



Automated Computer Management with Ansible

Beau Harrison

5 November 2020

What is Ansible?

- Simple and easy IT automation
 - Push model via OpenSSH, agentless
- Provisioning
 - PXE booting and kickstarting bare-metal servers or VMs, or creating virtual or cloud instances from templates
- Configuration management
 - Idempotent server state definitions
- Application deployment
 - One-command standard deployments to update applications across many machines
- Continuous delivery
 - Require sequential success of multiple processes
- Security automation
 - Automate and standardize threat-scanning and firewall updates
- Orchestration
 - Define how multiple configurations interact and ensure the disparate pieces can be managed as a whole

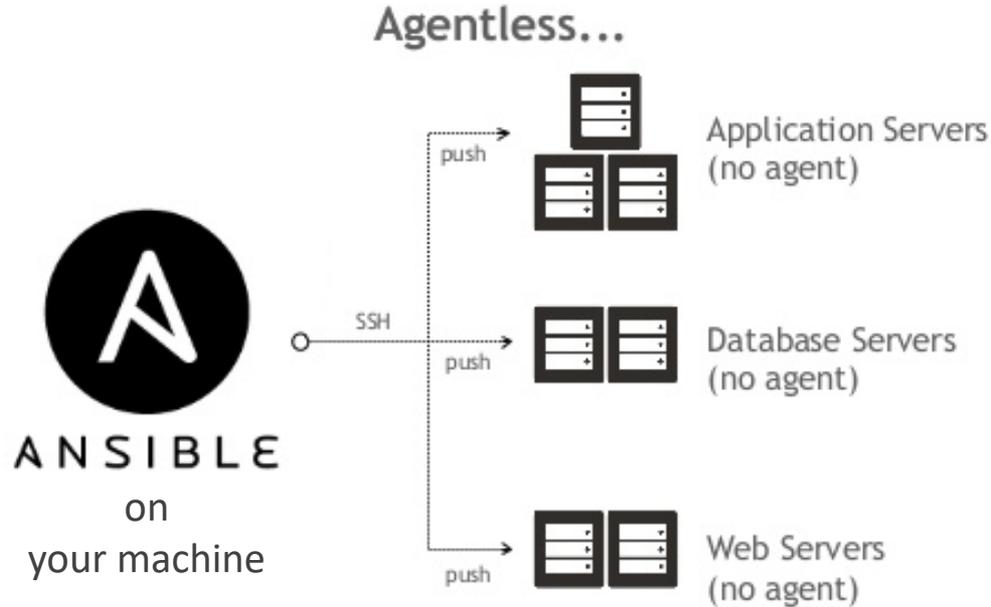
<https://docs.ansible.com/ansible/latest/index.html>

How does Ansible work?

- Agentless
 - Tasks are executed from your machine
- Automated
 - Reliable and less error prone
- Playbooks
 - Configuration/Installation/Deployment in a single YAML file, doubles as notes
- Inventories
 - Tasks can be executed on groups of machines
- Module
 - Atomic task e.g. Copy file
- Task
 - Uses a module with arguments
- Play
 - A series of tasks with designated hosts and user
- Playbook
 - A series of plays

<https://www.ansible.com/overview/how-ansible-works>

How does Ansible work? – Agentless



You don't have to install something extra onto the remote hosts you want to manage.

<https://www.slideshare.net/DonghuKIM2/ansible-with-oci-221441463>

<https://www.ansible.com/overview/how-ansible-works>

How does Ansible work? – Modules

```
> ansible outland.fnal.gov -m ping
```

ansible – command line tool

outland.fnal.gov – remote hostname

-m – module flag

ping – built-in Ansible module that does exactly what it says

Module examples

- copy – copy files from control node to target
- user – manage users and passwords
- package - install, update, remove tools using target package manager
- service – manage target system services using target init system
- firewalld – manage firewall configuration
- file – set permissions and ownership
- lineinfile – manage single lines on existing files
- command – allows for arbitrary commands, best practice is to avoid this

https://docs.ansible.com/ansible/latest/user_guide/modules_intro.html

How does Ansible work? – Tasks

Tasks are modules with arguments.

Ansible allows users to create predefined series of tasks in YAML files.

```
---
tasks:
  - name: create directory for nginx
    file:
      path: /path/to/nginx/dir
      state: directory

  - name: install nginx latest version
    yum:
      name: nginx
      state: latest

  - name: start nginx
    service:
      name: nginx
      state: started
```

https://docs.ansible.com/ansible/latest/user_guide/index.html#writing-tasks-plays-and-playbooks

How does Ansible work? – Plays

Plays are a series of tasks with a designated hosts and user.

Ansible allows users to create predefined plays in YAML files.

```
---
- hosts: webservers
  become: yes
  become_method: ksu
  become_user: root

tasks:
  - name: create directory for nginx
    file:
      path: /path/to/nginx/dir
      state: directory

  - name: install nginx latest version
    yum:
      name: nginx
      state: latest

  - name: start nginx
    service:
      name: nginx
      state: started
```

https://docs.ansible.com/ansible/latest/user_guide/index.html#writing-tasks-plays-and-playbooks

How does Ansible work? – Variables

Variables can help prevent typos and make updates simple.

It's clear that if we can substitute the variables this play is a generic solution for installing and starting a service.

```
---
- hosts: webservers
  become: yes
  become_method: ksu
  become_user: root
  vars:
    app_name: nginx
    app_path: /path/to/nginx/dir

  tasks:
    - name: create directory for {{ app_name }}
      file:
        path: {{ app_path }}
        state: directory

    - name: install {{ app_name }} latest version
      yum:
        name: {{ app_name }}
        state: latest

    - name: start {{ app_name }}
      service:
        name: {{ app_name }}
        state: started
```

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

How does Ansible work? – Playbook

Playbooks define how, in which order, on which machines, and what modules should be executed.

Now we are orchestrating module execution!

```
---
- name: Install and start the nginx service
- hosts: webservers
  become: yes
  become_method: ksu
  become_user: root
  vars:
    app_name: nginx
    app_path: /path/to/nginx/dir

  tasks:
    - name: create directory for {{ app_name }}
      file:
        path: {{ app_path }}
        state: directory

    - name: install {{ app_name }} latest version
      yum:
        name: {{ app_name }}
        state: latest

    - name: start {{ app_name }}
      service:
        name: {{ app_name }}
        state: started

- name: Create database table and set user
- hosts: databases
  become: yes
  become_method: ksu
  become_user: root
  vars:
    table_name: test_table
    table_owner: someuser

  tasks:
    - name: Create {{ table_name }} with several columns in ssd
      postgresql_table:
        name: {{ table_name }}
        columns:
          - id bigserial primary key
          - num bigint
          - stories text
        tablespace: ssd

    - name: Set owner to {{ table_owner }}
      postgresql_table:
        name: {{ table_name }}
        owner: {{ table_owner }}
```

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

How does Ansible work? – Playbook

Playbooks define how, in which order, on which machines, and what modules should be executed.

Now we are orchestrating module execution!

What is this magic!?

”webservers” and ”databases” are inventories of remote hosts.

```
---
- name: Install and start the nginx service
- hosts: webservers
  become: yes
  become_method: ksu
  become_user: root
  vars:
    app_name: nginx
    app_path: /path/to/nginx/dir

  tasks:
    - name: create directory for {{ app_name }}
      file:
        path: {{ app_path }}
        state: directory

    - name: install {{ app_name }} latest version
      yum:
        name: {{ app_name }}
        state: latest

    - name: start {{ app_name }}
      service:
        name: {{ app_name }}
        state: started

- name: Create database table and set user
- hosts: databases
  become: yes
  become_method: ksu
  become_user: root
  vars:
    table_name: test_table
    table_owner: someuser

  tasks:
    - name: Create {{ table_name }} with several columns in ssd
      postgresql_table:
        name: {{ table_name }}
        columns:
          - id bigserial primary key
          - num bigint
          - stories text
        tablespace: ssd

    - name: Set owner to {{ table_owner }}
      postgresql_table:
        name: {{ table_name }}
        owner: {{ table_owner }}
```

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

How does Ansible work? – Inventory

- Groups allow for aliasing lists of targets
- Hosts can be in multiple groups
- Hosts with a common naming scheme can be added using the range syntax
 - This syntax also allows for a stride e.g. `clx[01:80:2]` odd CLXs
- Ansible command line allows for pattern matching and sub-selection from inventory
 - Multiple hosts/groups – `dses:dpes`
 - Exclude – `daes:!dpms`
 - Intersection – `dces:&dpms_test`

> `ansible-playbook test-playbook.yaml --limit 'all:!clxs_test:!dpms_test'`

```
---
all:
  children:
    clxs_mcr:
      hosts:
        clx[1:14]:
    clxs_test:
      hosts:
        clx39:
    clxs:
      children:
        clxs_mcr:
        clxs_test:
      hosts:
        clx[15:80]:
    dses:
      hosts:
        dse[01:10]:
    dpes:
      hosts:
        dpe[01:10]:
    dues:
      hosts:
        due[01:47]:
    dces:
      hosts:
        dce[01:53]:
    dpms:
      hosts:
        dce[01:08]:
      children:
        dpms_test:
          hosts:
            dce46:
    daes:
      children:
        dses:
        dpes:
        dues:
        dces:
```

https://docs.ansible.com/ansible/latest/network/getting_started/first_inventory.html

How does Ansible work? – Ad-hoc commands

“Playbooks seem cumbersome for simple tasks”

Ad-hoc commands allow basic commands using Ansible inventories.

```
> ansible daes -a “/sbin/shutdown”
```

Modules can be used in conjunction with ad-hoc commands.

```
> ansible daes -m user -a “name=beau state=absent”
```

https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html

Ansible Galaxy

Hosted shared Ansible content at <https://galaxy.ansible.com/>

Can be a useful reference to see how common problems are solved.

Beau's imagined use cases

- Deploy applications and configurations to production and backup nodes
 - Doing this with the Interlocks servers
- Upgrade OS
 - I'm going to attempt to use Ansible to aid in upgrading Interlocks server from SLF6 to SLF7
- Gather statistics
 - Replace Bash and Python scripts in determining the state of many machines
- Manage permissions
 - New users could be added to disparate systems using a common Controls inventory
- Last leg of continuous deployment
 - Simply automate fetching, testing, and uploading code
- Quick security updates
 - New firewall settings or application updates can be deployed to all systems with one command
- Rolling updates
 - For systems with a load balancer or auto discovery, updates can be rolled out with interrupting service
- Coordinated deployments
 - Interdependent services can be coordinated to deploy simultaneously

Potential Future Topics

Let me know if you want to know more.

- Ansible Tower
 - Web-based dashboard for managing and executing tasks and playbooks
- Specific strategies
 - Provisioning
 - Configuration management
 - Application deployment
 - Continuous delivery
 - Security automation
 - Orchestration
- Roles
 - automatically load related vars_files, tasks, handlers, and other Ansible artifacts based on a known file structure

Questions?