ANSIBLE.tar.gz

Hans-Peter Dietz
h.p.dietz@gmail.com
@h_p_d
https://haensl.github.io

2016

Contents

1 Configuration Management

The practice of continuous integration and continuous delivery induced by lean and agile software development has gained traction within the last decade [40] [13]. One of the reasons for this trend being *"that long release cycles ("waterfall projects") have dramatically higher overhead than more frequently released ("iterative" or "agile") shorter cycles"* [34]. In order to achieve such short release cycles, tools and techniques are required to facilitate the process - so called continuous integration *(CI)*[1] and continuous delivery *(CD)*[2] automation tools.

This development away from traditional long term release cycles therefore is now replaced by *"stringing together services that run on a distributed set of computing resources and communicate over different networking protocols"* [16]. Wiring up such services manually can surely be done - booting up servers, SSHing to them and installing packages, configuring services etc. - but it is error prone and time-consuming [16].

While CI/CD tools facilitate the automation of the software release process, provisioning and maintenance of IT systems, including the ones running CI/CD tools, is the responsibility of configuration management solutions:

> *"The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle. Software Configuration Management involves identifying configuration items for the software project, controlling these configuration items and changes to them, and recording and reporting status and change activity for these configuration items."* [39]

Configuration Management therefore forms a discipline which encompasses evaluation, coordination and implementation of changes in artefacts used in construction and maintenance of software systems [39].

In practice this means that configuration management deals with tasks such as:

- Recording details about computer systems
- Installing and configuring the system *(e.g. networking)*
- Installing and configuring application software *(e.g. Docker [11])*
- Updating system software and patching security issues

In support of configuration management tasks a variety of different software solutions have been built. Some of the currently most popular solutions include:

PUPPET Established, Ruby based, open source industry standard in configuration management by PUPPET LABS [23]

[1] Continuous integration (CI): Build systems that watch source control, on change run tests and build the latest version. Examples include Jenkins [17] and Bamboo [2].

[2] Continuous delivery (CD): Systems which automatically *(e.g. triggered via SCM hooks or network events)* deploy application artefacts to any given environment *(e.g. QA, production)*.

CHEF Code-driven, Ruby based and git centered open source configuration management by CHEF SOFTWARE INC. [7]

SALTSTACK Highly modular and fast, Python based, open source configuration management solution and remote execution engine by SALTSTACK INC. [37]

ANSIBLE Easy-to-use, streamlined, YAML driven and Python based open source configuration management by RED HAT INC. [24]

Such software usually provides **"Infrastructure as Code"** *(IaC)* capabilities, meaning the definition of the configuration in machine-processable form, rather than physical hardware configuration or interactive configuration clients [42].

While each of the above mentioned solutions is equally capable and in turn outperforms the others in specific areas *(see section 2.5 for a comparison)* the remainder of this report will focus on ANSIBLE and should serve the reader as an introductory overview to the platform.

2 Ansible: Automation for everyone

As mentioned earlier, Ansible is a open source *configuration management* solution written in the Python programming language. The term configuration management solution is often synonymous to writing some sort of state description for machines and then using the tool to enforce that IT systems reflect the defined state [16]. Similar to other configuration management tools Ansible exposes a *domain specific language (DSL)* that is used to describe this desired state.

Apart from configuration management Ansible can also be used for *deployment*, referring to the process of compiling software into artefacts, running tests, copying the required files to server(s) and finally (re)starting the service(s).

Another aspect of deployment often mentioned is *orchestration*, meaning the management of multiple remote machines involved in the software release process in respect to the time domain. For example, it might be necessary to bring up a database server *before* booting up web servers or taking machines out of a loadbalancer *before* starting the update process. *Orchestration* is another thing Ansible is good at, as it is specifically designed to perform actions on multiple servers [16].

Finally, Ansible also handles *provisioning*, referring to the task of booting up new virtual machine instances [21].

> ***Author's Note:*** **What is an Ansible?**
>
> An Ansible is a fictional communication device that is capable of transferring information faster than light invented by the author Ursula K. Le Guin in her book Rocannon's World [19].

2.1 Characteristics

Human and machine readable syntax Ansible configuration management scripts are organised into so called *playbooks (see section 2.2)*. The syntax used within playbooks is derived from YAML - a data format language designed to be easy for humans to both read and write [3].

Agentless In contrast to most other CM tools, Ansible does not require an agent to be installed on managed machines. The only requirements Ansible brings forth are secure shell access[3] and Python 2.5 *(or later)* to be setup on the system.

SSH-based Ansible connects to managed machines via the secure shell cryptographic network protocol. This usually implies minimum setup efforts, since SSH is *"[...] a remote management framework that already exists natively on most server platforms"* [31].

Non-Root level access Using SSH as a network protocol enables Ansible to connect to remote hosts as any user. This facilitates multiple usage scenarios, e.g. there can be a dedicated `ansible` user, or changes to the system can be traced back to specific dev(Ops) personal. In case root access is required, Ansible ships with one of the most powerful privilege escalation methods available: `sudo`[4] [35].

Push-based Some of the CM tools that are based on agents use a pull-based approach in which the agents installed on the machines periodically check in with the central management server to pull configuration information and changes. The workflow looks roughly like this *(derived from [16])*:

1. Dev(Op)[5] changes configuration in management script
2. Dev(Op) pushes changes to central configuration management service
3. Agent periodically wakes up
4. Agent connects to central configuration management service
5. Agent pulls new configuration
6. Agent executes configuration management scripts locally to change server state

Ansible's push-based approach in contrast both simplifies this workflow *("No "managing the management"" [35])* as well as takes out the random time-dependency introduced in step 3 of the pull based workflow, giving the developer full control over when changes are applied to managed machines:

[3] Although SSH is the preferred method of accessing remote machines in Ansible, alternative network protocols can also be used *(see [26])*.

[4] `sudo` := *"Sudo (su "do") allows a system administrator to give certain users (or groups of users) the ability to run some (or all) commands as root while logging all commands and arguments. Sudo operates on a per-command basis, it is not a replacement for the shell."* [22]

[5] DevOps := A culture based around collaboration between software developers and other IT professionals on automation of software delivery and infrastructure changes. [20]

1. Dev(Op) changes playbook
2. Dev(Op) runs playbook
3. Ansible connects to machines and executes modules changing server state

Scaling down Ansible can be used to configure thousands of nodes or just one while scaling linear[6]. This means Ansible obeys Alan Kay's maxim of *"Simple things should be simple, complex things should be possible"* [15].

Modularized Architecture Ansible is based on a modularized architecture as opposed to monolithical approaches. Two principle types of modularisation are distinguished in Ansible: roles *(see section 2.2.3)* and modules *(see section 2.2)*. While roles provide a way to share and reuse plays written for a specific purpose *(e.g. setup mongo db)*, modules are used to perform tasks on managed machines. These modules are *declarative*[7] in nature, meaning that they are used to describe the desired state of the machine. As an example consider the following *declarative* statement to start the *nginx* service:

```
service name="nginx" state="started"
```

The example shows the `service` module being used to declare that the `nginx` load-balancer and web server is `started` on the remote machine.

No need for convergence In configuration management the term *convergence* introduced by Mark Burgess' CFENGINE [5] is an often discussed subject. Convergence in this context means that a configuration management tool is run multiple times to bring a machine into the desired state, each run bringing it closer to - *i.e. converging toward* - this state [16]. However, this idea does not apply to Ansible as it does not intent to be run multiple times in order to bring the machine(s) into the desired state. Instead a single playbook run puts machines into the therein described state of operation *(see [10] for a discussion on the topic by Michael DeHaan, the author of Ansible*).

Idempotent *Idempotence* in the configuration management context means that applying the exact same configuration multiple times to managed systems does not change the system state, but it either reaches or keeps the desired state [8]. It therefore stands in direct contradiction to aforementioned concept of *convergence*. All Ansible modules *should*[8] adhere to this principle. Development guidelines encompass the policy that *"...modules [...] strive to be 'idempotent', meaning they will only make changes when the desired state expressed to the module does not match the current state"* [27].

[6] $\Theta(n)$, where n is the number of modules.

[7] *Declarative* vs: *commanding* statements: *"service x [is] started"* vs. *"start service x"*.

[8] Ansible ships with a set of core modules which are guaranteed to adhere to *idempotence*. Apart from those core modules, a considerable amount of community contributed modules and roles can be accessed and integrated easily via the Ansible-Galaxy repository. [36]. When using these third-party components it is advisable to read their documentation and code carefully to make sure they do not violate this principle.

Thin layer of abstraction As opposed to other configuration management tools Ansible provides only a thin layer of abstraction. This means that while other tools abstract operating system specifics, Ansible does not. For example, there is no abstract module for package management, but instead specific modules targeting the respective OS are available, such as `yum`[9] or `apt`[10]. Though it is possible to write Ansible roles and playbooks in a way to handle different operating systems, it is not a widely adopted practice to do so.

File based project layout Ansible uses an implicit project structure convention defined on the filesystem level. This means that Ansible looks if specific directories and files are present and if so evaluates and incorporates their content *(see listing 5)*.

Dynamic inventory support Ansible supports statically as well as dynamically defined inventories *(see section 2.2.1)*. An inventory in this context describes the host machines Ansible operates on. Both approaches can be combined, enabling Ansible to manage statically configured machines on an internal company network as well as dynamically booted instances on an arbitrary cloud computing platform like Amazon's Elastic Compute Cloud [1].

2.2 The Basics

Ansible follows a "play" - as in *on a stage* - analogy:

Plays are the unit of granularity that associate *tasks* with *hosts*.

Tasks are units of granularity that use *modules* to describe desired system state. Tasks are executed in sequential order as defined in the *play*.

Hosts are the machines a *task* is applied to.

Modules are used to declare the desired system state.

Playbooks are collections of *plays*.

Consider the example playbook in listing 1. Within this excerpt of a playbook that configures a mongo db setup, different plays are listed. The first one is named *"Install mongo db"* the second one *"Setup replica set"*. The dashes in front of the names are YAML syntax for an array of things - namely an array of *plays* which is how a *playbook* is defined. Each play is acted out on the *hosts* referenced in the play - `mongo_servers` in the example, which is a *variable* referencing a group of servers defined in the *inventory (see section 2.2.1)*. Following, it is indicated that the *play* requires root privileges and some *variables* are defined. Concluding is the list of *tasks* which form the play:

[9] yum := *Yellowdog Updater, Modified*, a cli package manager for RPM based Linux distributions.

[10] apt := *Advanced Package Tool*, set of core tools for package management on Linux systems of the Debian family.

```
---
# mongodb.yml
# play: associates tasks with hosts
- name: Install mongo db # name of the play
  hosts: mongo_servers # hosts to apply the tasks to
  sudo: True # this play does require root access
  vars: # variable definitions
    mongo_repo_key_url: "http://..."
    mongo_db_repo_url: "deb http://...."
  tasks: # list of tasks to be executed sequentially
    - name: Add mongodb repository key # task name
      apt_key: url={{ mongo_repo_key_url }} # module declaration

    - name: Add mongodb repository
      apt_repository: repo={{ mongo_db_repo_url}} update_cache=yes

    - name: Install mongodb
      apt: mongodb-org state=installed

- name: Setup replica set
# ...
```

Listing 1: Example Playbook: MongoDB setup

`Add mongodb repository key` The `apt_key` *module* is used to add the authentication key of the mongo db repository. It is supplied with one *argument*, the `url` to the key which is stored in a *variable* called `mongo_repo_key_url`.

`Add mongodb repository` The `apt_repository` *module* is used to add the mongo db repository. It is supplied two *arguments*, the `url` to the repository, which is again stored in a *variable*, and `update_cache` is set to `yes`, indicating that the package cache should be updated after adding the repository.

`Install mongodb` The `apt` module is used to declare that the package named `mongodb-org` should be `installed` on the system.

See figure 2.1 for a visualisation of the relationships between playbook, plays, hosts, tasks and modules.

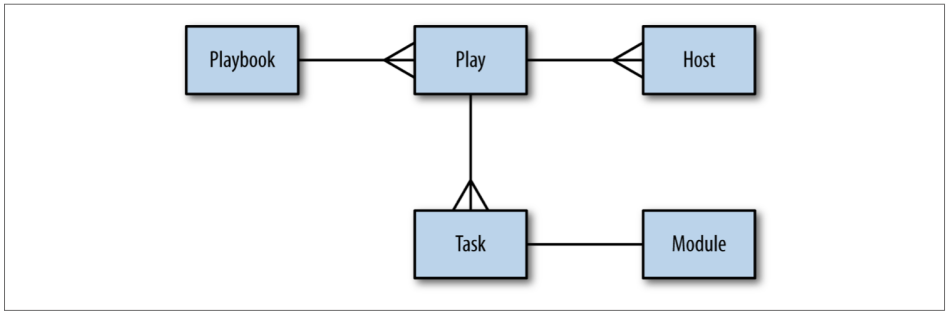

Figure 2.1: Entity relationship between playbook, plays, hosts, tasks and modules [16].

```
# hosts
[mongo_servers] # group definition
mongodb.somehost.example.com # host
mongodb.anotherhost.example.com
216.81.59.173

#...
```

Listing 2: Example of a static inventory file

2.2.1 Inventory

As has been mentioned in the example above, Ansible plays are executed against specific *hosts*. These hosts can either be specified via *static* inventory files or *dynamic* inventory scripts.

Static inventory files can be specified on system level or on a per project basis in a `INI`-like format. The system wide configuration file is usually stored at `/etc/ansible/hosts`, while for a per project specification a file named `hosts` is used which is placed at the root folder of the project, i.e. as a sibling to the playbook. See listing 2 for an example.

Dynamic inventory scripts are executable scripts, usually decorated by a *shebang*, that upon execution return host specification in *JSON*[11] format. This makes it easy to manage Amazon EC2 [1] instances or Cobbler [6] maintained installations via Ansible *(see [28] for further details)*.

2.2.2 Variables, Facts & Templates

Variables Ansible allows the definition of variables in a variety of contexts and locations. The simplest form are variable definitions in plays *(see the example in listing 1)*. Apart from those, variables can also be specified in a multitude of different files:

`hosts` It is possible to define *(and dereference)* variables in inventory files.

`vars/` Play(book) and role specific variables should be placed in a YAML file within a directory named `vars` *(see listings 3 & 5)*.

`defaults/` The `defaults` directory can contain YAML variable files which are meant to be default values and therefore likely to be overridden by more specifically targeted variable definitions.

`host_vars/` This directory is meant to harbour YAML variable files targeting specific hosts. A file containing such definitions intended to be available *(and override otherwise set values)* on a specific host, must be named after the host url specified in the inventory.

[11] JSON := JavaScript Object Notation, a data interchange format based on the JavaScript object literal notation. [9]

```
# ~/playbooks/myplaybook
./roles/mongodb/tasks/main.yml # tasks
./roles/mongodb/files/ # holds files related to the role
./roles/mongodb/templates/ # holds jinja2 templates
./roles/mongodb/handlers/main.yml # handlers
./roles/mongodb/vars/main.yml # role specific variables
./roles/mongodb/defaults/main.yml # default variables
./roles/mongodb/meta/main.yml # dependency information and metadata
```

Listing 3: Example role file layout: Mongo DB

```
---
- hosts: mongo_servers
  roles:
    - mongodb
```

Listing 4: Example role based playbook: Mongo DB

`group_vars/` This directory can hold YAML variable files that target specific groups, as defined in the inventory. Filenames must thereby mirror group names.

`command line` When invoking `ansible[-playbook]`, variables can be passed as command line arguments. Any definitions given this way have the highest possible precedence.

Facts are information Ansible gathers about managed machines. They are readily available[12] for use within playbooks just like any other variables. See [29] for an overview of available facts.

Templates Ansible supports the JINJA2 templating language. JINJA2 is a Python based templating language modelled after Django's templates. It supports a variety of control structures, such as loops and conditionals, as well as filters. The JINJA2 implementation that comes with Ansible, is furthermore enhanced via additional, Ansible specific and exclusive filters *(see [30])*.

2.2.3 Roles

Since Ansible follows a modular approach, components within the system strive to be reusable. *Roles* are reusable components with a granularity level roughly equivalent to *plays* enabling upscaling by breaking a playbook into multiple files. Roles are created by adding a *roles* directory at the project root level wherein roles are in turn defined in their own subdirectories. See listing 3 for an exemplary role structure of our mongo db example demonstrating the implicit role file layout convention. Many community contributed roles can also be integrated via the `ansible-galaxy` cli and repository [36].

After splitting up the tasks and variables from the playbook example of listing 1 it can be refactored to just a few lines *(see listing 4)*.

[12] If `fact_gathering` has not been deliberately disabled.

2.3 Operation

2.3.1 Setup

The Ansible application bundle is readily available through default package management systems on most Linux operating systems. If not available it can be cloned and built from source via their GitHub repository [32].

2.3.2 Command Line Interface (CLI)

Once installed on the control machine, the base installation makes several different commands available:

`ansible` is used to execute ad-hoc commands. The following example demonstrates invocation of the `ping` module to check connectivity to the host `mongodb.somehost.example.com`:

```
ansible mongodb.somehost.example.com -m ping
```

`ansible-playbook` is used to execute a playbook.

```
ansible-playbook my-playbook.yml
```

`ansible-doc` is used to access man-page like documentation for modules. The following example displays the documentation for the `file` module:

```
ansible-doc file
```

`ansible-galaxy` is used to search for, install, create and publish roles to and from the Ansible Galaxy repository [36]. The following command searches for roles for the Elasticsearch [14] search engine:

```
ansible-galaxy search elasticsearch
```

2.3.3 Configuration

Many of the behaviours and settings of the Ansible engine can be changed and adapted to one's need. Ansible supports both system-wide configuration as well as a per project configuration via environment variables or configuration files located at the user's home folder or at the project root. Precedence is given in the following order:

1. `$ANSIBLE_CONFIG` - environment variable
2. `./ansible.cfg` - configuration file at the project root
3. `/.ansible.cfg` - configuration file in the user's home folder
4. `/etc/ansible/ansible.cfg` - system wide configuration file

For detailed documentation on the available configuration options see [25].

```
ansible.cfg # project scoped configuration
files/
  some.conf # files related to the playbook
handlers/
  main.yml # handler definitions
group_vars/
  dbservers.yml # variable definitions for the dbservers inventory
    group
host_vars/
  mongodb.somehost.example.com.yml # variable definitions for specific
    hosts
hosts # inventory
roles/
  mongodb/
    defaults/
      main.yml # default variables for the mongodb role
    files/
      some.conf # files related to the role
    handlers/
      main.yml # handler related to the role
    meta/
      main.yml # metadata related to the role
    tasks/
      main.yml # tasks related to the role
    templates/
      some.template.j2.conf # templates related to the role
    vars/
      main.yml # variables related to the role
templates/
  some.template.j2.conf # templates related to the playbook
vars/
  main.yml # arbitrary variables related to the playbook
```

Listing 5: Playbook file structure

2.3.4 Playbook Handling

As has been mentioned earlier, Ansible follows a implicit file layout convention to structure it's playbooks *(see listing 5)*. When it is executed Ansible generates a Python script from the contents, connects to the host(s) via SSH and transfers the script. The script is then executed on the remote machine(s) while Ansible waits for it's completion. Each task therein is executed in parallel on all[13] hosts defined in the inventory, but execution is not pipelined[14], i.e. Ansible waits until one task is finished on each host before starting the next.

2.3.5 Maintainance

When maintaining machines which have been setup via Ansible, e.g. updating packages, changing configuration, etc, all that has to be done is updating and rerunning the playbook. The idempotent nature of the tool will only update necessary units and leave everything else untouched.

[13] *"By default, Ansible will try to manage all of the machines referenced in a play in parallel."* [25]

[14] Indiscriminate execution is disabled by default, see [25] for details.

2.3.6 Advanced Features & Usage Scenarios

Apart from the thus far mentioned features and use cases Ansible offers more advanced techniques and features, some of which are listed here in a nutshell:

Handlers are basically tasks triggered by other tasks. They are mainly used to restart services after configuration changes. Handlers are only ever executed once even if notified by multiple different tasks.

Registers are a way to capture the output of module invocations into variables for later use within a play.

Local Actions are tasks executed on the machine running Ansible.

Shebang Ansible introduces it's own shebang[15]: `#!/usr/bin/env ansible-playbook`

Bash scripts replacement Many of the tasks within DevOps usually implemented in Bash scripts, e.g. deployment of Docker containers, can easily be done via Ansible reducing complexity, portability as well as error potential.

Trigger CI/CD Ansible can trigger and potentially replace CI/CD tools, as plays and playbooks are not restricted to configuration of machines, but capable of executing arbitrary tasks. This can decrease the number of necessary utilities, make the tool-landscape more homogenous and reduce learning overhead in personal.

Vagrant integration The Vagrant virtual machine provisioning tool already includes Ansible as a provisioning option. Funny enough, both tools disagree on their respective responsibilities: Vagrant regards Ansible as a provisioner, while Ansible defines provisioning as the act of booting up virtual machines, therefore making Vagrant the provisioner.

Jobs It's versatility, shebang support and non-agent architecture makes Ansible a predestined tool to orchestrate and execute jobs.

Ansible Tower is a elaborate commercial interface to Ansible offering advanced monitoring, user management, scheduling and UI to the Ansible CLI [33].

2.4 Showcase: vm-php

To showcase a complete example of an Ansible project, the interrested reader is pointed towards a VAGRANT provisioning profile for a PHP development environment located at: https://github.com/haensl/vm-php

[15] shebang := a special preamble consisting of `#!` followed by an interpreter directive allowing the program loader to hand the script to the interpreter it was intended for

2.5 Performance vs. competitors

A comparison of pros and cons of Ansible, Chef, Salt and Puppet compiled from [12] and [41] can be found in figure 2.2. For further information on how the different tools perform and where they strive the interested reader is also pointed towards [18] and [4].

	Ansible	Chef	SaltStack	Puppet
Strengths	• SSH-based • Easy learning curve • Streamlined code base • Agentless	• Many modules • Code driven • Git centered	• Featurerich DSL • Good introspection • High scalability • High resiliency	• Well established • Complete UI • Strong reporting
Weaknesses	• Poor introspection • Poor performance speed	• Steep learning curve • No push functionality • Vague documentation	• Difficult Setup • Basically Linux-only • Weak reporting	• Steep learning curve • Codebase can grow complex

Figure 2.2: Strengths and Weaknesses of Ansible, Chef, Salt and Puppet [12] [41]

Author's Note: **Watch & Learn**

For more insights on how the different solutions perform and compare, the author recommends watching Animesh Singh, Daniel Krook and Paul Czarkowski in their *OpenStack* talk: *Chef vs. Puppet vs. Ansible vs. Salt - What's Best for Deploying and Managing OpenStack?* [38].

3 Conclusion

Ansible is a powerful tool capable of far more than *just* configuration management. It complements infrastructures based on Docker and Vagrant and is easy to learn and use. It's modular approach makes way for a growing community of role and module developers sharing their work and improving the platform every day. Although Ansible does not outperform it's competitors, the author has grown fond of the tool during his learning journey for this report and can recommend it as a *jack-of-many-trades*-utility.

References

[1] AMAZON WEB SERVICES, INC. Amazon Elastic Compute Cloud. https://aws.amazon.com/ec2/, 2016. Accessed on July 3rd, 2016.

[2] ATLASSSIAN CORP. Bamboo - Build, Test, Deploy. https://www.atlassian.com/software/bamboo, 2016. Accessed on July 2nd, 2016.

[3] BEN-KIKI, O., EVANS, C., AND INGERSON, B. YAML Ain't Markup Language (YAML™) Version 1.1. *yaml. org, Tech. Rep* (2005).

[4] BERTRAM, A. *Choosing The Right Configuration Management Tool*. http://www.tomsitpro.com/articles/configuration-management-tools,2-920.html, June 2015. Accessed on July 2nd, 2016.

[5] BURGESS, M., ET AL. CFEngine: a site configuration engine. In *USENIX Computing systems, Vol* (1995), Citeseer.

[6] CAMMARATA, JAMES AND MAAS, JÖRGEN AND DEHAAN, MICHAEL. Cobbler - linux installation server. http://cobbler.github.io/, 2016. Accessed on July 3rd, 2016.

[7] CHEF SOFTWARE, INC. Chef - Platform Overview. https://www.chef.io/chef/, 2016. Accessed on July 2nd, 2016.

[8] COUCH, A., AND SUN, Y. On the algebraic structure of convergence. In *International Workshop on Distributed Systems: Operations and Management* (2003), Springer, pp. 28–40.

[9] CROCKFORD, D. *JavaScript: The Good Parts: The Good Parts*. " O'Reilly Media, Inc.", 2008.

[10] DEHAAN, M. Idempotence, convergence, and other silly fancy words we use too often. https://groups.google.com/forum/#!topic/ansible-project/WpRblldA2PQ, 11 2013. Accessed on July 3rd, 2016.

[11] DOCKER INC. Docker - Build, Ship, Run. https://www.docker.com, 2016. Accessed on July 2nd, 2016.

[12] DREYFUSS, J. Takipi Blog. *Deployment Management Tools: Chef vs. Puppet vs. Ansible vs. SaltStack vs. Fabric*. http://blog.takipi.com/deployment-management-tools-chef-vs-puppet-vs-ansible-vs-saltstack-vs-fabric/, 8 2015. Accessed on July 2nd, 2016.

[13] DYBÅ, T., AND DINGSØYR, T. Empirical studies of agile software development: A systematic review. *Information and software technology 50*, 9 (2008), 833–859.

[14] ELASTICSEARCH™. Elasticsearch - Search & Analyze Data in Real Time. https://www.elastic.co/products/elasticsearch, 2016. Accessed on July 4th, 2016.

[15] FELDMAN, S., AND KAY, A. C. A conversation with Alan Kay. *ACM Queue 2*, 9 (2004), 20–30.

[16] HOCHSTEIN, L. *Ansible: Up and Running*. " O'Reilly Media, Inc.", 2014.

[17] JENKINS™. Jenkins - Build great things at any scale. https://jenkins.io, 2016. Accessed on July 2nd, 2016.

[18] LANE, R. *Moving away from Puppet: SaltStack or Ansible?* http://ryandlane.com/blog/2014/08/04/moving-away-from-puppet-saltstack-or-ansible/, August 2014. Accessed on July 2nd, 2016.

[19] LE GUIN, U. K. *Rocannon's World*. Ace Books, 1966.

[20] LOUKIDES, M. *What is DevOps?* O'Reilly Media, Inc., 2012.

[21] MIETZNER, R., AND LEYMANN, F. Towards provisioning the cloud: On the usage of multi-granularity flows and services to realize a unified provisioning infrastructure for saas applications. In *2008 IEEE Congress on Services-Part I* (2008), IEEE, pp. 3–10.

[22] MILLER, T. C. Sudo. https://www.sudo.ws, 2016. Accessed on July 4th, 2016.

[23] PUPPET LABS. Puppet - Opensource configuration management. https://puppet.com, 2016. Accessed on July 2nd, 2016.

[24] RED HAT, INC. Ansible - Automation for everyone. https://www.ansible.com, 2016. Accessed on July 2nd, 2016.

[25] RED HAT, INC. Ansible Documentation - Configuration File. http://docs.ansible.com/ansible/intro_configuration.html, 2016. Accessed on July 4th, 2016.

[26] RED HAT, INC. Ansible Documentation - Configuration File: *transport*. http://docs.ansible.com/ansible/intro_configuration.html#transport, 2016. Accessed on July 4th, 2016.

[27] RED HAT, INC. Ansible Documentation - Developer Information: *Developing Modules - Conventions/Recommendations*. http://docs.ansible.com/ansible/developing_modules.html#conventions-recommendations, 2016. Accessed on July 3rd, 2017.

[28] RED HAT, INC. Ansible Documentation - Dynamic Inventory. http://docs.ansible.com/ansible/intro_dynamic_inventory.html, 2016. Accessed on July 3rd, 2016.

[29] RED HAT, INC. Ansible Documentation - Information discovered from systems: *Facts*. http://docs.ansible.com/ansible/playbooks_variables.html#information-discovered-from-systems-facts, 2016. Accessed on July 5th, 2016.

[30] RED HAT, INC. Ansible Documentation - Jinja2 filters. http://docs.ansible.com/ansible/playbooks_filters.html, 2016. Accessed on July 5th, 2016.

[31] RED HAT, INC. Ansible In Depth. Tech. rep., Red Hat, Inc., 2016.

[32] RED HAT, INC. Ansible on GitHub. https://github.com/ansible/ansible, 2016. Accessed on July 4th, 2016.

[33] RED HAT, INC. Ansible Tower - Mission Control for Ansible. https://www.ansible.com/tower, 2016. Accessed on July 7th, 2016.

[34] RED HAT, INC. Continuous integration and continuous delivery with Ansible. Tech. rep., 2016.

[35] RED HAT, INC. The Benefits of Agentless Architecture. Tech. rep., Red Hat, Inc., 2016.

[36] REDHAT, INC. Ansible Galaxy - Your hub for finding, reusing and sharing the best Ansible content. https://galaxy.ansible.com/, 2016. Accessed on July 3rd, 2016.

[37] SALTSTACK, INC. SaltStack - Automation for enterprise IT ops, event-driven data centre orchestration and the most flexible configuration management for DevOps at scale. https://saltstack.com/, 2016. Accessed on July 2nd, 2016.

[38] SINGH, A., KROOK, D., AND CZARKOWSKI, P. Chef vs. Puppet vs. Ansible vs. Salt - What's Best for Deploying and Managing OpenStack? https://www.openstack.org/summit/tokyo-2015/videos/presentation/chef-vs-puppet-vs-ansible-vs-salt-whats-best-for-deploying-and-managing-openstack, 10 2015. Accessed on July 5th, 2016.

[39] TEAM, C. P. D. CMMI for Systems Engineering/Software Engineering, Version 1.02, Staged Representation (CMMI-SE/SW, V1.02, Staged). Tech. Rep. CMU/SEI-2000-TR-018, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[40] VAN DER HAAGEN, E. Cloudbees - Emily van der Haagen's blog. *2015 Predictions for Continuous Integration and Continuous Delivery*. https://www.cloudbees.com/blog/2015-predictions-continuous-integration-and-continuous-delivery, 12 2014. Accessed on July 2nd, 2016.

[41] VENEZIA, P. InfoWorld - *Review: Puppet vs. Chef vs. Ansible vs. Salt*. http://www.infoworld.com/article/2609482/data-center/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.html, 11 2013. Accessed on July 5th, 2016.

[42] WITTIG, A., AND WITTIG, M. *Amazon Web Services in Action*. Manning Publications Co., 2015.