OpenSource Control in Cloud Computing

Hongyan Sun

Software College, University of Science and Technology Liaoning, China

gykwcs@163.com

Abstract. This paper covers basic OpenSource control and configuration management tools that can be used to alleviate common operations tasks and processes in Cloud systems. It shows a quick survey of major Cloud Computing solutions and introduces simple abstraction layer for the management of physical and virtual resources. The last chapter covers some of the common control scenarios from both Cloud Computing provider and consumer perspectives and their possible open source implementations.

Keywords: Cloud Computing; Infrastructure as a Service (IaaS); configuration management; contious integration; control.

1. Introduction

The cloud computing can be denned as "computing in an independent or remote location with shared resources available on demand"[1]. The cloud computing is a new delivery and consumption model for IT services. This involves provision of dynamically scalable and often virtualized resources typically over the Internet. This paper narrows use of the term Cloud Computing to supply Infrastructure as a service. In Cloud Computing, everything becomes a service. Primary motivation for organizations to move to Cloud computing may reduce the cost by dynamic resource allocation [2]. The Underlying physical infrastructure of Cloud Computing solutions is maintained by the Cloud providers who configure the physical servers along with variety network and storage devices. The individual devices and physical servers can be usually managed [3, 4] by consoles or SSH which needs to be properly configured.

The Cloud computing consumers do not have to worry about hardware layer and work solely with virtualized resources. Many consumers today need [1] more than one Cloud Computing solution when they solve their final IT infrastructure. This typically means private or public clouds matched with development and support systems. These multi-cloud deployments are now commonly set up [2]. Individual cloud computing resources can be managed with their native interfaces and consoles. These approaches are hardly scalable [3]. There are too many parts to consider and the operations without the advantage of control will not be effective in the long term. Another approach is to remove yourself from these different interfaces or devices by means of an abstraction layer. As a result, both providers and consumers can easily provide and manage [5] heterogeneous systems across many types of cloud computing solutions and hardware components.

This paper shows how modern OpenSource tools can provide complete application stack for the purpose of automatic management of Cloud Computing used by cloud consumers as well as providers. At some point, h both the consumers and providers use the same deployment and operation patterns and thus face the same problem.

The continuous integration is used to test the development of OpenStack platform core services [6]. The Apache Foundation [7] uses CI tools to test the majority projects that it supports.

2. Configuration Management of Models

Configuration management usually details the information that describes the enterprise's hardware and software [5]. This information typically includes the versions and updates that have been applied to the installed software packages, the locations and addresses of the hardware network devices and

the service interfaces, etc. When a system or a component of a system needs to upgrade a hardware or software, a technician can access the configuration management to see what has been currently installed. The configuration management software is usually available in both the free OpenSource and the paid commercial forms. 2.1 Configuration management tools.

Today there are lot of configuration management tools; but our focus is narrowed to the most popular configuration management platforms. The user describes the system services [8] and their state by using the platform's declarative language. The configuration server discovers the system information and compiles the resources into a system-specific catalogue. This catalogue contain all resources and dependencies, which are applied to the target system.

CFEngine is the first OpenSource configuration management system written by Mark Burgess in 1993 [9]. It is primarily to provide automated configuration and maintenance of large-scale computer systems. It involves the management of servers and embeds the networked devices or mobile smartphones. Shortly after its birth, CFEngine inspired a field of research into automated configuration management.

Puppet is a tool designed to manage the declarative configuration of Unix/Linux and Windows systems [10]. It has great community with the configuration management recipes with lot of software components and unique features.

Ansible combines the multi-node deployment [11], the ad-hoc task execution and the configuration management in one tool. Ansible manages nodes over SSH and does not require any additional remote software to be installed on them. Modules works over JSON6 and standard output, and can be written in any language. YAML7 is used to express reusable descriptions of systems.

SaltStack takes a new approach to the infrastructure management by developing a software that is easy enough to get running in seconds[12], scalable enough to manage tens of thousands of servers, and fast enough to control and communicate with them in milliseconds. SaltStack delivers a dynamic infrastructure communication bus used for orchestration, remote execution and configuration management, etc.

All changes to the managment infrastructure are propagated in the configuration management cycles. This is common to all configuration management systems. If the configuration is managed in a decentralized way, the managed node will become its own configuration server!

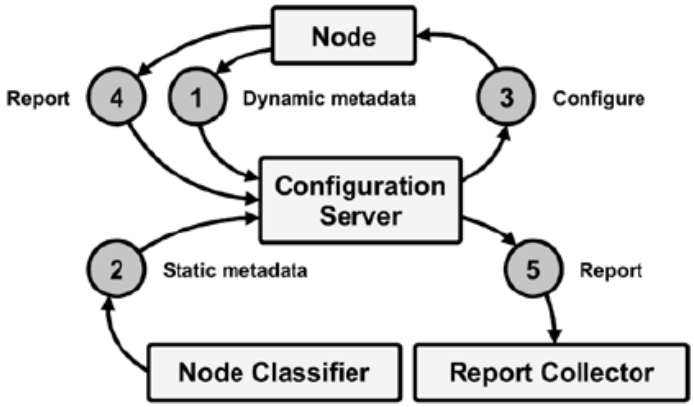

Fig. 1 Conguration Management Cycle

The following enumeration explains every step in the configuration management cycle. 1. The node sends normalized data about itself. 2. The external metadata server sends service classification for the given node. 3. The facts and metadata are used to compile final service catalogue that species how the node will be configured. 4. The node sends report back to configuration to indicate that the config run is complete.

3. Continous Integration

Continuous Integration (CI) adds an important value to software configuration management. It ensures that each change is integrated. It provides important information about the health of a project. CI integrates all changes as early as possible. The basic idea [5] is to integrate new code as early as possible and test the modifications in the context of the entire project. In order to be able to use CI for configuration management, a few requirements must be fulfilled.

The changes to the infrastructure through the configuration management tools are handled in continuous integration cycles. A CI service fetches the source code of configuration management definitions after each change and runs tests to check the functionality. If a defect is found in the code, it can be identified and corrected as soon as possible. If the code passes the initial tests, it is built in the test environment and the result is tested.

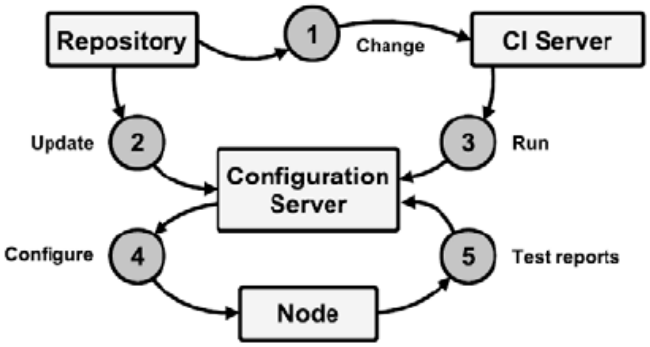

Fig. 2 Continous Integration Cycle

1. CI server detects the code changes in version control system and starts to auto-test the changed code by basic test suite. 2. After the CI server confirms to code changes (recipes, metadata), they get propagated to the configuration server. 3. After the configuration server is updated, the CI server calls configuration server to commit the new settings. 4. The configuration server uses new configuration to setup the target systems. 5. The target systems send report about their configuration management runs to the configuration server. CI server processes data from the configuration server and sets the changes to the code as good or broken.

The continuous integration is important for the rapid software development. It is very helpful for testing changes as to complex software systems because the change in one part of the system may consequently affect other components. By turning server's provision and configuration into an automated process driven by the service models, the same principles as the software development are applicable. The traditional continuous integration tools can be used to facilitate the test integrity, security and performance whenever the change to the system models is detected.

4. Automated Cloud Computing Scenarions

There are many use cases concerning where the control can be used in Cloud Computing. The cloud providers have to set up physical infrastructures to monitor its operation and manage hardware failures as quickly as possible. The cloud consumers want to automate the development and operations of their application infrastructures; besides, everyone needs proper backup and recovery policies. The following examples show how the open source solutions solve most of the common control scenarios. The scenarios are tested in the CEPSOS laboratory [13] at Hradec Kralove.

Auto-provisioning - With well defined models of architectures, the cloud controller or physical hardware suppliers can set up the desired servers and configuration management service to complete their configuration. Mirantis Fuel [14] is a tool based on the OpenStack Fuel project [15, 16, 17] that automates installation of entire OpenStack infrastructure. Fuel server provides web interface to set up proper metadata while the Cobbler service is to provide physical servers and Puppet master with detailed configuration of every server. Basic setup starts with three controller servers in high

availability setup with two or more compute servers. As Mirantis Fuel is an open source solution backed up by commercial support, it is put together by Mirantis with the majority coded based on the community.

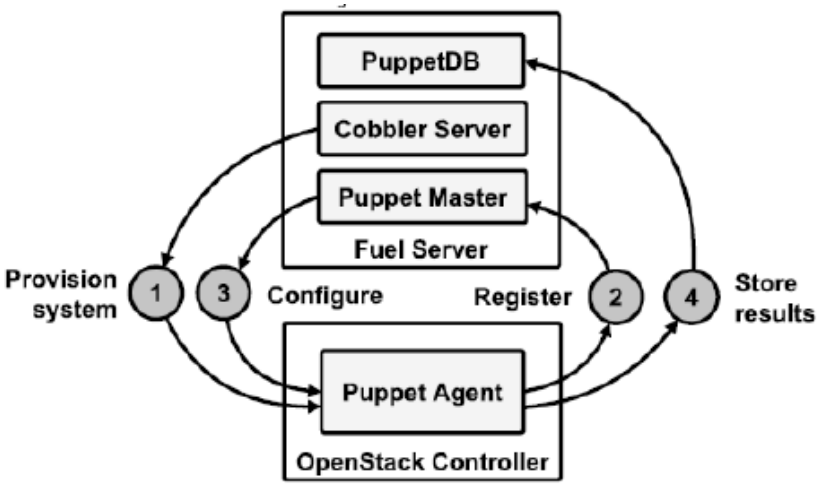

Fig. 3 Mirantis Fuel Architecture

1. Cobbler server installs operating system with Puppet Agent on all servers. 2. Servers are auto-registered in Fuel's integrated Puppet Master. 3. State definition from Fuel is used to configure the target systems. 4. Server's report status of their configuration runs.

Another tool that automates the installation of OpenStack infrastructure is DevStack project [18] by RedHat, which is a set of scripts for quick deployment of an OpenStack cloud infrastructure. These tools provide simple way to provide metadata to hardware suppliers as well as configuration management service. This approach can provide any number or server at once. More examples of automated provision can be found in chapter Multi-cloud Architectures.

Monitoring and metering — the service monitoring is an important part of the system operations. The monitoring is essential for measuring the quality of services and should be set up along with services that are being measured. The configuration management can ease the deployment of local and remote checks and update them on y whenever the infrastructure changes. Sensu [19] is often described as the monitoring router. Sensu takes the results of each check script from across many systems. If certain conditions are met, it passes their information to one or more handlers. Checks are used, for example, to determine whether a service like Apache is up or down. Checks can be used to collect other data, such as PostgreSQL or Redis query statistics or any application metrics. Handlers take actions by using the information from the check results, send an email, send message to a chat room or add a data point to a graph.

Multi-cloud architectures - modern enterprise systems require distribution of application infrastructures across availability zones, regions or even cloud services. Distribution of load across infrastructure components is hosted in one or more cloud services. OpenSource tools combine standard cloud libraries with configuration management services to fully automate the creation process within Cloud services. With the standard libraries, there will be no vendor lockin. Foreman [20] is a lifecycle management tool for both physical and virtual servers. Though it's a smart proxy architecture, it can automate repetitive tasks, quickly deploy applications and manage changes on premise VMs, baremetal or virtual servers in the cloud. Foreman well integrates Puppet or Chef and provides external classification service for Puppet. It provides nice web interface as well as API for further integrations. Aside cloud integration Foreman can provide hardware resources although it's a proxy architecture [21].

Another example of tool supporting multi-cloud architectures is Salt cloud [22], which can be configured to provide multiple virtual servers at different Cloud Computing providers at once [22, 23, 24]. 1. The Salt cloud creates new keys for all new virtual servers. 2. The Salt cloud calls the Cloud API through standard library to create new virtual servers. 3. The Cloud controller launches virtual

machines and sets up connection to the Salt master. 4. Salt Master enforces configuration on all virtual servers. 5. Salt Minions report status of their configuration runs to the Salt Master.

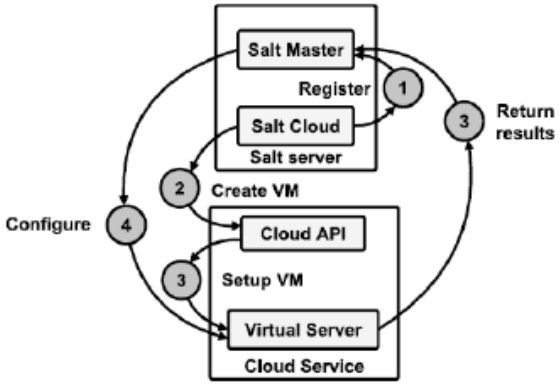

Fig. 4 Salt Cloud Provision Architecture

5. Conclusion

The operating enterprise systems without control are not sustainable in a long term. We can see rapid development of new Cloud Computing resources coping up with increasing demand to support these complex infrastructures. The open source world introduces some enterprise level solutions to many common control scenarios. The ability of automatic operations and transition processes by continuous integration has become crucial in the IT environment that is never still. With the open source control, we utilize more efficiently the laboratory hardware resources by turning the system architecture into fully edged IaaS solution that can now support educational as well as research projects. The own cloud computing platform allows us to automate the provision of new virtual servers and thus adopt the last missing step to continuously integrate the process. With this infrastructure we can continuously test the open source control scenarios involving the installation of OpenStack platform on physical servers and the deployment of virtual servers for education and various distributed systems. With the continuous integration, we can keep enhancing the quality of all undergoing projects in CEPSOS laboratory. This work and the contribution have been supported by project “SP/2014/05 - Smart Solutions for Ubiquitous Computing Environments” from University of Hradec Kralove.

References

[1] George Reese Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. O'Reilly Media, 978-0596156367, Sebastopol, USA, 2009
[2] Bill Wilder Cloud Architecture Patterns O'Reilly Media, 978-1449319779, Sebastopol, USA, 2012
[3] George Reese, J. A. Scott and D. Nisse, editors. Guide to the Software Engineering Body of Knowledge, 2004 version IEEE Computer Society Press, 0-769523307, Los Alamitos, USA, 2004
[4] NIST. The NIST De_nition of Cloud Computing http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.
[5] Paul M. Duvall Continuous Integration: Improving Software Quality and Reducing Risk. Addison- Wesley Professional, 978-0321336385, TODO, USA, 2007
[6] OpenStack. OpenStack Project Infrastructure http://ci.openstack.org/, cited at 9.5.2014.
[7] Apache Foundation: Build Services http://ci.apache.org/, cited at 15.5.2014.
[8] Ivan Ivanov, Marten van Sinderen and Boris Shishkov, editors. Cloud Computing and Services Science Springer Science, 978-1461423256, New York, USA, 2012

[9] CFEngine. FCEngine 3.5 Dcoumentation https://cfengine.com/docs/3.5/index.html, cited at 2.3.2014.
[10] Dunn, C.: Craig Dunn Designing Puppet: Roles and Profiles. http: //www. craigdunn. org/2012 /05/239/, cited at 3.3.2014.
[11] Ansible Documentation. Ansible Documentation http://docs.ansible.com/, cited at 2.3.2014.
[12] SaltStack. Salt Built-in Pillars http://docs.saltstack.com/ref/pillar/all/index.html#all- salt-pillars, cited at 3.3.2014.
[13] CEPSOS. Projekty a veda http://www.cepsos.cz/cinnost.html, cited at 11.5.2014.
[14] Mirantis. Mirantis OpenStack v4.1 Documentation: Reference Architectures http://docs. mirantis. com/fuel/fuel-4.1/reference-architecture.html, cited at 2.5.2014.
[15] OpenStack. Fuel Wiki https://wiki.openstack.org/wiki/Fuel, cited at 15.5.2014.
[16] OpenStack. Jenkins Job Builder http://ci.openstack.org/jenkins-job-builder/, cited at 15.5.2014.
[17] OpenSource. Duplicity http://duplicity.nongnu.org/index.html, cited at 1.4.2014.
[18] Fedora Project. OpenStack devstack http://fedoraproject.org/wiki/OpenStack devstack, cited at 15.4.2014.
[19] Sensu. Sensu Documentation http://sensuapp.org/docs/0. 12/overview, cited at 21.4.2014.
[20] Lang, J. P.:The Foreman. The Manual: Compute Resources http:// theforeman. org/manuals /1.4/index.html#5.2ComputeResources, cited at 10.4.2014.
[21] Lang, J. P.: The Foreman. Smart Proxy http://projects.theforeman.org/projects/smart-proxy/wiki, cited at 10.4.2014.
[22] SaltStack. Salt Cloud http://docs.saltstack.com/topics/cloud/index.html, cited at 5.4.2014.
[23] SaltStack. Salt Virt http://salt.readthedocs.org/en/latest/topics/virt/index.html, cited at 5.4.2014.
[24] Gitlab.com. GitLab Community Edition https://www.gitlab.com/gitlab-ce/, cited at 2.3.2014.