

Ansible and Firebird

Managing [Firebird](#) with [Ansible](#)



Author: Philippe Makowski IBPhoenix - R.Tech

Email: pmakowski@ibphoenix.com

Licence: Public Documentation License

Date: 2016-10-05

Part of these slides are from [Gülçin Yildirim](#) talk at Fosdem

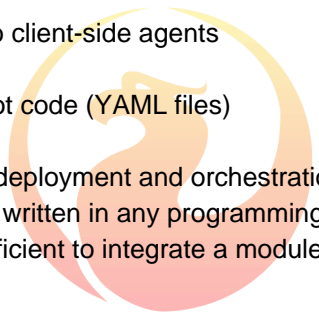
What is Ansible?

Simple, agentless and powerful open source IT automation tool

- Provisioning
- Configuration management
- Application Deployment
- Continuous Delivery
- Security & Compliance
- Orchestration



Why Ansible?

- Agent-less architecture (no agent is required, everything is done by using SSH, ssh for communication)
 - No centralized server, no client-side agents
 - SSH based
 - Configuration as data, not code (YAML files)
 - Batteries included
 - Full conf. management, deployment and orchestration
 - Custom modules can be written in any programming language.
 - JSON input/output is sufficient to integrate a module into Ansible.
- 

Building blocks

Typically with *Ansible*

we execute **tasks** for an **inventory**,
utilizing some **modules**,
using or populating some **variables**,
processing some file **templates**,
in a **playbook**,
which can be organized in **roles**.



Inventory

- Tells *Ansible* about hosts it should manage
 - Hostnames, IPs, ports, SSH parameters
 - Server specific variables
- 2 common ways to provide Ansible an inventory:
 - **Static inventory:** A flat INI file (e.g., *hosts.ini*)
 - **Dynamic inventory:** An application returning a JSON data (e.g.,: [ec2.py](#) for *Amazon EC2*)
- Hosts are grouped. They can belong to multiple groups
- Groups can also belong to multiple groups

Inventory

Below inventory file in **INI format** contains **3 hosts** under **2 groups**.

There is also a 3rd **group which contains other groups** and all of the hosts.

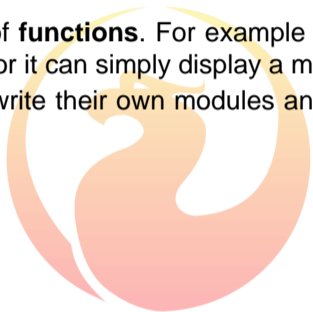
```
# "master" group with 1 host
[master]
firebird-master      ansible_ssh_host=10.0.0.5

# "standby" group with 2 hosts
[standbys]
firebird-standby-01  ansible_ssh_host=10.0.0.10
firebird-standby-02  ansible_ssh_host=10.0.0.11

# the "replication" group contains both "master" and "standbys" groups
[replication:children]
master
standbys
```

Module

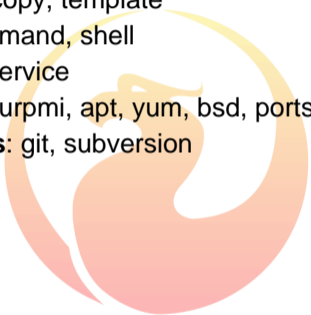
- Modules provide *Ansible* means to control or manage resources on local or remote servers.
- They perform a variety of **functions**. For example a module may be responsible for rebooting a machine or it can simply display a message on the screen.
- *Ansible* allows users to write their own modules and also provides out-of-the-box core or extras modules.
 - Core
 - Extras



Module

Some of the most commonly used modules are:

- **File handling:** file, stat, copy, template
- **Remote execution:** command, shell
- **Service management:** service
- **Package management:** urpmi, apt, yum, bsd, ports
- **Source control systems:** git, subversion



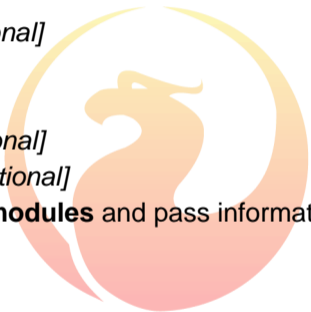
Task

Tasks are responsible for calling a **module** with a specific set of **parameters**.

Each Ansible task contains:

- a descriptive name *[optional]*
- a module to be called
- module parameters
- pre/post-conditions *[optional]*
- processing directives *[optional]*

They allow us to call *Ansible* **modules** and pass information to **consecutive tasks**.



Task

Below task invokes the `file` module by providing 4 parameters. It ensures 3 conditions are true:

- `/var/lib/firebird` exists as a directory
- owner of `/var/lib/firebird` is `firebird`
- group of `/var/lib/firebird` is `firebird`

If it doesn't exist, *Ansible* creates the directory and assigns owner & group. If only the owner is different, *Ansible* makes it `firebird`.

```
- name: Ensure the data folder has right ownership
  file: path="/var/lib/firebird" state=directory owner=firebird group=firebird
```

Task

Following example shows relationships between tasks. The first task checks if a device exists and the second task mounts the device depending on the result from the first task.

Please note "**register**" and "**when**" keywords.

```
- name: Check if the data volume partition exists
  stat: path=/dev/sdcl
  register: partition

- name: Ensure the Firebird data volume is mounted
  mount: src=/dev/sdcl name="/var/lib/firebird/data" fstype=ext4 state=mounted
  when: partition.stat.exists is defined and partition.stat.exists
```

Variable

Variables in *Ansible* are very useful for reusing information. Sources for variables are:

- **Inventory:** We can assign variables to hosts or groups (group vars, host vars).
- **YAML files:** We can include files containing variables.
- **Task results:** Result of a task can be assigned to a variable using the **register** keyword as shown in the previous slide.
- **Playbooks:** We can define variables in Ansible playbooks.
- **Command line:** (-e means extra variable // -e "uservar=philippe")

Variable

There is also discovered variables (`facts`) and they can be found by using `setup` module:

```
ansible -i hosts.ini -m setup -a 'filter=ansible_nodename'
```

All of the output in json: `bios_version`, `architecture`, `default_ipv4_address`, `ansible_os_family`, etc.

You can use variables in tasks, templates themselves and you can iterate over using `with_type` functions.

```
- name: Ensure Firebird users are present
  firebird_user:
    state: present
    name: "{{ item.name }}"
    password: "{{ item.password }}"
  with_items: firebird_users
```

Template

We can think templates as our configuration files. *Ansible* lets us use the [Jinja2 template engine](#) for reforming and parameterising our files.

The Jinja2 templating engine offers a wide range of control structures, functions and filters.

Some of useful capabilities of Jinja2:

- for-loops
- join(), default(), range(), format()
- union(), intersect(), difference()
- to_json, to_nice_yaml, from_json, from_yaml
- min, max, unique, version_compare, random

Template

Let's have a look at `firebird.conf.j2` file:

```
# {{ ansible_managed }}  
DatabaseAccess = {{ DatabaseAccess | default('/var/lib/firebird/data')}}  
ServerMode = {{ ServerMode | default('Super')}}
```



Playbook

A Playbook

```
---  
- name: install and start apache  
  hosts: webservers  
  user: root  
  
  tasks:  
  
    - name: install httpd  
      yum: name=httpd state=latest  
  
    - name: start httpd  
      service: name=httpd state=running
```



Page 23 of 53

Copyright (C) 2014 Ansible, Inc.

- **Playbooks** contains **Plays**
 - **Plays** contain **Tasks**
 - **Task** call **Modules** and may (*optionally*) trigger **handlers** (*run once, run at the end*)

Playbook

If *Ansible* **modules** are the **tools** in your workshop, **playbooks** are your **design plans**.

- *Ansible* playbooks are written using the **YAML** syntax.
- Playbooks may contain more than one **plays**
- Each play contains:
 - **name** of host groups to connect to
 - **tasks** it needs to perform.

Strict dependency ordering: everything in file performs in a sequential order. (Before v.2)

Playbook

Let's look at a playbook example:

```
---  
- name: Ensure all required Firebird dependencies ready  
  hosts: firebird-all # manage all Firebird servers  
  sudo: yes  
  sudo_user: root  
  vars_files:  
    - 'defaults/firebird.yml'  
  tasks:  
    - include: 'tasks/firebird.yml' # load Firebird setup tasks
```

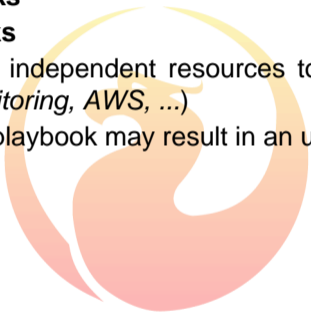
Role

In *Ansible*,

- **playbooks** organize **tasks**
- **roles** organize **playbooks**

Imagine that we have lots of independent resources to manage (*e.g.*, *web servers*, *Firebird servers*, *logging*, *monitoring*, *AWS*, ...)

Putting everything in a single playbook may result in an unmaintainable solution.



Role

Here a playbook using roles :

```
---  
# This playbook deploys the whole application stack on monitoring box.  
- hosts: monitoring  
  user: root  
  roles:  
    - base-apache  
    - munin  
    - ganglia-gmetad  
    - nagios
```

Role

Here you can see a role directory structure:

```
├── base-apache
│   └── tasks
│       └── main.yml
├── ganglia-gmetad
│   ├── files
│   │   ├── apache.conf
│   │   └── gmetad.conf
│   ├── handlers
│   │   └── main.yml
│   └── tasks
│       └── main.yml
├── munin
│   ├── files
│   │   ├── apache.conf
│   │   └── munin.conf
│   ├── handlers
│   │   └── main.yml
│   └── tasks
│       └── main.yml
└── nagios
    ├── files
    │   └── nagios.cfg
    ├── handlers
    │   └── main.yml
    ├── tasks
    │   └── main.yml
    └── templates
        ├── ansible-managed-commands.cfg.j2
        └── switches.cfg.j2
```

How to Invoke Ansible?

To work with Ansible, we have 2 main alternatives;

1. Running ad-hoc commands
2. Running playbooks



Ad-hoc Commands

We can call any Ansible module from the command line, anytime.

The **ansible** CLI tool works like a single task. It requires an inventory, a module name, and module parameters.

For example, given an inventory file like:

```
[dbservers]  
db.example.com
```

Now we can call any module.

Ad-hoc Commands

We can check uptimes of all hosts in dbservers using:

```
ansible dbservers -i hosts.ini -m command -a "uptime"
```

Here we can see the Ansible output:

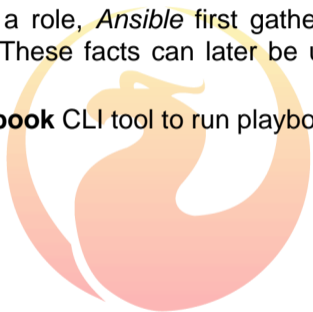
```
philippe@callandor ~ # ansible dbservers -i hosts.ini -m command -a "uptime"
db.example.com | success | rc=0 >>
21:16:24 up 93 days,  9:17,  4 users,  load average: 0.08, 0.03, 0.05
```


How to Run Playbooks?

For more complex scenarios, we can create playbooks or roles and ask *Ansible* to run them.

When we run a playbook or a role, *Ansible* first gathers a lot of useful facts about remote systems it manages. These facts can later be used in playbooks, templates, config files, etc.

We can use the **ansible-playbook** CLI tool to run playbooks.



How to Run Playbooks?

Given an inventory file like this:

```
[dbservers]  
db.example.com
```

We may have a playbook that connects to hosts in **dbservers** group, executes the **uptime** command, and then displays that command's output.

Now let's create a simple playbook to see how it can be ran.

How to Run Playbooks?

Here is the main.yml file for the playbook we just described:

```
---  
  
- hosts: dbservers  
  
  tasks:  
    - name: retrieve the uptime  
      command: uptime  
      register: command_result # Store this command's result in this variable  
  
    - name: Display the uptime  
      debug: msg="{{ command_result.stdout }}" # Display command output here
```

How to Run Playbooks?

Now we can run the playbook and see it's output here:

```
philippe@callandor ~ $ ansible-playbook -i hosts.ini main.yml

PLAY [dbservers] *****

GATHERING FACTS *****
ok: [db.example.com]

TASK: [retrieve the uptime] *****
changed: [db.example.com]

TASK: [Display the uptime] *****
ok: [db.example.com] => {
  "msg": " 15:54:47 up 3 days, 14:32,  2 users,  load average: 0.00, 0.01, 0.05"
}

PLAY RECAP *****
db.example.com          : ok=3    changed=1    unreachable=0    failed=0
```

Ansible targets

Ansible have to be run on a Gnu/Linux or MacOSx system.

But *Ansible* have modules to work with Windows target.

See : [List of Windows modules](#).



What can be done in the future

Since we have a good python driver, we could create some Firebird module

- firebird_db: Creates/removes a given db.
- firebird_user: Adds/removes users and roles from a db.
- firebird_privs: Grant/revokes privileds on db objects.

and certainly others if needed.



Thank you !



References

These slides and samples <https://github.com/pmakowski/fbconf-2016>

Ansible quick start video <http://www.ansible.com/videos>

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>

Jinja2 for better Ansible playbooks and templates

<https://blog.codecentric.de/en/2014/08/jinja2-better-ansible-playbooks-templates/>

Managing PostgreSQL with Ansible

<http://slides.com/apatheticmagpie/managing-postgres-with-ansible-fosdem>