条的三角网格细分曲面造型技术的研究[D]. 哈尔滨:哈尔滨理工大学,2011.
LIU Yang. Research on Triangular Mesh Subdivision Surface Modeling Technology Based on B Spline[D]. Harbin: Harbin University of Science and Technology,2011.

(上接第 194 页)

TOSCA-A Runtime for TOSCA-based Cloud Applications [C]//Proceedings of the 11th International Conference on Service-Oriented Computing, 2013: 692-695.
[7] HashiCorp, Inc. Terraform: Write, Plan, and Create Infrastructure as Code[EB/OL]. [2018-12-18]. https://terraform.io.
[8] Apache Foundation. Brooklyn: Your applications, any clouds, any containers, anywhere [EB/OL]. [2018-12-18]. https://brooklyn.incubator.apache.org.
[9] Object Management Group, Inc. Business Process Model and Notation (BPMN) Version 2.0 [EB/OL]. [2018-12-18]. http://www.bpmn.org/.
[10] OASIS organization. Topology and Orchestration Specification for Cloud Applications Version 1.0 [EB/OL]. [2018-12-18]. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.
[11] WETTINGER J, BREITENBUCHER U, LEYMANN F. Streamlining APIfication by Generating APIs for Diverse Executables Using Any2API[C]//Proceedings of the International Conference on Cloud Computing and Services Science, 2015:216-238.
[12] KOPP O, BINZ T, BREITENBUCHER U, LEYMANN F. Winery-A Modeling Tool for TOSCA-based Cloud Applications[C]//Proceedings of the 11th International Conference on Service-Oriented Computing, 2013:700-704.
[13] LU H, SHTERN M, SIMMONS B, SMIT M, et al. Pattern-based Deployment Service for Next Generation Clouds[C]//Proceedings of the IEEE 9th World Congress on Services, 2013:464-471.
[14] VUKOJEVIC-HAUPT K, KARASTORYANOVA D, LEYMANN F. On-demand Provisioning of Infrastructure Middleware and Services for Simulation Workflows[C]//Proceedings of the 6th International Conference on Service-Oriented Computing and Applications, 2013: 91-98.
[15] ANDRIKOPOULOS V, REUTER A, GOMEZ SEAEZ S, et al. A GENTL Approach for Cloud Application Topologies [C]//Proceedings of European Conference on Service-Oriented and Cloud Computing, 2014:148-159.
[16] ROSENER T. Learning AWS OpsWorks[M]. Birmingham: Packt Publishing Ltd., 2013.
[17] COUTERMARSH M. Heroku Cookbook[M]. Birmingham: Packt Publishing Ltd., 2014.
[18] ZHANG Y, LI Y, ZHENG W. Automatic software deployment using user-level virtualization for cloud-computing[J]. Future Generation Computer Systems, 2013, 29(1):323-329.
[19] SRIRAMA S N, LURII T, VIIL J. Dynamic Deployment and Auto-scaling Enterprise Applications on the Heterogeneous Cloud[C]//Proceedings of the IEEE International Conference on Cloud Computing, 2016:927-932.
[20] SEINTURIER L, MERLE P, FOURNIER D, et al. A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures[J]. Software Practice and Experience, 2012, 42(5):559-583.