Journal of Engineering and Applied Sciences 12 (9): 2396-2401, 2017
ISSN: 1816-949X

Hybrid System Orchestration with TOSCA and Salt

Ales Komarek, Jakub Pavlik and Vladimir Sobeslav
Faculty of Informatics and Management, University of Hradec Kralove,
Hradec Kralove, Czech Republic

Abstract: As the cloud services and container based micro-services are starting to get the production workload within enterprise IT infrastructures the need for application portability and effective orchestration across various virtualized, containerized, hardware and legacy platforms is getting apparent. With application systems spawning multiple platforms the network functions as routers, load balancers or firewalls are becoming integral part of production application stacks. In this study, we provide a comparison of state-of-the-art orchestration tools. The main goal of the work is to create PoC environment that can test execution of sample application architecture model at various deployment scenarios: hybrid multi-cloud and cloud-container deployment setups.

Key words: TOSCA, micro-service, cloud service, heat, OpenStack, docker, topology, workflow, orchestration

INTRODUCTION

Orchestration is the automation of tasks involved with managing and coordinating complex resources. The ultimate goal of IT orchestration is to automate configuration and coordinated management of all resources in cloud service environments or container platforms. This task involves inter-connecting processes running across heterogeneous systems in multiple locations, these service usually have proprietary interfaces.

There's been a lot of efforts in cloud management research (Binz *et al*., 2012; Antonescu *et al*., 2012; Juve and Deelman, 2011; Liu *et al*., 2011) but also the industry started (IBM, 2013) orchestration tools and specifications to describe service topologies. Some are independent open source projects supported by only a part of industry or academic organizations. There are projects with support from large part of industry and they become official standards (TC. O.T, 2012).

Orchestration tools often use model infrastructure to define the application topology and deployment process. The orchestration engine uses this definition as an input and translates it into a set of tasks that interact with the underlying infrastructure which allow it to create and manage various resources: virtual machines, configure and install application stacks, etc.

The model infrastructures can be saved in form of simple code definition. Then, we have the ability to design, implement and deploy complete infrastructures from that model. The model infrastructures can be brought to the known software best practices by collaboration of software developers who deliver flexible application code changes and IT infrastructure administrators who deliver environment setup to create complete devops environment. The ability to treat the infrastructure model as an application code and the use of the software development tools allows developers to rapidly deploy infrastructures in same pace as applications.

To ensure interoperability of the model infrastructures several formal standards were devised to provide model definition standards. These types of formal language are called Domain Specific Languages (DSL). HOT (Heat Orchestration Template) is standard language to orchestrate resources in OpenStack cloud. It describes a topology of OpenStack based resources and their relationships. The TOSCA (Topology and Orchestration Specification for Cloud Applications) is standard language to describe a topology of cloud based components their relationships and the processes to manage them (TC, 2012). This effort is driven by OASIS (Organization for the Advancement of Structured Information Standards) and is sponsored by major companies of the ICT sector such as IBM, CA technologies, Hewlett-Packard, Red Hat, SAP and others.

The orchestration methods can be either declarative (functional) or imperative (procedural). The difference

Corresponding Author: Ales Komarek, Faculty of Informatics and Management, University of Hradec Kralove, Hradec Kralove, Czech Republic

between the declarative and imperative approach is essentially "what" versus "how". The declarative approach aims on what the target state should be on other hand, the imperative focuses on how the infrastructure has to be changed to meet the target state (TC, 2012).

The declarative approach defines the desired state and the system executes what needs to happen to achieve that desired state. Imperative defines specific commands that need to be executed in the appropriate order to end with the desired conclusion.

Within the service orchestration both approaches are used. The declarative orchestration is good to enforce desired system state over long periods of time. The imperative approach enables process runs to be specified in more complex and introduce task not directly related to the actual service management.

Orchestration tools can be divided into several categories according to the type of targeted service to manage. The following list shows the main focus areas of configuration management tools.

Infrastructure centric orchestration: Orchestration tools for management of IaaS cloud services along with several SaaS services as database, message in declarative way for example; OpenStack heat, AWS CloudFormation.

Platform centric orchestration: Orchestration tools for management of PaaS cloud services in declarative way, for example; CloudFoundry, OpensShift.

Container centric orchestration: Orchestration tools for management container micro services for example; Mesos, Kubernetes.

Software configuration management: Orchestration tools for management of software configurations in declarative way for example; salt, puppet, ansible and chef (Anonymous, 2015).

Platform agnostic orchestration: Orchestration tools for management of any service in declarative way and enabling imperative workflows, for example; TOSCA/cloudify, TOSCA/heat, terraform.

A team from tcp cloud a.s. and University of Hradec Kralove carried out a Proof of Concept (POC) project to investigate the possibilities to deploy selected infrastructure across various deployment environments. The activities of this PoC included the state-of-the-art analysis of orchestration tools and the selection of the appropriate technologies for carrying out the use case testing.

MATERIALS AND METHODS

State-of-the-art orchestration: The following study provides details of selected orchestration tools. Due to the scope of the study PaaS orchestrating solutions like cloud foundry or openshift and software configuration management tools like Salt, Puppet, Ansible and Chef were excluded from the list of orchestration tools. We have chosen a only selected sample for each orchestration category representing the category concept. The software configuration management is used only for the configuration not the orchestration workload.

Infrastructure as service brought by simple model enforced by an execution service and is tightly coupled to particular cloud service provider. The first company to provide infrastructure orchestration is Amazon with its CloudFormation declarative orchestration standard (Wittig, 2015). The heat project is part of OpenStack platform has basic compatibility with CloudFormation.

AWS CloudFormation: AWS CloudFormation provides a declarative template-based infrastructure as Code model for managing AWS infrastructure deployments. AWS CloudFormation can work Amazon EC2, EBS, Amazon SNS, ELB and auto scaling resources (Wittig, 2015). All resources and dependencies are declared in a JSON encoded template. A collection of resources is called a stack. The service itself is closed source and internal architecture is hidden.

OpenStack heat: Heat implements an orchestration engine that can manage multiple composite cloud applications using YAML based resource templates. Heat can work with any OpenStack based resources. It implements just simple installation, update and delete workflows (Fig. 1).

Managing container cluster is defined by simple model enforced by management service at each container host node. Provides limited options for integrating advanced networking and storage functions. Containers are separated software processes and have different lifecycle management than virtualized or paravirtualized virtual machines. Containers do not provide any

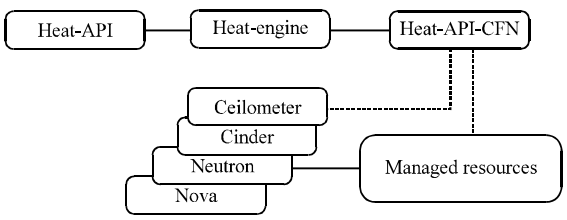

Fig. 1: Heat orchestrator architecture

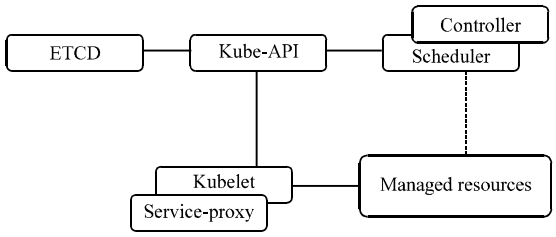

Fig. 2: Kubernetes orchestrator architecture (Kubernetes, 2017)

persistent storage on its own, provide specific service endpoints. The target software services are operated in foreground blocking mode and if service stops, the entire container stops as well.

Kubernetes: Kubernetes is an open source container cluster manager that provides a "platform for automating deployment, scaling and operations of application containers across clusters of hosts" (Kubernetes, 2017). Reuses Docker based LXC containers and adds level of cluster management for containers (Fig. 2).

Platform agnostic orchestration: This family of orchestration tools is not bound to any specific platform or technology. It uses client libraries and services APIs to enforce state on various software and hardware resources. The orchestrators use plugin approach to control various resources that can be managed by application API.

Cloudify: Cloudify is an open source cloud orchestration framework. Cloudify models complex infrastructures and automate their entire life cycle, including deployment on any cloud or data center environment, monitoring all aspects of the deployed application, detecting issues and failure, manually or automatically remediating them and handle ongoing maintenance tasks. Cloudify uses DSL based on simplified version of TOSCA simple profile. Cloudify executes TOSCA blueprints by reading resources from templates and mapping them to operations in cloudify plugins. Cloudify uses asynchronous orchestrators to enforce blueprint deployments and keep the state of deployed systems separate (Fig. 3).

INDIGO DataCloud: Project INDIGO DataCloud aims at developing a data and computing platform deployable on multiple hardware and provisioned over hybrid infrastructures (Anonymous, 2016). It uses TOSCA simple profile to model compute, storage and network resources. The management is realized by heat translator converting TOSCA Model to HOT and heat service performing the actual orchestration.

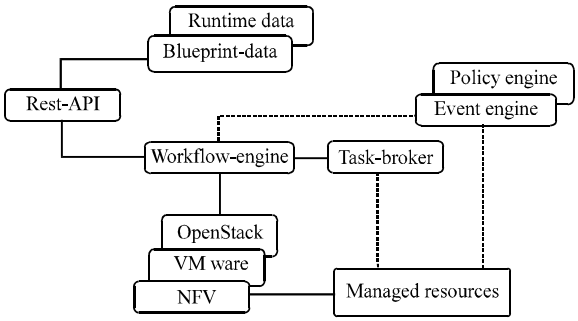

Fig. 3: Cloudify orchestration architecture

Terraform.io: Terraform is an open source orchestration tool made by Hashicorp that can manage low-level components such as compute instances, storage and networking as well as high-level components such as DNS entries, SaaS features, etc. It uses single service for orchestration execution.

Fundamental technologies: The deployment infrastructure consists of several key components. The orchestration tool prepare hardware resources and configuration management perform the software configuration. The virtual server based resources are handled by OpenStack cloud platform and container resources are handled by Kubernetes cluster management.

The cloudify platform was chosen for the testbed orchestrator as the cloudify has the support for all platforms needed for the use-case tests. It has official plugins for OpenStack and Kubernetes resource management. We have added resources describing the application services. Configuration management and software orchestration is handled by salt stack (Salt Stack, 2016). It provides necessary software components called formulas for service configuration. We use client-master execution model for managing virtual servers and local execution model for container management.

OpenStack provides all IaaS resources needed. All application stacks need a network with SNAT routing and public entry point realized by floating IP resource. When virtual hardware resources are set up, SaltStack does the software configuration. Configuration is enforced by applying model state on virtual servers. The model is stored at salt master server and all new servers have salt minion service created and configured to connect the shared salt master. After all, nodes are created an orchestration process is run to orchestrate services across all nodes in system in correct order (Fig. 4). Kubernetes provides container based microservices.

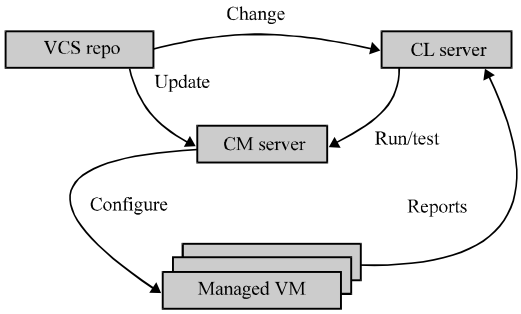

Fig. 4: Virtual machine service configuration cycle

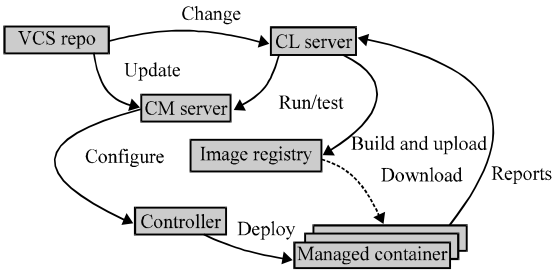

Fig. 5: Container microservice configuration cycle

Containers are clustered in pods. Container configuration is used to build base images that are stored in local image registry. There are several ways to create docker based containers out of salt formulas: substituting Docker file with parsed formula states (Flyingcloud, 2017). Flying cloud runs Salt in masterless mode, applying Salt states for each Docker layer. The other way is to setup salt minion on the node and run the state enforcement. We have chosen the second way as several services require certain resources be created at container start time, for example; MySQL database definitions, Rabbitmq virtual hosts, etc. (Fig. 5).

RESULTS AND DISCUSSION

Use-case scenarios: For the realization of this proof of concept project a specific multi-tier application service topology had been selected as the use case on which we would evaluate the related technologies.

Tested application is standard 3-tier web application in Python programming language, comprising a Load Balancer component implemented through Nginx open source software package and scaling cluster of applications implemented with Leonardo CMS running on Gunicorn web server. Data layer is realized by a Postgre SQL database server implementation. The graphical representation of the application topology is presented in Fig. 3 “tier Application Topology”.

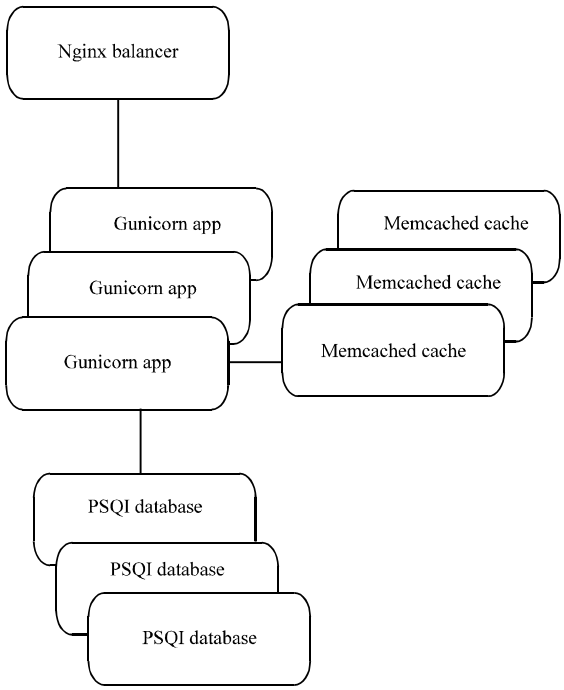

Fig. 6: Service topology of multi-cloud architecture

Deployment cases: For the description of the use case topology following the TOSCA specification, a series of node types, relationship types and artifact types have been defined in order to capture all the entities of the application and their interrelation.

Multi-cloud architecture: While those types are actually re-usable definitions, they could populate a pool of resources that could be used by application developers for other service topology descriptions. The node types definitions of the use case application are presented in Fig. 6-8.

Service architecture defines the relationships between the nodes and all instances of node type. The topology template of the multi-cloud use case application showing the software components (Nginx, Leonardo CMS, Memcached and Postgre SQL server) to be hosted on an operating system that is hosted on an OpenStack server and Amazon EC2 instances.

Cloud to containers architecture: The node types definitions of the “Cloud to Containers” use caseconsist of selected software components (Nginx, Gunicorn, Memcached are in form of microservice hosted on Kubernetes cluster. Postgre SQL server remains hosted on an operating system that is hosted on an OpenStack server.

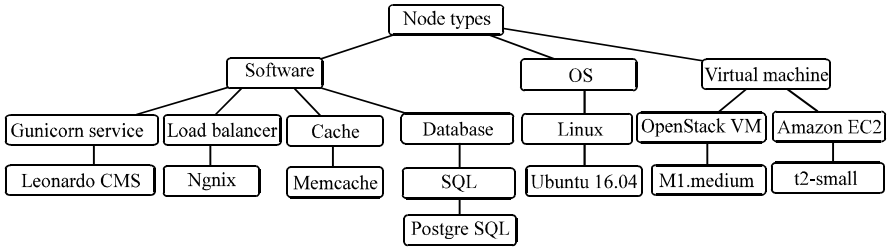

Fig. 7: Node type definitions for cloud to containers architecture

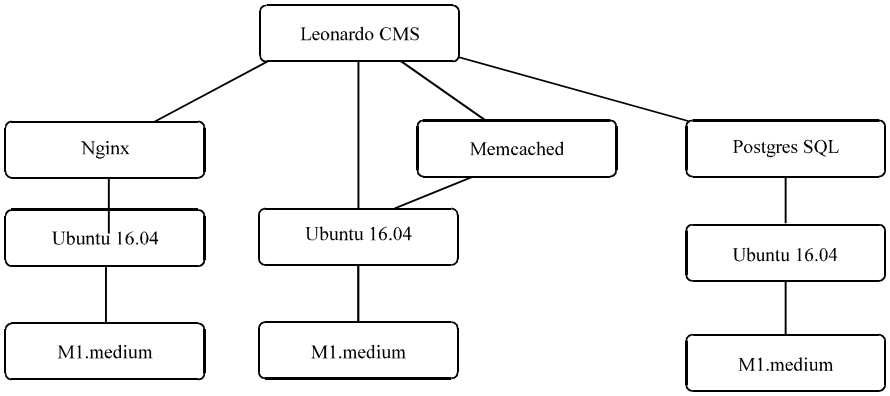

Fig. 8: Service topology of cloud to containers architecture

CONCLUSION

The capability of reusable application modeling for cloud enables the reusability and portability and is an important precondition to truly realize benefits of virtualized services in cloud environment. Different requirements lead to different approaches. Kubernetes may be the right choice for container micro-services based applications. OpenStack heat or Amazon CloudFormation are viable options for orchestration of specific cloud infrastructure.

But if target application uses combination of servers and containers or multiple cloud platforms then orchestration tool that isn't bound to a specific technology is needed. This requires well-defined standard that is generally adopted by the industry. TOSCA based infrastructure models provide such a standard language for description of topology, processes and policies of governed systems. TOSCA specification is an important initiative that bring a holistic approach to cloud application portability and orchestration.

This PoC shows how single model can provide metadata for both virtual server services and container microservices. Execution layer is realized by cloudify orchestrator combined with SaltStack configuration management platform. The container image building process reuses existing SaltStack formulas and introduces reasonable size. The cloudify provisioner plugin design supports VM ware, Hyper-V, CloudStack, Azure.

ACKNOWLEDGEMENT

This research and the contribution were also supported by project "Smart Solutions for Ubiquitous Computing Environments" FIM, University of Hradec Kralove, Czech Republic (under ID: UHK-FIM-SP-2016 2102).

REFERENCES

Anonymous, 2015. Building docker containers using salt. Loglib.org, Rome, Italy. https://www.logilab.org/blogentry/290489.

Anonymous, 2016. INDIGO-datacloud: Foundations and architectural description of a platform as a Service oriented to scientific computing. INDIGO, Gurgaon, India. https://arxiv.org/pdf/1603.09536v3.pdf.

Antonescu, A.F., P. Robinson and T. Braun, 2012. Dynamic topology orchestration for distributed cloud-based applications. Proceedings of the 2012 2nd Symposium on Network Cloud Computing and Applications (NCCA), December 3-4, 2012, IEEE, Switzerland, Europe, ISBN:978-1-4673-5581-0, pp: 116-123.

Binz, T., G. Breiter, F. Leyman and T. Spatzier, 2012. Portable cloud services using tosca. IEEE. Internet Comput., 16: 80-85.

Flyingcloud, 2017. Build docker images using saltstack. Flyingcloud, UK. https://github. com/cookbrite/flying cloud.

IBM., 2013. IBM cloud orchestrator cloud management for your it services that allows you to manage public, private and hybrid clouds with an easy-to-use interface. IBM Computer manufacturing company, Armonk, North Castle. http://www-03.ibm.com/soft ware/products/us/en/smartcloud-orchestrator/.

Juve, G. and E. Deelman, 2011. Automating application deployment in infrastructure clouds. Proceedings of the 2011 IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom), 29 November-1 December, 2011, IEEE, California, USA., ISBN:978-1-4673-0090-2, pp: 658-665.

Kubernetes, 2017. What is Kubernetes. Kubernetes, Phoenix, Arizona. http://kubernetes.io/v1.0/docs/ whatisk8s.html.

Liu, C., Y. Mao, V.D.J. Merwe and M. Fernandez, 2011. Cloud resource orchestration: A data-centric approach. Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR), January 9-12, 2011, Asilomar, California, USA., pp: 1-8.

Salt Stack, 2016. SaltStack documentation. Salt Stack, Phoenix, Arizona. https://docs.saltstack.com/en/ latest/.

TC, O.T., 2012. Topology and orchestration specification for cloud applications version 1.0. Oasis, Manchester?, England.

Wittig, A., 2015. Amazon Web Services in Action. Manning Publications Company, Greenwich, Connecticut, ISBN: 9781617292880, Pages: 397.