Automating with **ansible** (part A)

Iman darabi

https://www.linkedin.com/in/imandarabi/

Table of contents (part A - Basics)

Iman Darabi

https://www.linkedin.com/in/imandarabi/

Table of contents
(part B - Advanced)

Iman Darabi

https://www.linkedin.com/in/imandarabi/

1_Introduction

Iman darabi

https://www.linkedin.com/in/imandarabi/

1.a) Definitions

- Simple automation language that can perfectly describe an IT application infrastructure in ansible playbooks
- Open-source software provisioning, configuration management, and application-deployment tool (wikipedia).
- Initial release: February 20, 2012
- Repository: github.com/ansible/ansible
- Ansible, an open source community project sponsored by Red Hat,
- Other configuration management tools:
  - Puppet
  - Chef
  - Juju
  - Salt ...

1.b) Why ansible

- Minimal learning required
  - Human readable automation
  - No special coding skills needed
  - Tasks executed in order
- Only OpenSSH and Python are required on the managed nodes
- Agent-less (no exploit or update)
- Powerful
  - App deployment
  - Configuration management
  - Workflow orchestration
  -

1.b_Understanding **YAML**

Iman darabi

https://www.linkedin.com/in/imandarabi/

What is YAML?

YAML **A**in’t **M**arkup **L**anguage

https://yaml.org

```
%YAML 1.2
---
YAML: YAML Ain't Markup Language

What It Is: YAML is a human friendly data serialization
  standard for all programming languages.
```

- YAML is a human-readable data serialization standard that can be used in conjunction with all programming languages and is often used to write configuration files.
- YAML has no executable commands. It is simply a data-representation language.
- It is commonly used for configuration files and in applications where data is being stored or transmitted.
- YAML files should end in .yaml/.yml
- YAML is case sensitive.
- YAML does not allow the use of tabs. Spaces are used instead as tabs are not universally supported.
- We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON

YAML Data presentation syntax

YAML style comparison

XML

```
<Servers>
  <Server>
    <name>SERVER1</name>
    <owner>iman</owner>
    <created>1399</created>
  </Server>
 </Servers>
```

JSON

```
{
  Servers: [
    {
    name: Server1,
    owner: iman,
    created: 1399,
    status: active,
    }
  ]
}
```

YAML

```
Servers:
  -     Name: Server1
        owner: iman
        created: 1399
        status: active
```

Document separator

--- <- Document header

... <- Document terminator

Data type - key value pair

- Key value pairs are defined using a colon (:) and a space (▮)

Fruit: Apple

Vegetable: Carrot

Liquid: Water

Meet: Chiken

Data type - Array / Lists

- Dash “-” indicates elements of lists

```
# Block Format
Fruites:

  -   Orange
  -   Apple


# Inline Format
Fruits:  [ Orange, Apple ]
```

Data type - Dictionary / Map

- There must be equal number of blank space ( ) for each of properties

Banana:

Calories: 105

Fat: 0.4

Grapes:

Calories: 62

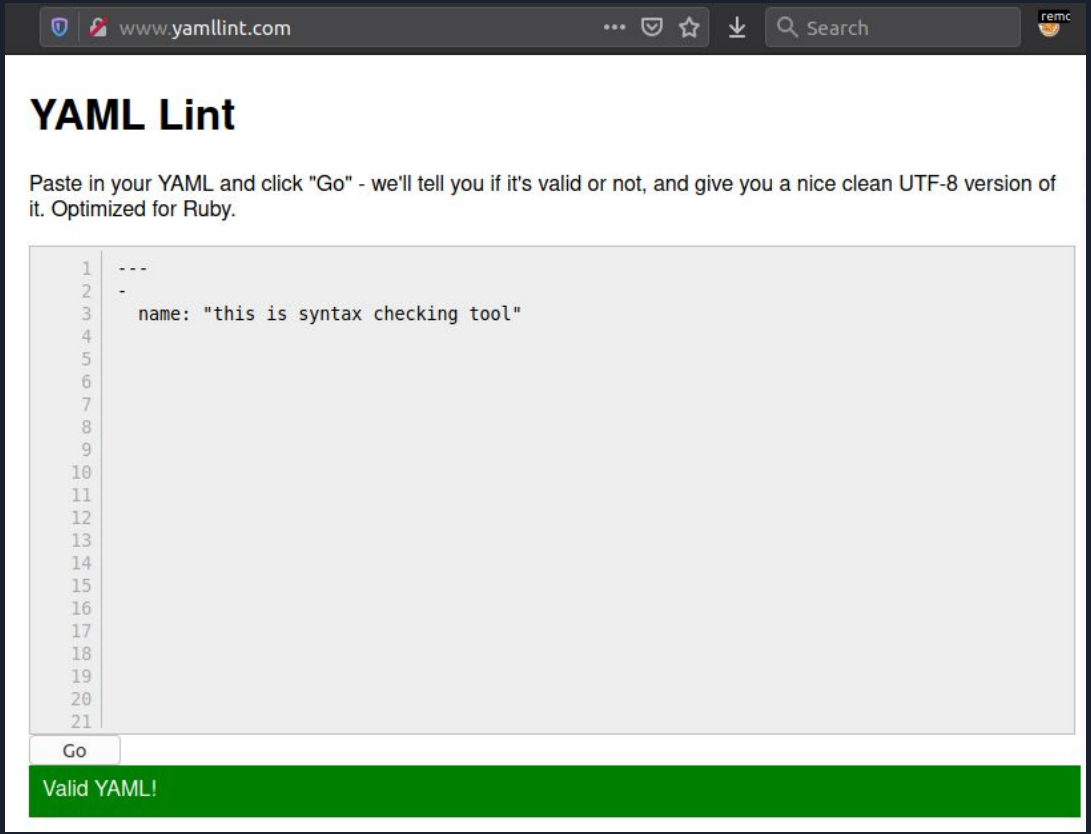
Fat: 0.3

Data type - Strings

```
data: |
    Each of these
    Newlines
    Will be broken up
```

```
data: >
    This text is
    wrapped and will
    be formed into
    a single paragraph
```

Check syntax
with
http://yamllint.com



2_Basic getting started

Iman darabi

https://www.linkedin.com/in/imandarabi/

2.a) Setting up ansible

Iman darabi

https://www.linkedin.com/in/imandarabi/

Setting up controller and remote hosts

1. Install Ansible software
    a. Only the control node needs ansible
        i. # apt install ansible
    b. Managed hosts need Python and SSH
2. Create a non-root user and perform all tasks as this user
3. Set up SSH for communications
4. Configure sudo
    a. user ALL=(ALL) NOPASSWD: ALL

2.b) Managing configuration and Inventory

Iman darabi

https://www.linkedin.com/in/imandarabi/

configuration file

- ansible.cfg defines ansible configurations
- It is common work to work with project directories that contain these files
- Search order of changes in configuration files are as follow:
    - ANSIBLE_CONFIG (environment variable if set)
    - ansible.cfg (in the current directory)
    - ~/.ansible.cfg (in the home directory)
    - /etc/ansible/ansible.cfg
- Create Base directory and add ansible.cfg & inventory files as follows
- Change directory to the Base dir and run the following command to check it all works:
    - $ ansible all --list-hosts

Inventory file

- Inventory contains a list of hostnames or IP addresses
- Dynamic inventory is discussed later
- Default inventory file: /etc/ansible/hosts
-

Inventory file sample

INI format

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

YAML format

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Base directory

ansible.cfg

```
[defaults]
remote_user = ansible
host_key_checking = false
inventory = inventory

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = False
```

inventory

```
[all]
node1.ansible.local
node2.ansible.local
```

2.c) Ad-hoc Commands

Iman darabi

https://www.linkedin.com/in/imandarabi/

Ad-hoc Commands Summarized

- Ansible modules can be called using the ansible command
- Use **ansible-doc -l** for a list of modules, and **ansible-doc modulename** for information
- Specify which module to use, using -m
- The **command** module is default, and does not have to be specified
    - ansible all ping
- about module options
    - Ansible <category_inventory> -m <module_name> -a <module_arg>
    - Ansible all -m command -a id
    - Ansible all -m shell -a env

2.d) working with modules

Iman darabi

https://www.linkedin.com/in/imandarabi/

modules

- Modules are reusable, standalone scripts that can be used by the Ansible API, the **ansible** command, or the **ansible-playbook** command.
- Modules provide a defined interface, accepting arguments and returning information to Ansible by printing a JSON string to stdout before exiting.

Module usage

Ad-hoc command

```
# ansible all -m apt -a "name=htop
state=present"
```

Playbook

```
hosts: all
tasks:
- name: install htop application
    apt:
    name: htop
    state: present
```

2.e) Understanding Playbooks

Iman darabi

https://www.linkedin.com/in/imandarabi/

playbooks

- A list of instruction describing the steps to bring your server to a certain configuration stat
- Playbook - A single YAML file
    - Play - Defines a set of activities (task) to be run on hosts
        - Task - An action to be performed on the host
            - Execute a command
            - Run a script
            - Install a package
            - Shutdown/Restart

Playbook format

```
# simple  Ansible Playbook.yml
---
-
  name: Play 1
  hosts: localhost
  tasks:
    - name: Execute command 'date'
      command: date
    - name: Execute script on server
      script: test_script.sh

-
  name: Play 2
  hosts: localhost
  tasks:
    - name: Install web service
      yum:
        name: httpd
        state: present
```

Playbook keywords

```
# simple  Ansible Playbook.yml
---
-
  name: Play 1
  hosts: localhost
  tasks:
    - name: Execute command 'date'
      command: date
    - name: Execute script on server
      script: test_script.sh

-
  name: Play 2
  hosts: localhost
  tasks:
    - name: Install web service
      yum:
        name: httpd
        state: present
```

Playbook keywords

- name: Identifier. Can be used for documentation, in or tasks/handlers.
- hosts: A list of groups, hosts or host pattern that translates into a list of hosts that are the play's target.
- tasks: Main list of tasks to execute in the play, they run after roles and before post_tasks.
- vars, when, tags, until, async, become, delegate_to and …
  - https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

Playbooks VS shell scripts

```
---
- hosts: all
  tasks:
  -   name: Install Apache
      Command: yum install  httpd
  -  name: Copy configuration files.
    command: >
      Cp /<path>/httpd.conf /<path2>/httpd.conf
  -  name: start apache service
    command: systemctl start httpd
```

```
echo 'Install Apache'
yum install --quiet -y httpd
cp /<path>/httpd.conf  /<path2>/httpd.conf
echo  'start apache service'
Systemctl start httpd
```

Simple VS complex playbooks

Simple Ansible Playbook

- Run command1 on server1
- Run command2 on server2
- …
- Run command10 on server10
- Restarting server1
- Restarting server2
- …
- Restarting server10

Complex Ansible Playbook

- Deploy 50 VMs on Public Cloud
- Deploy 50 VMs on Private Cloud
- Provision Storage to all VMs
- Setup Cluster Configuration
- Install and Config Backup clients
- Update CMDB database
- …

2.f) variables, include, imports and facts

Iman darabi

https://www.linkedin.com/in/imandarabi/

Understanding Variables and Inclusions

- Variables are what makes ansible flexible: set a variable locally to have it behave differently on different managed hosts
    - Variables can be set through the inventory file, at the top of a playbook or using inclusions
- Ansible facts can be used like variables as well:
- Inclusions allow you to make ansible playbooks more modular
    - Instead of writing one big playbook, you can work with several small playbooks, where each playbook is used to focus on a specific set of tasks.

Including and Importing

- All import* statements are pre-processed at the time playbooks are parsed.
- All include* statements are processed as they are encountered during the execution of the playbook.

include: install-packages.yml

import_tasks: install-packages.yml

Facts

- As we've already seen, when Ansible runs a playbook, before the first task runs, this happens:
  - GATHERING FACTS **********
  - ok: [servername]
- When Ansible gather facts, it connects to the host and queries the host for all kind of details about the host: CPU arch, operating system, IP address, memory, disk info …
- This information is stored in variables that are called facts, and they behave just like any other variable does.

Facts - ad-hoc samples

- Display facts from all hosts and store them indexed by I(hostname) at C(/tmp/facts).
  - # ansible all -m setup --tree /tmp/facts

- Display only facts regarding memory found by ansible on all hosts and output them.
  - # ansible all -m setup -a 'filter=ansible_*_mb'

- Display only facts about certain interfaces.
  - # ansible all -m setup -a 'filter=ansible_eth[0-2]'

Special variables

- These variables cannot be set directly by the user; Ansible will always override them to reflect internal state.
- Inventory_hostname
  - The inventory name for the ‘current’ host being iterated over in the play
- Inventory_hostname_short
  - The short version of inventory_hostname
- ansible_distribution
  - Os distribution like “Debian” or “Ubuntu” or “CentOS”
- Reference:
  - https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html

2.g) Understanding Jinja2 Templates

Iman darabi

https://www.linkedin.com/in/imandarabi/

Jinja2 Templates

- Jinja2 templates are Python-based templates that are used to put host-specific data on hosts, using generic YAML and Jinja2 files
- Jinja2 templates are used to modify Yaml files before they are sent to the managed host
- Jinja2 can also be used to reference variables in playbooks
- As advanced usage, Jinja2 loops and conditionals can be used in templates to generate very specific code
- The host-specific data is generated through variables or facts

3) working with roles

Iman darabi

https://www.linkedin.com/in/imandarabi/

3.a) Understanding role structure

- Idea: if you design in a structured way, everybody knows where to find what !
- Ansible roles provide uniform ways to load tasks, handlers, and variables from external files
- A role typically corresponds to the type of service that is offered (web, database, etc.)
- The purpose is to keep the size of playbooks manageable
- Roles use a specific directory structure, with locations for default, handlers, tasks, templates, and variables

Role directory structure contents

- **defaults**: contains a main.yml with default values for variables
- **files**: static files that are referenced by role tasks
- **handlers**: contains a main.yml with handler definitions
- **meta**: contains a main.yaml with informations about the role, including author, license, platforms, and dependencies
- **tasks**: has a main.yml with task definitions
- **vars**: has a main.yml file with role variable definitions

Understanding role variables

- Role variables are defined in vars/main.yml
- These variables have a high priority and cannot be overridden by inventory variables
- Default variables can be defined in defaults/main.yml and have the lowest precedence
- Use default variables only if you intend to have the variable overridden somewhere else
    - Overriding variables is common as roles are used as templates, where specific variables may be overridden in specific playbooks

Using Roles

- Roles are easily used from playbooks:

```
---
- hosts: node2.ansible.local
  roles:
    - role1
    - role2
```

3.b) Creating roles

- Creating roles involves the following steps
  - Create the role structure
  - Define the role content
  - Use the role in a playbook
- Use the **ansible-galaxy init <role_name>** command to automate creating the role directory structure
-

3.c) Deploying roles with ansible galaxy

- Ansible Galaxy (http://galaxy.ansible.com) is the community resource for getting and publishing roles
- Many roles that are ready to use are offered for download
- Roles that are still in development can be followed by watchers
- Use ansible-galaxy command to install roles from galaxy.ansible.com
  - Ansible-galaxy install <rolename>

http://galaxy.ansible.com

https://galaxy.ansible.com

GALAXY

About Help Documentation Login

Home

Search

Community

Home

Most Popular

System
Development
Networking
Cloud
Database
Monitoring
Packaging
Playbook Bundles
Security
Web

Download

Jump-start your automation project with great content from the Ansible community. Galaxy provides pre-packaged units of work known to Ansible as roles.

Share

Help other Ansible users by sharing the awesome roles you create.

Maybe you have a role for installing and configuring a popular software package, or a role for deploying

Featured

Read the latest from The Inside Playbook, and keep up with what's happening in the Ansible universe.

3.d) Using the ansible galaxy CLI utility

- # Ansible-galaxy search
  - Search for roles
- Use options --author, --platforms and --galaxy-tags to narrow down search results
- # ansible-galaxy search “install mariadb” --platforms el
- # ansible-galaxy info
  - Provides information about roles
  - # ansible galaxy info f500.mariadb55
- # ansible-galaxy install <rolename>
  - Download a role and install it on the control node in ~/.ansible/roles

End of part A

Iman darabi

https://www.linkedin.com/in/imandarabi/