Survey of software configuration management tools of nodes in heterogeneous distributed computing environment

R. O. Kostromin[1]

[1]Matrosov Institute for System Dynamics and Control Theory of SB RAS, Lermontov St. 134, Irkutsk, Russia, 664033

kostromin@icc.ru

Abstract. The paper discusses the problem of preparing a computing environment for large-scale scientific experiments in the process of continuous integration of applied and system software. A comparative analysis of software configuration management tools (such as Chef, Ansible, Puppet, and SaltStack) of computational nodes in a heterogeneous environment is being performed. These tools are intended for automating the configuration of different nodes. Such automation reduces the setup time of nodes and increases the reliability of computations by minimizing the number of software and hardware failures, associated with the human factor in the manual configuration process. For the development of scientific applications, the Orlando Tools framework is used. Based on the results of the comparative analysis and requirements of this framework, the Ansible framework was selected for further integration into the chain of continuous integration of applied and system software. Practical experiments have shown the advantages of using Ansible in comparison with other systems of a similar purpose.

1. Introduction

Currently, software development is a complex and multi-step process [1]. Usually, these processes include the steps of testing, debugging, and delivering software to end-users. The development of an automated chain of effective delivery for both the applications in a whole and their components and modules is still a vital problem. In practice, a useful approach to solving this problem is the Continuous Integration (CI) of software [2].

The CI tools create an automated chain of steps from receiving updated source code from the repository to deploying a finished scientific application. Within of CI, users receive a software product as a service. They regularly develop new or modify existing versions of applications, debug their software, and test it.

Developers are forced to accompany both the application and computing infrastructure with software development environment necessary for debugging and testing during the CI process. This leads to a new concept of representing the infrastructure in the form of code (Infrastructure as Code [3]). As a result, a manual configuration of such infrastructure takes significant time in preparing an application for release.

Manual infrastructure management obstructs the development and maintenance of software. Using own automation scripts complicates its development and maintenance. In particular, this refers to the Parallel SSH tools for a parallel script execution [4]. The above-listed problems arise in both the commercial and scientific sphere of software development.

Applications for assisting in solving research and applied problems on supercomputers are truly important in the scientific field [5]. Usually, such applications have a modular structure and represent a distributed applied software package [6]. Such packages are designed for multiple runs with different sets of input data in environments with the various configurations and sets of resources.

Software packages for the public access Irkutsk Supercomputer Center (ISC) [7] are developed using the Orlando Tools framework [8]. The specifics of the problems solved with these packages determine the frequent modifications of the application software. In this regard, new CI tools are integrated into its architecture [9].

However, infrastructure configuration automation for debugging and testing packages is still a challenge [10]. In particular, there remains a need to prepare and store various Virtual Machine (VM) images for different Operating Systems (OSes) and software versions. In this case, the disk space is quickly exhausted. In addition, this complicates the infrastructure maintenance.

Thus, the following requirements for configuration management tools are formulated:

- Support managing a wide range of OSes,
- Full control at every stage of node configurations for the computing infrastructure,
- Easy deployment with the minimal preparation of user nodes,
- Free distribution,
- Server-side initialization of slave node configuration changes (within of the CI chain),
- Node management using the SSH protocol (in particular, for managing low-performance peripherals used in fog computing).

The paper discusses the well-known Software Configuration Management (SCM) tools of nodes in a computing environment [10]. The important properties of such systems and configuration management scheme are considered.

The rest of the paper is organized as follows. Section 2 reviews the properties of four SCM tools. Particular attention is paid for two systems. Their characteristics are compared in section 3. A comparative analysis of different methods for managing configurations of nodes in an experimental environment is presented in Section 4. The last section concludes the paper.

2. Related work

Currently, a wide range of the SCM tools is known [10]. These tools automate the configuration processes of various nodes and deploy a computing infrastructure with the subsequent delivery of applications to its nodes.

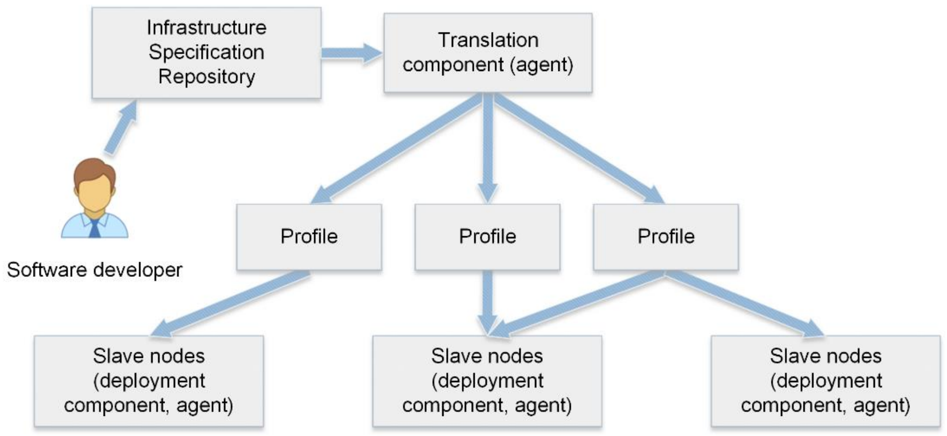
Infrastructure Specification Repository
Translation component (agent)
Software developer
Profile
Profile
Profile
Slave nodes (deployment component, agent)
Slave nodes (deployment component, agent)
Slave nodes (deployment component, agent)

Figure 1. General SCM scheme.

In addition, such systems can improve the manageability and reliability of both the configuration processes and maintenance of infrastructure nodes. However, preparing and launching VMs are not supported in SCM. This process is implemented using hypervisors or special add-ons [11].

It is assumed that the SCM tools executed on OS with a clean installation. Tools configure and prepare the nodes for work. Thus, the software developer defines the configuration requirements for the node. During the operation of nodes, their configurations are monitored and configured by the SCM tools in accordance with the requirements. The general SCM scheme is presented in Figure 1.

By executing special program code (scripts), the SCM tools allow us automatically lead the computing infrastructure to the target state [12]. Such code is conveniently developed, modified, and maintained using version control systems such as Git [13] and Mercurial [14]. The software developer creates and stores the infrastructure specification in the repository. Then the translation component (agent) independently or at the signal of the administrator passes the specification (profile) to the slave nodes. In nodes, these profiles are represented as a set of executable commands by deployment components (agents). On slave nodes, these profiles are represented as a set of executable commands by deployment components (agents).

The most popular SCM tools are Puppet [15], Chef [16], Ansible [17], and SaltStack [18]. Their general characteristics are presented in Table 1. Among them the system development language, Configuration Description Language (CDL), system architecture, software distribution license, and supported platform. Within the execution of distributed application packages, an important requirement is a support for managing Windows and Linux OSes families. The paper does not discuss systems whose development is suspended or discontinued, as well as proprietary systems.

Table 1. General characteristics of the SCM tools.

System	**System development language**	**License**	**Architecture**	**Configuration Description Language**	**Platform support**	
					Linux	**Windows**
Ansible	Python	GPLv3+	Agentless	YAML-like, JSON	+	+
Chef	Ruby, Erlang	Apache 2.0	Client-server	Ruby	+	+
Puppet	C++ & Clojure	Apache 2.0	Client-server	Ruby-like (proprietary)	+	+
SaltStack	Python	Apache 2.0	Agentless / Client-server	YAML-like	+	+

In a client-server architecture, an agent is a host-side software that manages the configuration of its node within SCM. The agentless architecture allows us to control nodes without installing additional software via the SSH protocol. The presence of a special (proprietary) agent in the SCM tools limits the types of OSes whose configurations can be controlled. Among the systems reviewed, only Ansible is a system with a completely agentless architecture. At the same time, SaltStack also allows us to use the agentless mode. However, in this mode, it works much slower than its proprietary protocol. In general, in each system, there are agents to support the most popular OSes.

All of the considered systems are actively developing. The official websites of the systems present ull manuals. They describe all processes from the installation of systems to their usage in software development. Each system is surrounded by an international community of users and enthusiasts. This positively affects the development of the systems.

All four systems provide a web-interface for configuration management. It can be used to create reports and visualize the infrastructure configuration. In addition, all of the systems have the ability for connecting to an external monitoring system. SaltStack uses its own implementation for that.

The features of the considered SCM tools are reflected in the system characteristics, description specifics, delivery methods, and installation of system configurations. These characteristics are shown in Table 2.

Table 2. System characteristics of the SCM tools.

System	**Database**	**Configuration delivery**	**Supported server OSes**	**Step-by-step installation**	**Delivery method**
Ansible	–	SSH	IBM AIX, BSD, Linux, MacOSX, Solaris	+	Push
Chef	PostgreSQL	RabbitMQ	Linux only	+	Pull (Push only in the corporate version)
Puppet	PuppetDB	Mcollective	Linux only	–	Push, Pull
SaltStack	–	ZeroMQ	Linux, BSD	–	Pull

Database. On deploying Puppet and Chef, the PuppetDB and PostgreSQL database systems are used, respectively. These databases provide centralized configuration storage. The performance and scalability of each system directly depend on the database used. In addition, the database server requires additional maintenance.

Enterprise version of Ansible uses PostgreSQL. In addition, it is allowing to install MongoDB (MySQL) to build fault-tolerant architectures and for storing logs.

Configuration Delivery. Each of the considered systems uses its own method of the configuration delivery (transportation) from the repository to the node. At the same time, only Ansible uses SSH (Powershell on Windows) to deliver configurations. Theoretically, Ansible allowing control of any device that supports SSH.

Supported server OS. Ansible provides support for a wide range of OSes on the master node. Theoretically, any OS that supports Python 2.6 is also supported. SaltStack runs on both Linux and BSD. Puppet and Chef master nodes work only on Linux-kernel OSes. Among them are RHEL, SLES, and the latest versions of Ubuntu.

Step-by-step installation. Only Ansible and Chef use a step-by-step approach to configuring nodes. Within the SCM, a step-by-step execution is understood as a sequential execution of all the actions described in the configuration file. This approach is called an imperative configuration. Puppet and SaltStack use a declarative configuration. This means that the configuration file describes only the final state of the node. In this case, the system itself chooses the best path to achieve this state. A declarative configuration is preferred for nodes of the same type. At the same time, the imperative configuration provides full control over the process of changing the state of the node.

Method for delivering configuration changes from a master node to slave nodes. The following two delivery methods are considered: Push and Pull. In the case of Push, the master node itself monitors changes and, if necessary, initiates an update from clients. For the Pull method, client agents periodically request the server for updates.

The monitoring of changes in configurations is an important characteristic in creating an experimental and production computing infrastructure. Push-approach provides quick delivery of configurations to nodes. Only Puppet and Ansible support Push. In this case, Chef is not considered, since the use of Push is available only in the commercial version. In this regard, only the Puppet and Ansible systems are discussed in more detail below. Both of these systems have integration tools with Docker, Kubernetes, and Jenkins, which are actively used in the framework of CI.

3. Puppet vs Ansible

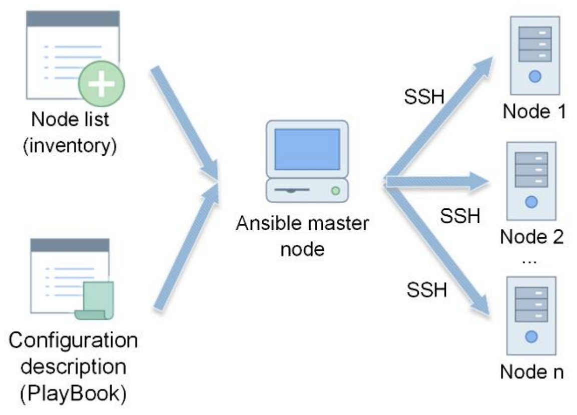

Figure 2. Node management scheme in Ansible.

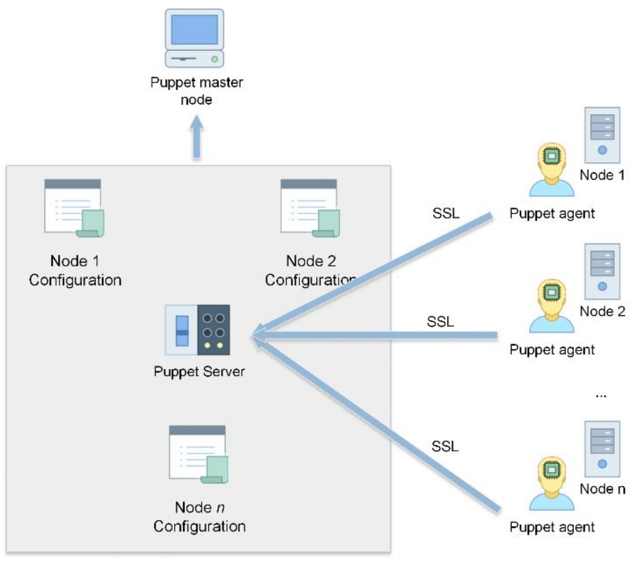

Figure 3. Node management scheme in Puppet.

Node Management in Puppet and Ansible. The first step in configuring Ansible is to select the master node. If there is an Ansible package in the repository of the OS of the node, this node may be the master-node. At this stage, we need to install Ansible and specify a list of IP-addresses of slave nodes (inventory). In this case, there is no need to configure client software. The only condition is providing access via SSH from the master node to the slave nodes. The node management scheme in Ansible is shown in Figure 2. The node configuration description in Ansible is presented as a PlayBook.

The node management scheme in Puppet is shown in Figure 3. Preparing Puppet for functioning is more difficult. First, it is necessary to synchronize time and time zone on all nodes. Next, it is required to install the server and client software. In addition, the following operations must be performed:

- Editing a configuration file for each slave node on the Puppet server,
- Opening port 8140,
- Starting the PuppetServer service,
- Specifying the IP-address of the master node and generating an SSL certificate on slave nodes.

The need to generate an SSL certificate is determined by the fact that the protocol uses HTTPS for communication.

Configuration description in Ansible. Listing 1 demonstrates the YAML code snippet of an Ansible PlayBook from the official Ansible examples repository [19]. In this example, the Tomcat web-server is installed on Centos on slave nodes. The first step is to install Java JDK 1.7. After that, the creation and configuration of the tomcat user are performed. In particular, super-user rights are added to him. Next, downloading the Tomcat source code from the repository, unpacking, configuring, and installing are performed. The YAML code is well readable. This simplifies the development with Ansible.

```
---
       - name: Install Java 1.7
         yum: name=java-1.7.0-openjdk state=present
       - name: add group "tomcat"
         group: name=tomcat
       - name: add user "tomcat"
         user: name=tomcat group=tomcat home=/usr/share/tomcat createhome=no
         become: True
         become_method: sudo
       - name: Download Tomcat
         get_url: url=http://archive.apache.org/dist/tomcat/tomcat-
       7/v7.0.61/bin/apache-tomcat-7.0.61.tar.gz dest=/opt/apache-tomcat-
       7.0.61.tar.gz
       - name: Extract archive
         command: chdir=/usr/share /bin/tar xvf /opt/apache-tomcat-
       7.0.61.tar.gz -C /opt/ creates=/opt/apache-tomcat-7.0.61
       - name: Symlink install directory
         file: src=/opt/apache-tomcat-7.0.61 path=/usr/share/tomcat state=link
       - name: Change ownership of Tomcat installation
         file: path=/usr/share/tomcat/ owner=tomcat group=tomcat
       state=directory recurse=yes
       - name: Configure Tomcat server
         template: src=server.xml dest=/usr/share/tomcat/conf/
         notify: restart tomcat
       - name: Configure Tomcat users
         template: src=tomcat-users.xml dest=/usr/share/tomcat/conf/
         notify: restart tomcat
       - name: Install Tomcat init script
         copy: src=tomcat-initscript.sh dest=/etc/init.d/tomcat mode=0755
       - name: Start Tomcat
         service: name=tomcat state=started enabled=yes
       - name: deploy iptables rules
         template: src=iptables-save dest=/etc/sysconfig/iptables
         when: "ansible_os_family == 'RedHat' and
       ansible_distribution_major_version == '6'"
         notify: restart iptables
       - name: insert firewalld rule for tomcat http port
         firewalld: port={{ http_port }}/tcp permanent=true state=enabled
       immediate=yes
         when: "ansible_os_family == 'RedHat' and
       ansible_distribution_major_version == '7'"
       - name: insert firewalld rule for tomcat https port
         firewalld: port={{ https_port }}/tcp permanent=true state=enabled
       immediate=yes
         when: "ansible_os_family == 'RedHat' and
       ansible_distribution_major_version == '7'"
       - name: wait for tomcat to start
         wait_for: port={{http_port}}
```

Listing 1: YAML-code snippet of Ansible PlayBook from the official Ansible examples repository.

Configuration description in Puppet. The configuration description (Puppet manifest) is created in a subject-oriented Ruby-like language. Listing 2 demonstrates a code snippet of the configuration description in Puppet from the repository puppetlabs/mysql, which covers the installation and configuration of MySQL [20].

Consider the advantages and disadvantages of Puppet and Ansible.

```
class {'::mysql::server':
  package_name     => 'mariadb-server',
  package_ensure   => '1:10.3.21+maria~xenial',
  service_name     => 'mysqld',
  root_password    => 'AVeryStrongPasswordUShouldEncrypt!',
  override_options => {
    mysqld => {
      'log-error' => '/var/log/mysql/mariadb.log',
      'pid-file'  => '/var/run/mysqld/mysqld.pid',
    },
    mysqld_safe => {
      'log-error' => '/var/log/mysql/mariadb.log',
    },
  }
}

# Dependency management. Only use that part if you are installing the repository
# as shown in the Preliminary step of this example.
Apt::Source['mariadb'] ~>
Class['apt::update'] ->
Class['::mysql::server']
Class {'::mysql::client':
  package_name    => 'mariadb-client',
  package_ensure  => '1:10.3.21+maria~xenial',
  bindings_enable => true,
}
# Dependency management. Only use that part if you are installing the repository as shown in the Preliminary step of this example.
Apt::Source['mariadb'] ~>
Class['apt::update'] ->
```

Listing 2: Ruby-like code snippet of the configurations description in Puppet from the repository puppetlabs/mysql.

Puppet has the following advantages:
- This is a well-known and stably developing product,
- It has a user-friendly graphical interface,
- All major OSes are supported in Puppet.

At the same time, Puppet has the following disadvantages:
- The poor performance of the Puppet, which is written in Ruby, in comparison with systems written in Python,
- The necessity to learn the proprietary Puppet language to describe configurations.

Ansible has the following benefits:
- High performance system,
- Agentless installation and deployment,
- Low overheads,
- Applying Python to develop this system,
- Easy to learn.

At the same time, Ansible is characterized by the following drawbacks:
- This is a relatively new system,
- Windows support is implemented only through PowerShell.

In general, each of the systems has its own advantages and disadvantages. Based on the requirements formulated in the introduction, it was decided to integrate the Ansible system into the CI chain of the Orlando Tools framework. This is due to the following reasons:

- Support for managing the most popular OSes,
- Using an agentless approach that simplifies a system setup,
- Using an imperative configuration of nodes provides complete control over the installation of software at each stage,
- Free distribution under the GPLv3+ license,
- Using the Push method to deliver configurations,
- Node control via the SSH protocol, which allows us to manage any device that supports this protocol.

4. Comparative analysis

In a comparative analysis, the makespan of preparing 10 nodes of the experimental computing environment is estimated. 10 VMs are launched on 10 nodes of the environment using a special hypervisor add-on developed in addition to OpenStack [11]. VMs are launched using prepared images.

The experimental environment is based on the ISC resources. Centos 7 is installed on five nodes, Ubuntu 18.04 is used on the remaining nodes. Each OS provides SSH key access.

All VMs should be prepared to deploy Orlando Tools framework. It is requires Apache Web server, PHP version 7.2, Maria DB version 10, and other system software. There are several ways to prepare an experimental computing environment without using Ansible:

- Preparation of the fully customized VM images for all possibleible cases,
- Manually installation and configuration of each executed VM,
- Automation of the installation using parallel execution of commands (scripts) through Parallel SSH.

Storing the customized images of VMs for all possible cases is impractical since the requirements for versions of system and application software can often change. In addition, in this approach, free disk space runs out rapidly.

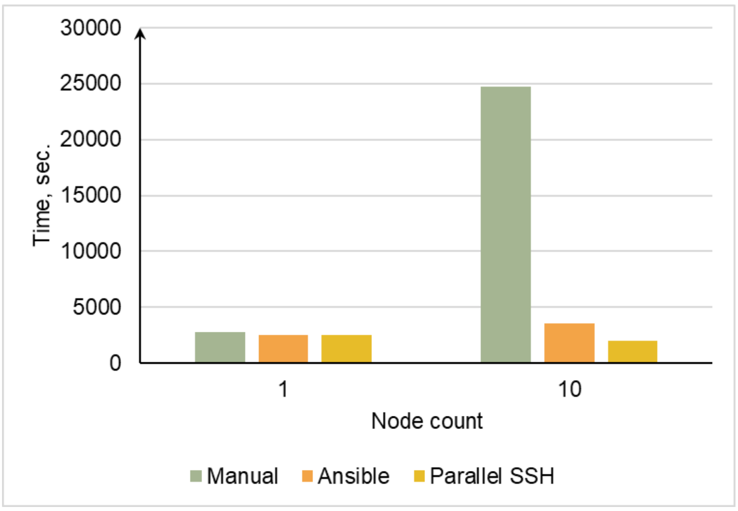

Figure 4. Comparison of the node configuration time.

In general, automating of a manual installation using Parallel SSH reduces a makespan to configure nodes. In this case, it is necessary to develop several versions of such scripts for each OS and its versions. The number of such scenarios is growing very fast since for each case it is necessary to prepare an appropriate script that will bring the node to working condition. Obviously, the use of Parallel SSH is unacceptable for numerous heterogeneous nodes in an experimental environment. In addition, the use of Parallel SSH can lead to additional errors in the development, maintenance, and debugging of various scripts.

The histogram in Figure 4 shows the significant reduction of the time for automatic configuration of nodes, compared to manual configuration. At the same time, the short time for nodes configuration using scripts is spoiled by the complexity of maintaining and deploying of own scripts. Ansible has no such problems.

5. Conclusions

The paper provides a comparative analysis of the SCM tools for computing infrastructures. Such tools are designed to automate the configuration of infrastructure nodes. As a result of automation, the time of node configurations is significantly reduced. In addition, the reliability of nodes is improved by reducing the configuration errors associated with the human factor.

The advantages of SCM are especially evident in its integration into the CI chain. At the same time, the required number of VMs is launched using virtualization tools. Further VM configuration is performed automatically by SCM.

Based on the results of the performed analysis, Ansible was selected for integration into the CI chain in Orlando Tools. The results of an experimental analysis of the preparation of an experimental computing environment have demonstrated the advantages of in using Ansible. Thus, it provides an improvement in the quality of development, testing, and distribution of modules of distributed applied software packages.

Acknowledgments

The study is supported by the Basic Research Program of SB RAS, project no. IV.38.1.1.

References

[1] Kim G, Debois P, Willis J, Humble J 2016 *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (IT Revolution Press) p 480

[2] Shahin M, Babar M A and Zhu L 2017 Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices *IEEE Access* **5** 3909–3943

[3] Morris K 2016 *Infrastructure as Code* (O'Reilly Media) p 362

[4] Parallel-SSH. Available at: https://parallel-ssh.org/ (accessed: 10.06.2020)

[5] Inggs G, Thomas D B and Luk W 2017 A Domain Specific Approach to High Performance Heterogeneous Computing *IEEE Transactions on Parallel and Distributed Systems* **28(1)** 2–15

[6] Feoktistov A G, Kostromin R O, Sidorov I A and Gorsky S A 2018 Development of Distributed Subject-Oriented Applications for Cloud Computing through the Integration of Conceptual and Modular Programming *Proceedings of the 41st Intern. Convention on information and communication technology, electronics and microelectronics (MIPRO-2018)* (Riejka: IEEE) pp 256–261

[7] Public access center Irkutsk Supercomputer Center. Available at: http://hpc.icc.ru (accessed: 10.06.2020)

[8] Tchernykh A, Feoktistov A, Gorsky S, Sidorov I, Kostromin R, Bychkov I, Basharina O, Alexandrov A and Rivera-Rodriguez R 2019 Orlando Tools: Development, Training, and Use of Scalable Applications in Heterogeneous Distributed Computing Environments *Comm. Com. Inf. Sc.* **979** 265–279

[9] Feoktistov A, Gorsky S, Sidorov I and Tchernykh A 2019 Continuous Integration in Distributed Applied Software Packages *Proc. of the 42th Int. Convention on information and communication technology, electronics and microelectronics* (Riejka: IEEE) pp 1775–1780

[10] Quigley J M, Robertson K L 2015 *Configuration Management: Theory, Practice, and Application* (Auerbach Publications) p 438

[11] Feoktistov A, Sidorov I, Sergeev V, Kostromin R and Bogdanova V 2017 Virtualization of heterogeneous HPC-clusters based on OpenStack platform *South Ural State University Bulletin: Computational Mathematics and Software Engineering series* **6(2)** 37–48

[12] Delaet T, Joosen W and Vanbrabant B 2010 A survey of system configuration tools *Proc. of the 24th Int. conf. on Large installation system administration* (USENIX Association, USA) pp 1–8

[13] GitHub. Available at: https://github.com/ (accessed: 10.06.2020)

[14] Mercurial. Available at: https://www.mercurial-scm.org/ (accessed: 10.06.2020)

[15] Puppet. Available at: https://puppet.com/ (accessed: 10.06.2020)

[16] Chef. Available at: https://www.chef.io/products/chef-infra (accessed: 10.06.2020)

[17] Ansible. Available at: https://www.ansible.com/ (accessed: 10.06.2020)

[18] SaltStack. Available at: https://www.saltstack.com/ (accessed: 10.06.2020)

[19] Ansible PlayBook examples repository. Available at: https://github.com/ansible/ansible-examples/tree/master/tomcat-standalone (accessed: 10.06.2020)

[20] Puppetlabs manifest examples repository. Available at: https://forge.puppet.com/puppetlabs/mysq (accessed: 10.06.2020)