Automated Testing of Chef Automation Scripts

Waldemar Hummer
Distributed Systems Group
Vienna University of Technology, Austria
hummer@dsg.tuwien.ac.at

Florian Rosenberg, Fábio Oliveira,
Tamar Eilam
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
{rosenberg,fabolive,eilamt}@us.ibm.com

ABSTRACT

Infrastructure as Code (IaC) is a novel approach for deployment of middleware and applications. IaC typically builds on automation scripts to put the system into a specific state. The series of steps in an automation should be *idempotent* to guarantee repeatability and convergence. These are key factors if automations are run periodically to override out-of-band changes and prevent drifts from the desired state. Rigorous testing must ensure that the system reliably converges from arbitrary initial/intermediate states to a desired state.

We tackle this issue and demonstrate our tool for automated testing of automation scripts. Our tool is tailored to Opscode's *Chef*, one of the most popular IaC frameworks to date. Various testing parameters can be configured, and the Web-based user interface allows to inspect the system state changes during execution. Detailed test reports are created at the end of a test suite, which facilitate tracking down the root cause of failures and issues of non-idempotence.

Keywords

Testing, Idempotence, Infrastructure as Code, Automation

1. INTRODUCTION

In order to repeatedly deploy middleware and applications to production environments, operations teams typically rely on automation logic (scripts). Poorly written automations incur an increased risk of compromising the stability of deployments. *Infrastructure as Code* (IaC) [8, 9] is becoming a key concept to facilitate the development of automation logic for deploying, configuring, and upgrading inter-related middleware components. IaC automations are designed to be repeatable, making the system converge to a desired state starting from arbitrary states. The notion of *idempotence* builds the foundation for repeatable, robust automations [3, 1]. State-of-the-art IaC tools, such as Chef [10] or Puppet [11], provide developers with abstractions to express automation steps as idempotent units of work.

We demonstrate our framework for comprehensive testing of automation scripts. The demo is based on and complements our paper [7] to be presented at the Middleware'13 main conference. While the approach is general, the demo is tailored to Chef. Throughout the demo, we showcase our tool based on testing scenarios with real-world Chef scripts.

In the rest of this paper, we discuss background and motivation (§2), outline the testing approach (§3), walk through the main steps of the demo (§4), and discuss related work (§5).

2. BACKGROUND AND MOTIVATION

We briefly discuss the principles behind modern IaC tools like Chef, and motivate the importance of testing for IaC.
Chef background. In Chef terminology, automation logic is written as *cookbooks*, consisting of *recipes*. Recipes describe a series of *resources* that should be in a particular state.

```
1  directory "/foo" do
2    mode 0755
3    action :create
4  end
5  package "tomcat6" do
6    action :install
7  end
8  service "tomcat6" do
9    action :enable
10 end
```

```
1  bash "build php" do
2    cwd /tmp
3    code <<-EOF
4  tar -zxf php.tar.gz
5  cd php
6  ./configure
7  make && make install
8  EOF
9    not_if "which php"
10 end
```

Figure 1: Declarative and Imperative Chef Recipes

Figure 1 shows two sample recipes. The left recipe has *declarative* resources which define a desired state (directory `/foo`, package `tomcat6`, OS service `tomcat6`). The resource types are implemented by platform-dependent providers, and Chef ensures that their implementation is idempotent. Thus, even if our sample recipe is executed multiple times, it will not fail trying to create directory `/foo` that already exists.

The right listing in Figure 1 illustrates an *imperative* `bash` resource installing PHP. This excerpt shows the common scenario of installing software from source code – unpack, compile, install (lines 4–7). To encourage idempotence even for arbitrary scripts, Chef provides statements such as `not_if` (line 9) or `only_if` to indicate conditional execution.
Threats to overall idempotence. Idempotence is critical to the correctness of Chef recipes, and we identify several challenges when it comes to ensuring that a recipe is idempotent and can make the system converge to a desired state.

First, for imperative script resources, the user has the burden of implementing the script in an idempotent way, which may not be trivial. Second, although Chef guarantees that declarative resources are idempotent, there is no guarantee that a sequence of multiple instances as a whole is idempotent [3]. Finally, if recipes depend on external components, achieving overall idempotence may become harder due to unforeseen interactions (e.g., a download server is down).

3. APPROACH SYNOPSIS

Our work proposes an approach and framework for testing Chef automations. We follow a model-based testing approach [13], according to the process outlined in Figure 2. Our test model consists of two main parts: 1) a system model of the automation under test and its environment, including the involved tasks, parameters, system states, and state changes; 2) a state transition graph (STG) which is constructed based on user-defined test coverage. The model is automatically constructed by parsing the Chef scripts. The test cases are materialized and executed in the real system, using light-weight virtual machine (VM) containers.

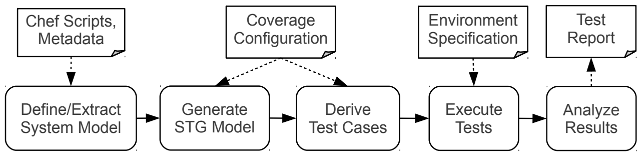


Figure 2: Model-based testing process.

4. DEMO OUTLINE

Throughout the demo, we showcase the capabilities of our testing framework based on selected real-world Chef recipes. The testing process is as follows:

- The Web user interface (UI) automatically lists the public Chef cookbooks from *http://cookbooks.opscode.com*. The demo will work with selected cookbook that we used in [7].
- The selected cookbook is automatically parsed for basic metadata, and a new test suite is initialized by running the Chef script with selected test settings in a clean VM container. (Note: The backend infrastructure executing the tests is deployed in a remote Cloud environment.)
- The gathered data are used to create and visualize STGs. The STG can be previewed with different test parameters and coverage settings (see Figure 3). State changes of each automation task are directly visible in the visualization.
- Next, we choose the test coverage settings and generate test cases. The test cases are stored to a database and queued for execution. The backend infrastructure spawns multiple parallel VM containers to execute the test cases.
- After test execution, detailed reports are visualized in the Web UI (see Figure 4). The UI reports which tests were (un-)successful, which pre- and post-states were registered for each task execution, which state changes were effected, and which tasks exhibit non-idempotent behavior.
- Finally, we take a closer look at the state capturing mechanism and illustrate how the system is able to detect state changes for arbitrary script resources (cf. Figure 1). (Note: we integrated *strace*-based OS-level system call tracing, extending the prototype in [7]).

5. RELATED WORK

Extensive research is conducted on automated software testing, however, most existing work and tools are not directly applicable to the IaC domain, for two main reasons: (i) IaC exposes fairly different characteristics than traditional software systems, i.e., idempotence and convergence; (ii) IaC needs to be tested in real environments to ensure that system state changes can be asserted accordingly. Such tests are hard to simulate, hence symbolic execution [2] has little practical value. Even though dry-run capabilities exist (e.g, Chef's *why-run*), they cannot replace systematic testing.

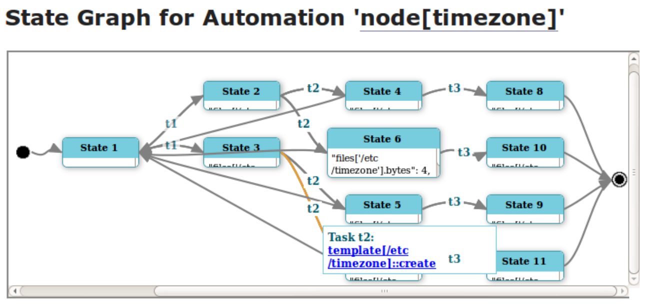
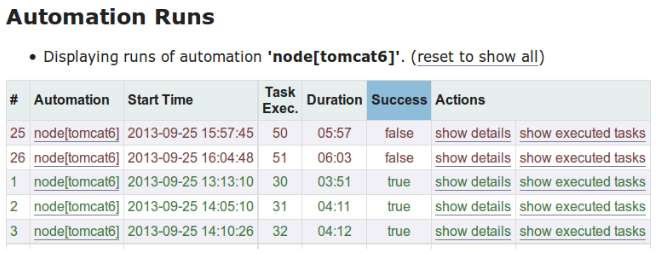


Figure 3: State Transition Graph in Web UI



Automation Runs

- Displaying runs of automation **'node[tomcat6]'**. (reset to show all)

#	Automation	Start Time	Task Exec.	Duration	Success	Actions	
25	node[tomcat6]	2013-09-25 15:57:45	50	05:57	false	show details	show executed tasks
26	node[tomcat6]	2013-09-25 16:04:48	51	06:03	false	show details	show executed tasks
1	node[tomcat6]	2013-09-25 13:13:10	30	03:51	true	show details	show executed tasks
2	node[tomcat6]	2013-09-25 14:05:10	31	04:11	true	show details	show executed tasks
3	node[tomcat6]	2013-09-25 14:10:26	32	04:12	true	show details	show executed tasks

Figure 4: Excerpt of Automation Test Results

Existing work has identified the importance of idempotence for building reliable distributed systems [4] and database systems [5]. Over the last years, the importance of building testable system administration [1] based on convergent models [12, 3] became more prevalent. More recently, IaC frameworks like Chef or Puppet heavily rely on these concepts. However, automated and systematic testing of IaC for verifying idempotence and convergence has received little attention, despite the increasing trend of automating system deployments, i.e., continuous delivery [6].

6. REFERENCES

[1] M. Burgess. Testable system administration. *Communications of the ACM*, 54(3):44–49, 2011.

[2] C. Cadar et al. Symbolic execution for software testing in practice: preliminary assessment. In *ICSE*, 2011.

[3] A. L. Couch and Y. Sun. On the algebraic structure of convergence. In *14th IEEE DSOM Workshop*, 2003.

[4] P. Helland. Idempotence is not a medical condition. *Queue*, 10(4):30:30–30:46, Apr. 2012.

[5] P. Helland and D. Campbell. Building on quicksand. In *CIDR*. www.cidrdb.org, 2009.

[6] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

[7] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam. Testing Idempotence for Infrastructure as Code. In *ACM/IFIP/USENIX Middleware Conference*, 2013.

[8] M. Hüttermann. *DevOps for Developers*. Apress, 2012.

[9] S. Nelson-Smith. *Test-Driven Infrastructure with Chef*. O'Reilly, 2011.

[10] Opscode. http://www.opscode.com/chef/.

[11] Puppet Labs. http://puppetlabs.com/.

[12] S. Traugott. Why order matters: Turing equivalence in automated systems administration. In *LISA*, 2002.

[13] M. Utting, A. Pretschner, and B. Legeard. Taxonomy of model-based testing approaches. *STVR*, 22(5), 2012.