Infrastructure-as-Code
Ansible vs Puppet

George Rezkalla - rezkalla@kth.se
Nour Alhuda Almajni - almajni@kth.se

Contents

1 Introduction

1.1 Why this topic?

In order to answer this question, we need to first understand what Ansible and Puppet are. Ansible and Puppet are tools that implements a concept called *Infrastructure-as-Code* (*IaC* from now on).

IaC is simply writing the configuration of machines, whether they are bare-metal or virtual machines, in form of machine-readable code by commonly combining imperative code with declarative code. Moreover, IaC aims also to automate provisioning consistent environments for all the nodes (or components) of the system by running the same configuration files across all of them from one or more central locations that can have all the information about all the nodes of the system. That is why IaC contributes to the continuous delivery and continuous deployment in DevOps, and IaC tools are also called *configuration management* (*CM* from now on) tools [1].

So, the question is: what is configuration management in general?

1.2 What is configuration management?

Configuration management is a term that has to do with systems engineering process in general. It is about configuration of any system or product in general to be consistent with what is intended regarding the system's or products requirements, design and operations throughout its life cycle. In other words, it is a more general term that encompasses any type of product, including software [2].

That is why IaC is considered a way of implementing configuration management in software development life cycle processes [2, 1].

1.3 The relationship between IaC and DevOps

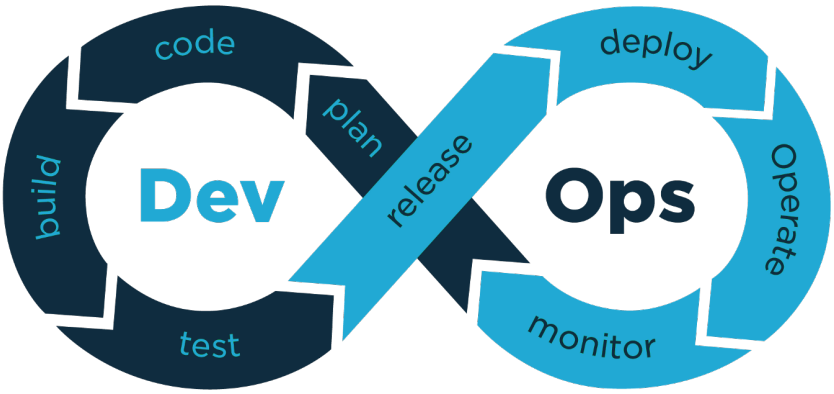

Figure 1: DevOps phases. Taken from: [3].

DevOps is a set of practices that aim to help making the development team and IT operations team work closer to each other to shorten the development life cycle of software and continuously deliver (and/or continuously deploy) high software quality [4].

IaC make the development and operations team work closer because they make the development team participate more in defining the configurations of the system and the operations team participate more in the development process in earlier stages in the software development life cycle. This achieves two of the main goals of DevOps [1].

In summary, IaC tools help with the provisioning automation (in other words, continous delivery and/or continuous deployment) and lead to more collaboration between the development and operations teams, which makes them an important and useful part of DevOps [1].

1.4 Some tools

There are a number of IaC tools that are widely used. Some examples are: Puppet, Ansible, Pulumi, Chef, Otter, SaltStack, CFEngine, Terraform, DSC and Paco [1]. In this essay, we are going to focus on Puppet and Ansible.

2 Puppet

Puppet is an open source IaC tool that is built on C++, Clojure and Ruby. There is a paid version of Puppet called Puppet Enterprise that also provides a GUI and some other things [5]. Puppet is also a very famous IaC technology that is used in Google, Twitter, the New York Stock Exchange. It helps to automatically configure and maintain a large number of machines with one or many system administrators [6].

In general, Puppet uses a client-server architecture. In Puppet's terminology, the server is called "master" and the client is "agent" or "node". There can be one or more masters that handles a large number of agents. Once you changed something (a configuration) in the Puppet master such as updating packages or configuration or adding new users, the Puppet agents will automatically pull the updates from the Puppet master if they didn't match the new configuration. Otherwise, it does nothing [6].

In the following sections, we will explain the architecture of Puppet and some of its main components in more detail.

2.1 Some technical information

First, there are some important concepts/terminology in Puppet that we need to understand:

2.1.1 Facts

Facts are the system information of agents such as the operating system, IP address, host name and some hardware information of the agent machines [5, 6]. The Puppet agent will generally use a program/external tool that can collect this information. This program is called "Facter". Many organizations use their own program or plugin of Facter because Facter is easily extensible [6].

2.1.2 Manifest

Manifests are the files that contain the Puppet code where administrators write the configurations that all the components of the system should have. These manifests are collected in related groups called "modules" [6]. A module can also contain some other things than manifests, such as text files and templates. Two examples of what manifest files can contain are in listing 1 and listing 2, and will be explained later in more detail.

2.1.3 Catalog

A catalog is a graph or a document that contains information about resources (packages, services, etc.) that we want to have on the agents, their desired states on the agents and the dependencies between those resources [7, 6].

The catalog will be compiled later by the Puppet master using the manifests and the Facts that it get from the Puppet agent's Facter [8]. This compiled catalog will then be sent back to the Puppet agents to be applied their. More details about this comes in the next section.

2.1.4 How does Puppet work?

So now after we understood the main terminology used in Puppet, let's sum up how Puppet actually works. According to [6, 5], and as you see in figure 2, there are 3 main steps in the communication between the master and the agents:

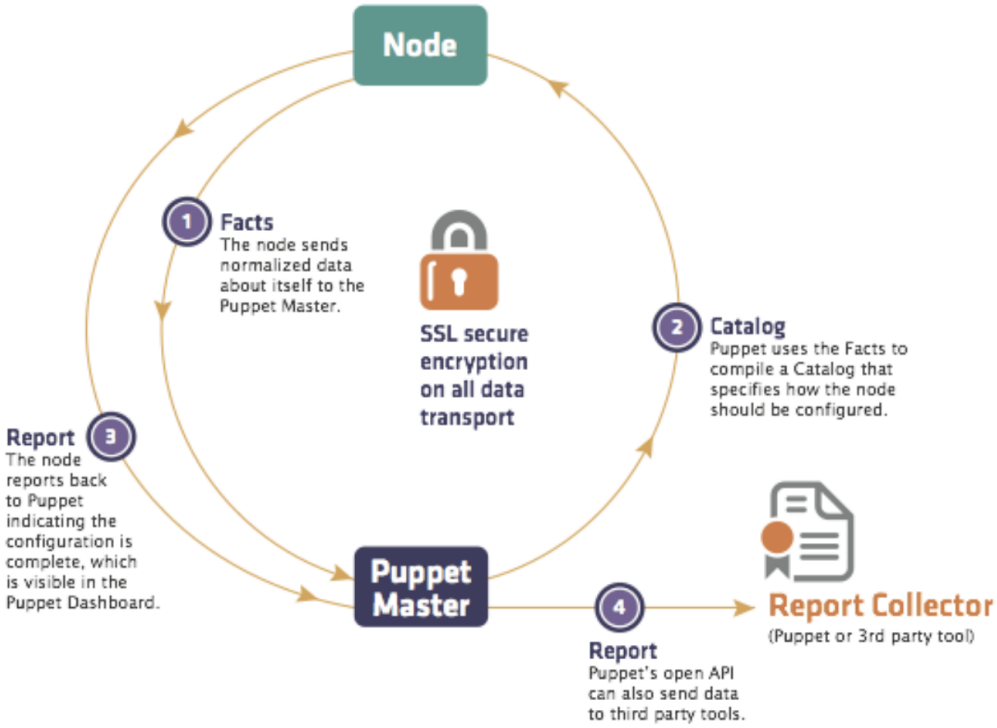

Figure 2: Puppet steps. Taken from: [6].

```
file { "/etc/hello-world.txt":
    ensure => file ,
    content => "Welcome to this machine",
    mode => 644
}
```

Listing 1: Example for a manifest file. This example is adapted from: [6].

1. The Puppet agent sends some information about itself (the facts) to the Puppet master.
2. The Puppet master compiles a catalog for that particular Puppet agent using facts (that it got in the previous step) and manifests/modules in the Puppet master. Then, the Puppet master sends this compiled catalog back to the Puppet agent.
3. The Puppet agent applies the compiled code (the configuration) sent to it from the Puppet master and send back a report where it tells the Puppet master if everything worked as intended or not. If the Puppet agent is already in the desired state that is obtained as a result of executing the compiled code, the Puppet agent will do nothing, which makes the transactions idempotent.

Note Each Puppet agent in the system may get different compiled catalog based on its operating systems and/or some other configurations in the agent.

However, there is an important step that needs to happen before these 3 steps; the authentication. Puppet comes with a built-in certificate authority (CA) and tools for public key infrastructure (PKI) that can be used for all its secure socket layer (SSL) communications between the Puppet masters and the Puppet agents. Puppet allow the use of an external existing CA to do this kind of communication.

The CA is one of the services that are located in Puppet master and these certificates that are issued from the CA are important to verify the identity of the Puppet agents. When a Puppet agent wants to communicate with Puppet master for the first time, the Puppet agent requests a certificate, and if the Puppet master decides that the request is safe, it will send the certificate to the agent. Now, the Puppet agent will know that it can trust the Puppet master and it will use this certificate later when it wants to request a catalog to identify itself to the Puppet master [9].

2.1.5 Example

It is quit easy to understand Puppet code (or code that is written in Puppet manifest files). For example, the code in listing 1 means the following: The administrator wants to ensure that the file called "hello-world.txt" exists in the directory `/etc/` (if not, the Puppet agent will create it). This file should also contain the sentence "Welcome to this machine". The desired access permissions mode for this file should be 644 (as indicated by chmod syntax).

Another easy example can be seen in listing 2 where we can easily understand that the administrator wants to ensure that the mysql-server package is installed on all Puppet agents.

```
1 package { 'mysql-server':
2     ensure => installed
3 }
```

Listing 2: Another example for a manifest file. This example is adapted from: [10].

3 Ansible

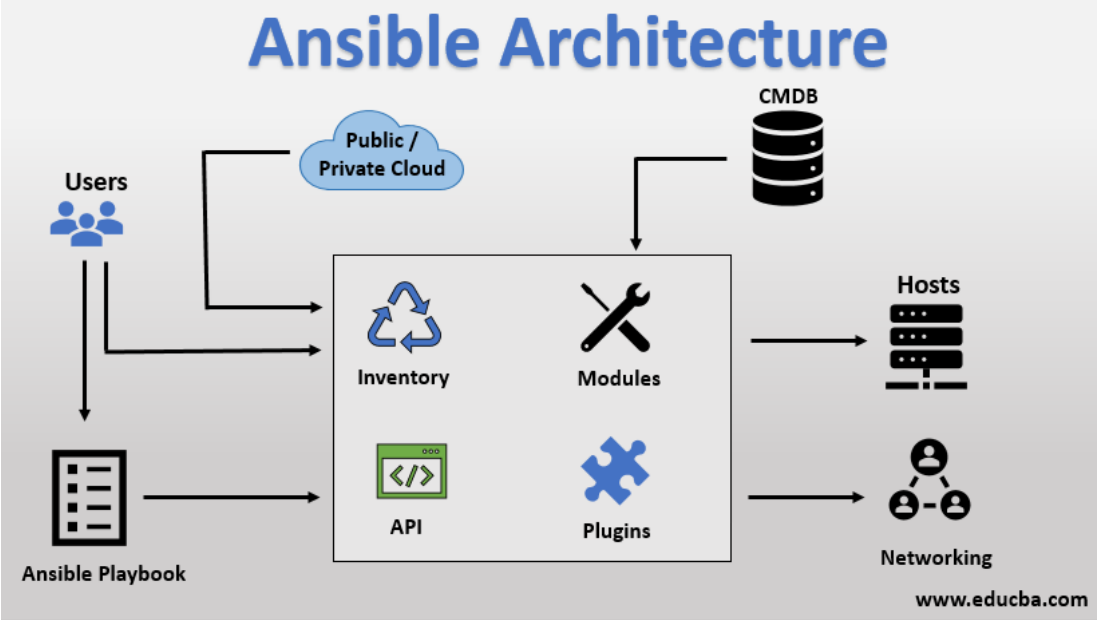

Figure 3: Ansible architecture. Image source: [11]

.

Ansible is an IaC tool that facilitates defining infrastructure using configuration files that are human-readable YAML files. It is also an IT automation tool that facilitates automating the execution of tasks across all nodes of a system. These tasks could be anything from installing the configuration files to orchestrating advanced tasks such as deploying software in zero-down time manner [12].

3.1 Some technical information

Ansible is designed to manage machines (whether bare-metal or virtual machines) in an agent-less manner. All the nodes in a system need only have an SSH software for communication by default (although Ansible could also be configured to use other communication methods). Ansible needs only to be installed on one control node (or multiple control nodes) that will then use SSH to communicate with all other nodes in the system. The nodes that are controlled are called managed nodes [12, 13, 14, 15].

The control node is the source machine of the configurations that are pushed from the control

```
1  mail.example.com
2
3  [webservers]
4  foo.example.com
5  bar.example.com
6
7  [dbservers]
8  one.example.com
9  two.example.com
10 three.example.com
```

Listing 3: INI hosts (inventory) file taken from [19]. As seen here, we have five servers, where foo.example.com and bar.example.com are grouped under webservers group, and one.example.com, two.example.com and three.example.com are grouped under dbservers group.

node to the managed nodes in order to get installed on those managed nodes [13]. That is why Ansible is said to follow the "push" workflow [16]. However, Ansible could also be configured to have an inverted (i.e. "pull") architecture [17].
To be more concrete, we will discuss the general architecture of the Ansible in the following sections.

3.2 Ansible architecture

Figure 3 shows sn overview of Ansible architecture. In the following sections, we will cover the most important components of the architecture.

3.2.1 Inventory

In Ansible's terms, an inventory refers to the managed nodes (or targeted hosts) that the control node(s) manage(s). By default, the IP-addresses or URLs managed nodes are saved locally on your control machine in an INI (or YAML-formatted) file called *hosts* whose path is `/etc/ansible/hosts`. The managed nodes can also be grouped to make it more easily accessable when only some of the managed nodes wants to handled. However, an inventory can also be pulled from remote data sources, such as EC2 [18, 15, 19]. An example of inventory file is in listing 3.

3.2.2 Modules

Modules are programs (or scripts) that are pushed by Ansible (over SSH by default) to all or some hosts that are defined in the inventory. Most modules are seen as "resource models" that describe the desired end-state of the system. They is done by passing in the desired set of parameters that describe the desired state of system [15, 18, 20]. That is why most modules of Ansible are idempotent, which means that regardless of how many times a module is executed on a managed node, the end result or state of the system should be the same [17]. This will become handy later when we talk about playbooks in section 3.2.4.

Ansible already comes with a large number of modules, but you can your own as well. If one wants to develop their own modules, they can use "module utilities" that are shared (common) pieces of code that provide helper functions for usage to avoid code duplication. Moreover,

Your modules can be saved and fetched from anywhere and they do not necessarily need to be save locally on the control nodes. A module can be executed as a single task using ad-hoc commands or from your playbook (see section 3.2.4) [15, 18, 20, 21].

3.2.3 Plugins

Plugins are code snippets that extend the core functionality of Ansible. The difference between plugins and modules is that modules are programs that are pushed and executed on managed nodes (normally, remote hosts) whereas plugins are programs that execute locally on the control node within the `/usr/bin/ansible` process. To be more concrete, plugins extend Ansible's core functionality that is about e.g. data transformation, logging, communication with hosts defined in the inventory. Ansible comes with a number of plugins. However, you can write your own as well [18].

3.2.4 Playbook

Playbooks are the core of how Ansible works. A playbook is a YAML file that consists of a number "plays". Each play maps all or some managed nodes to a number of tasks that can either be defined in the same playbook file or can be grouped in other re-usable files called "roles". Each task executes one module (see section 3.2.2) [17].

A playbook is run using the command

```
ansible-playbook <PLAYBOOK_NAME>.yml
```

where `<PLAYBOOK_NAME>` stands for the name of the playbook YAML file. You can check the syntax of the YAML file using `--syntax-check` flag. You could even execute what is called a "dry-run" using the command `--check` flag, which tries to predict what changes may occur if the playbook file was to be executed [17, 22].

The power of playbooks resides in the their readability and in most modules that are used in the playbooks being idempotent (as mentioned in section 3.2.2). This will make a playbook whose modules are idempotent also idempotent, which means that regardless of how many times a playbook is run on the managed nodes, the end result or state of those nodes will be the same. This makes playbooks handy in cases when there are some tasks that are changed in the playbook and want to be pushed to the managed nodes. The solution is simply to make the desired changes in the playbook and re-run it again, and the changed tasks are the only ones that will change the state of the managed nodes and the other non-changed tasks will execute again without changing the state of the system regarding those non-changed tasks [17]. More about this will be in the following example.

3.2.5 Example

An example of a playbook file is defined in listing 4. Let's go through how this YAML file is structured step-by-step.

In this example, we assume that we have the inventory file defined in listing 3. Ansible playbook begins with `---` in line 1, which indicates the start of a document, and is normally executed in order from top to bottom. Then, we have two plays: one play is to be executed on `webservers` hosts (line 2) and the other play is to be executed on `dbservers` hosts (line 28),

where both `webservers` hosts and `dbservers` hosts are defined in the inventory file in listing 3).

The first play starting from line 2 is executed first. Then, in line 3-5, some variables are declared that can be used in the play. In our case here, these variables are `http_port` and `max_clients` that have values 80 and 200 respectively. These two variables are relevant for the tasks starting from line 7 in this play that are responsible for the configuration of the Apache servers. Then, in line 6, the remote user that is used to execute the tasks on the `webservers` managed nodes is defined. In our case here, it is the `root` user.

Then, from line 7, the tasks to be executed on the managed nodes are defined, and these tasks are executed in a top-to-bottom order. The first task is defined in lines 8-11, where the `name` key in line 8 describes what this tasks is about in natural language. Then, in line 9, the module executed by this task is defined, which is the `yum` module which uses the "yum" package management utility. In line 10, the name of the desired package to be installed by the `yum` module is defined and, in line 11, the desired version of this package. In our case, the name of the desired package is "httpd" which is the Apache server, and the desired version is the "latest" version.

As you probably have noticed, the file style is declarative in the sense that we declare what the desired state of the managed nodes should be for each task so that if the playbook were to be executed multiple times and the managed nodes are already in the desired state, no changes will be made in the nodes. For example, in our case here, after successfully executing the first task in the playbook, the latest version httpd Apache server will be already installed on the managed nodes. If we run the playbook again later and no newer version of the httpd server was available, Ansible will observe that and will not change the managed node state when running the first task again.

The second task, in lines 12-17, uses the `template` module which simply copies the Apache configuration file from the local control node to the remote managed nodes. However, before transferring this file to the remote node, some placeholders in the local file will be replaced with actual values by Ansible, and then the file with the actual values will be sent to the remote node. This is handy when only some variables need to be changed in configuration files for example.

However, in line 16-17, we see the keyword `notify`, which will trigger the handler (which is simply a task) named `restart apache` only if the current task changes the state of a managed node. This handler (or task) is defined in lines 23-26 under `handlers` in line 22. In our case here, this means that if the Apache configuration file is changed, the Apache server will be restarted. However, in our case here, the triggered handlers are actually executed after all the tasks have been executed.

Now, after installing Apache in the first task, and transferring the Apache configuration file in the second task, we ensure that Apache server is started in the third task in lines 18-21.

Now that all the tasks associated with the `webservers` hosts have already been executed, the execution proceeds with the second play in lines 28-38 that will execute tasks on `dbservers` hosts as the `root` remote user. There are two tasks that will ensure that the latest version of "postgresql" database system is installed and that it is then started after installing it successfully.

```yaml
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service:
      name: httpd
      state: started
  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted

- hosts: dbservers
  remote_user: root
  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

Listing 4: Playbook example YAML file. This example is adapted from [17].

4 Puppet VS Ansible

In addition to the going through Ansible and Puppet in more detail in the previous sections, here is an overview comparison of the mentioned points together with some notable points to mention.

4.1 Comparison: Ansible vs Puppet

- Ansible takes less time to install since it does not need to be installed on all the clients (managed nodes in Ansible's terms) whereas Puppet needs to install Puppet on all masters and all agents (clients).
- Ansible uses "Push" (by default) or "Pull" architecture while Puppet uses "Pull" architecture.
- Moreover, Ansible uses YAML syntax whereas Puppet uses an in-house developed declarative Domain Specific Language (DSL) that has similarities with Ruby[23]. YAML is a human-readable data-serialization language commonly used in configuration files [24].
- Both Ansible and Puppet handle clients in an idempotent way [17, 5].
- Regarding the scheduling, the Puppet agents generally check the configurations with the master (and that they are in the desired state) periodically (every 30 minutes) while that is not available in the free edition of Ansible [25].
- Ansible Galaxy and Puppet Forge are hubs for sharing content for Ansible and Puppet respectively [25].
- Ansible has an work-in-progress GUI while Puppet has a more mature one [23].

4.2 Comparison as a Table

	Ansible	**Puppet**
Written in	Python, PowerShell, Shell, Ruby [26]	C++, Clojure from 4.0, Ruby[5]
Language	YAML (Declarative)	In-house DSL (Declarative)
Architecture	Push (by default) and Pull if desired	Pull
Idempotent	Yes	Yes
Public Hubs	Ansible Galaxy	Puppet Forge
GUI	Work-in-progress	More mature

5 Conclusion

IaC tools are important for DevOps, since they automate system configuration and help ensure consistency across all nodes in a system. They can also increase the collaboration between the development team and the operation team.

In this essay, we have covered two widely used IaC tools: Ansible and Puppet, and gave a comparison between their features. Both Ansible and Puppet have their own advantages and disadvantages, and depending on the trade-offs that are desired, one tool can be preferred more than the other.

References

[1] Wikipedia contributors. *Infrastructure as code — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Infrastructure_as_code&oldid=943989495. [Online; accessed 25-April-2020]. 2020.

[2] Wikipedia contributors. *Configuration management — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-April-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Configuration_management&oldid=948991534.

[3] *DevOps in a Scaling Environment*. https://medium.com/tech-tajawal/devops-in-a-scaling-environment-9d5416ecb928. [Online; accessed 30-April-2020]. 2020.

[4] Wikipedia contributors. *DevOps — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=DevOps&oldid=952709409. [Online; accessed 30-April-2020]. 2020.

[5] Wikipedia contributors. *Puppet (company) — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-April-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Puppet_(company)&oldid=951958575.

[6] Luke Kanies. "Puppet". In: *The Architecture of Open Source Applications*. Ed. by Amy Brown and Greg Wilson. [Online accessed; 30-April-2020; From https://www.aosabook.org/en/puppet.html]. 2012.

[7] *Static catalogs*. https://puppet.com/docs/puppet/latest/static_catalogs.html. [Online; accessed 30-April-2020].

[8] *Catalog compilation*. https://puppet.com/docs/puppet/latest/subsystem_catalog_compilation.html. [Online; accessed 30-April-2020].

[9] *Certificate authority and SSL*. https://puppet.com/docs/puppet/latest/ssl_certificates.html. [Online; accessed 30-April-2020].

[10] Anicas Mitchell. *Getting Started With Puppet Code: Manifests and Modules*. https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules. [Online; accessed 30-April-2020].

[11] *Ansible architecture*. https://www.educba.com/ansible-architecture/. [Online; accessed 26-April-2020].

[12] *About Ansible*. https://docs.ansible.com/ansible/latest/index.html. [Online; accessed 26-April-2020]. 2020.

[13] *Installing Ansible*. https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html. [Online; accessed 26-April-2020]. 2020.

[14] *Connection plugins*. https://docs.ansible.com/ansible/latest/plugins/connection.html. [Online; accessed 26-April-2020]. 2020.

[15] *Ansible concepts*. https://docs.ansible.com/ansible/latest/user_guide/basic_concepts.html\#playbooks. [Online; accessed 26-April-2020]. 2020.

[16] *Configuration Management Tools: Ansible vs Puppet*. https://blog.cloudthat.com/configuration-management-tools-ansible-vs-puppet/. [Online; accessed 26-April-2020]. 2017.

[17] *Intro to Playbooks*. https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html\#about-playbooks. [Online; accessed 26-April-2020]. 2020.

[18] *Ansible architecture*. https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html. [Online; accessed 26-April-2020]. 2020.

[19] *How to build your inventory*. https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html\#intro-inventory. [Online; accessed 26-April-2020]. 2020.

[20] *Overview - How Ansible Works*. https://www.ansible.com/overview/how-ansible-works. [Online; accessed 26-April-2020].

[21] *Using and Developing Module Utilities*. https://docs.ansible.com/ansible/latest/dev_guide/developing_module_utilities.html\#developing-module-utilities. [Online; accessed 26-April-2020]. 2020.

[22] *Check Mode ("Dry Run")*. https://docs.ansible.com/ansible/latest/user_guide/playbooks_checkmode.html. [Online; accessed 26-April-2020]. 2020.

[23] *Ansible vs Puppet*. https://www.upguard.com/articles/ansible-puppet. [Online; accessed 29-April-2020]. Jan. 2020.

[24] Wikipedia contributors. *YAML — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=YAML&oldid=949886621. [Online; accessed 30-April-2020]. 2020.

[25] Shivam Arora. *Ansible vs. Puppet: The Key Differences to Know*. https://www.simplilearn.com/ansible-vs-puppet-the-key-differences-to-know-article. [Online; accessed 29-April-2020]. Apr. 2020.

[26] Wikipedia contributors. *Ansible (software) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Ansible_(software)&oldid=952449964. [Online; accessed 26-April-2020]. 2020.