# Chapter 1 - Getting Started with Ansible

## Ansible and Infrastructure Management

### On snowflakes and shell scripts

Many developers and system administrators manage servers by logging into them via SSH, making changes, and logging off. Some of these changes would be documented, some would not. If an admin needed to make the same change to many servers (for example, changing one value in a config file), the admin would manually log into *each* server and repeatedly make this change.

If there were only one or two changes in the course of a server's lifetime, and if the server were extremely simple (running only one process, with one configuration, and a very simple firewall), *and* if every change were thoroughly documented, this process wouldn't be a problem.

But for almost every company in existence, servers are more complex—most run tens, sometimes hundreds of different applications or application containers. Most servers have complicated firewalls and dozens of tweaked configuration files. And even with change documentation, the manual process usually results in some servers or some steps being forgotten.

If the admins at these companies wanted to set up a new server *exactly* like one that is currently running, they would need to spend a good deal of time going through all of the installed packages, documenting configurations, versions, and settings; and they would spend a lot of unnecessary time manually reinstalling, updating, and tweaking everything to get the new server to run close to how the old server did.

Some admins may use shell scripts to try to reach some level of sanity, but I've yet to see a complex shell script that handles all edge cases correctly while synchronizing multiple servers' configuration and deploying new code.

## Configuration management

Lucky for you, there are tools to help you avoid having these *snowflake servers*—servers that are uniquely configured and impossible to recreate from scratch because they were hand-configured without documentation. Tools like CFEngine[21], Puppet[22] and Chef[23] became very popular in the mid-to-late 2000s.

But there's a reason why many developers and sysadmins stick to shell scripting and command-line configuration: it's simple and easy-to-use, and they've had years of experience using bash and command-line tools. Why throw all that out the window and learn a new configuration language and methodology?

Enter Ansible. Ansible was built (and continues to be improved) by developers and sysadmins who know the command line—and want to make a tool that helps them manage their servers exactly the same as they have in the past, but in a repeatable and centrally managed way. Ansible also has other tricks up its sleeve, making it a true Swiss Army knife for people involved in DevOps (not just the operations side).

One of Ansible's greatest strengths is its ability to run regular shell commands verbatim, so you can take existing scripts and commands and work on converting them into idempotent playbooks as time allows. For someone (like me) who was comfortable with the command line, but never became proficient in more complicated tools like Puppet or Chef (which both required at least a *slight* understanding of Ruby and/or a custom language just to get started), Ansible was a breath of fresh air.

Ansible works by pushing changes out to all your servers (by default), and requires no extra software to be installed on your servers (thus no extra memory footprint, and no extra daemon to manage), unlike most other configuration management tools.

[21] http://cfengine.com/
[22] http://puppetlabs.com/
[23] http://www.getchef.com/chef/

**Idempotence** is the ability to run an operation which produces the same result whether run once or multiple times (source[24]).

An important feature of a configuration management tool is its ability to ensure the same configuration is maintained whether you run it once or a thousand times. Many shell scripts have unintended consequences if run more than once, but Ansible deploys the same configuration to a server over and over again without making any changes after the first deployment.

In fact, almost every aspect of Ansible modules and commands is idempotent, and for those that aren't, Ansible allows you to define when the given command should be run, and what constitutes a changed or failed command, so you can easily maintain an idempotent configuration on all your servers.

# Installing Ansible

Ansible's only real dependency is Python. Once Python is installed, the simplest way to get Ansible running is to use `pip`, a simple package manager for Python.

**If you're on a Mac**, installing Ansible is a piece of cake:

1. Check if `pip` is installed (`which pip`). If not, install it: `sudo easy_install pip`
2. Install Ansible: `sudo pip install ansible`

You could also install Ansible via Homebrew[25] with `brew install ansible`. Either way (`pip` or `brew`) is fine, but make sure you update Ansible using the same system with which it was installed!

**If you're running Windows** (i.e. you work for a large company that forces you to use Windows), it will take a little extra work to set everything up. There are two ways you can go about using Ansible if you use Windows:

1. The easiest solution would be to use a Linux virtual machine (with something like VirtualBox) to do your work, or to work within the Windows Subsystem for

[24] http://en.wikipedia.org/wiki/Idempotence#Computer_science_meaning
[25] http://brew.sh/

Linux. For detailed instructions, see Appendix A - Using Ansible on Windows workstations.

2. Ansible runs (somewhat) within an appropriately-configured Cygwin[26] environment. For setup instructions, please see my blog post Running Ansible within Windows[27]), and note that *running Ansible directly within Windows is unsupported and prone to breaking.*

**If you're running Linux**, chances are you already have Ansible's dependencies installed, but we'll cover the most common installation methods.

If you have `python-pip` and `python-devel` (`python-dev` on Debian/Ubuntu) installed, use `pip` to install Ansible (this assumes you also have the 'Development Tools' package installed, so you have `gcc`, `make`, etc. available):

```
$ sudo pip install ansible
```

Using pip allows you to upgrade Ansible with `pip install --upgrade ansible`.

*Fedora/Red Hat Enterprise Linux/CentOS:*

The easiest way to install Ansible on a Fedora-like system is to use the official yum package. If you're running Red Hat Enterprise Linux (RHEL) or CentOS, you need to install EPEL's RPM before you install Ansible (see the info section below for instructions):

```
$ yum -y install ansible
```

[26] http://cygwin.com/
[27] https://servercheck.in/blog/running-ansible-within-windows

On RHEL/CentOS systems, `python-pip` and `ansible` are available via the EPEL repository[28]. If you run the command `yum repolist | grep epel` (to see if the EPEL repo is already available) and there are no results, you need to install it with the following commands:

```
# If you're on RHEL/CentOS 6:
$ rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/\
epel-release-6-8.noarch.rpm
# If you're on RHEL/CentOS 7:
$ yum install epel-release
```

*Debian/Ubuntu:*

The easiest way to install Ansible on a Debian or Ubuntu system is to use the official apt package.

```
$ sudo apt-add-repository -y ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install -y ansible
```

If you get an error like "sudo: add-apt-repository: command not found", you're probably missing the `python-software-properties` package. Install it with the command:

```
$ sudo apt-get install python-software-properties
```

**Once Ansible is installed**, make sure it's working properly by entering `ansible --version` on the command line. You should see the currently-installed version:

[28] https://fedoraproject.org/wiki/EPEL

```
$ ansible --version
ansible 2.9.5
```

# Creating a basic inventory file

Ansible uses an inventory file (basically, a list of servers) to communicate with your servers. Like a hosts file (at `/etc/hosts`) that matches IP addresses to domain names, an Ansible inventory file matches servers (IP addresses or domain names) to groups. Inventory files can do a lot more, but for now, we'll just create a simple file with one server. Create a file at `/etc/ansible/hosts` (the default location for Ansible's inventory file), and add one server to it:

```
$ sudo mkdir /etc/ansible
$ sudo touch /etc/ansible/hosts
```

Edit this hosts file with nano, vim, or whatever editor you'd like, but note you'll need to edit it with sudo as root. Put the following into the file:

```
1 [example]
2 www.example.com
```

...where `example` is the group of servers you're managing and `www.example.com` is the domain name (or IP address) of a server in that group. If you're not using port 22 for SSH on this server, you will need to add it to the address, like `www.example.com:2222`, since Ansible defaults to port 22 and won't get this value from your ssh config file.

This first example assumes you have a server set up that you can test with; if you don't already have a spare server somewhere that you can connect to, you might want to create a small VM using DigitalOcean, Amazon Web Services, Linode, or some other service that bills by the hour. That way you have a full server environment to work with when learning Ansible—and when you're finished testing, delete the server and you'll only be billed a few pennies!

Replace the `www.example.com` in the above example with the name or IP address of your server.

# Running your first Ad-Hoc Ansible command

Now that you've installed Ansible and created an inventory file, it's time to run a command to see if everything works! Enter the following in the terminal (we'll do something safe so it doesn't make any changes on the server):

```
$ ansible example -m ping -u [username]
```

...where `[username]` is the user you use to log into the server. If everything worked, you should see a message that shows `www.example.com | success >>`, then the result of your ping. If it didn't work, run the command again with `-vvvv` on the end to see verbose output. Chances are you don't have SSH keys configured properly—if you login with `ssh username@www.example.com` and that works, the above Ansible command should work, too.

Ansible assumes you're using passwordless (key-based) login for SSH (e.g. you login by entering `ssh username@example.com` and don't have to type a password). If you're still logging into your remote servers with a username and password, or if you need a primer on Linux remote authentication and security best practices, please read Chapter 10 - Server Security and Ansible. If you insist on using passwords, add the `--ask-pass` (`-k`) flag to Ansible commands (you may also need to install the `sshpass` package for this to work). This entire book is written assuming passwordless authentication, so you'll need to keep this in mind every time you run a command or playbook.

Need a primer on SSH key-based authentication? Please read through Ubuntu's community documentation on SSH/OpenSSH/Keys[29].

Let's run a more useful command:

[29] https://help.ubuntu.com/community/SSH/OpenSSH/Keys

```
$ ansible example -a "free -h" -u [username]
```

In this example, we quickly see memory usage (in a human-readable format) on all the servers (for now, just one) in the `example` group. Commands like this are helpful for quickly finding a server that has a value out of a normal range. I often use commands like `free -m` (to see memory statistics), `df -h` (to see disk usage statistics), and the like to make sure none of my servers is behaving erratically. While it's good to track these details in an external tool like Nagios[30], Munin[31], or Cacti[32], it's also nice to check these stats on all your servers with one simple command and one terminal window!

# Summary

That's it! You've just learned about configuration management and Ansible, installed it, told it about your server, and ran a couple commands on that server through Ansible. If you're not impressed yet, that's okay—you've only seen the *tip* of the iceberg.

```
 ________________________________________
/ A doctor can bury his mistakes but an \
| architect can only advise his clients |
\ to plant vines. (Frank Lloyd Wright)  /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

[30] http://www.nagios.org/
[31] http://munin-monitoring.org/
[32] http://www.cacti.net/