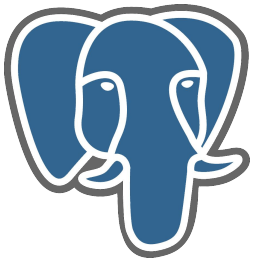
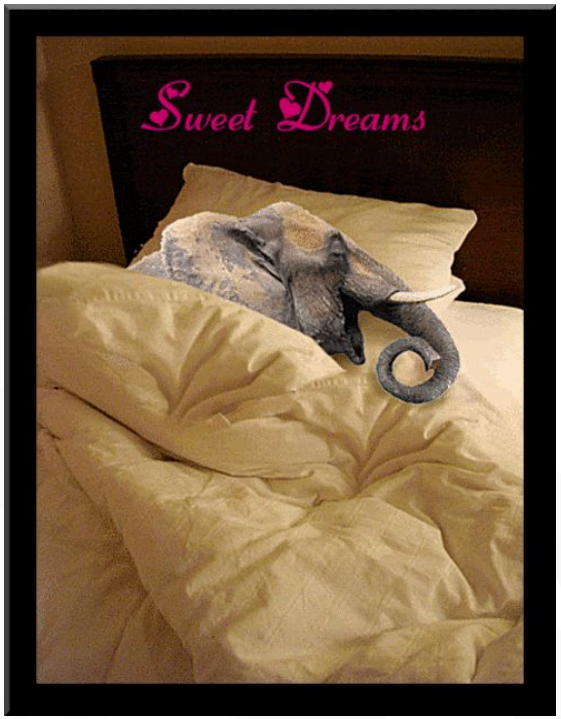


Keep Calm and Install PostGreSQL

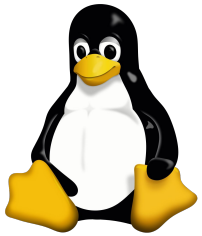
Using

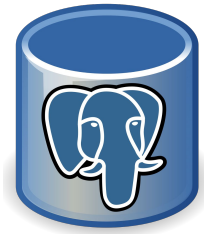
ANSIBLE

13 Ottobre 2017
PGDay.IT



In un futuro non molto lontano ...





… dovremo realizzare un nuovo progetto…

Bello! Lo so fare di sicuro!

Quand’è che si comincia?

… ma come al solito…

… il diavolo si nasconde nei dettagli!

Ci sono un paio di requisiti...

- L'applicazione deve essere scalabile nel cloud:
  - orizzontalmente
    - load balancer sul web server
  - verticalmente
    - se l'applicazione avrà successo, si vorrà replicare una struttura simile in altre zone AWS
- Il database PostgreSQL è composto da:
  - un master
  - 2 slave in replica a caldo
  - un server per i backup incrementali

Idee?

Per esempio nel caso dei server slave PostGreSQL si potrebbe:

- predisporre il server con il database preinstallato
- creare uno script per collegarlo al server master
- preparare un'immagine del server
- se si deve aggiungere un nuovo slave, lo si crea a partire dall'immagine

- Tempi:
  - setup del server (ma lo si sarebbe dovuto comunque fare)
  - setup alla creazione di un nuovo server per via di conf specifiche (nome host, parametri di connessione, etc)

E se devo pubblicare un aggiornamento?

- di sicurezza
- di un parametro di configurazione
- aggiungere allo slave un’estensione di PgSql

… su 100 server slave???

Denis Gasparin
Senior DBA and Web Developer

- Sviluppo di soluzioni software basate su PostgreSQL
  - PHP, NodeJS, Ruby, Python
- Cloud Architect
- Analista e Database Administrator
- Contributor del driver PDO PostgreSQL per PHP
- Pgrepup: tool opensource per l'aggiornamento a caldo di PostgreSQL
- rtshome.pgsql: ruolo ansible per interagire con PostGreSQL

DevOps (a clipped compound of "software **DEV**elopment" and "information technology **OP**eration**S**") is a term used to refer to a set of practices that emphasize the collaboration and communication of both software developers and information technology (IT) professionals while automating the process of software delivery and infrastructure changes.

It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.

https://en.wikipedia.org/wiki/DevOps

Non c’è un tool unico ma un insieme di strumenti che si possono utilizzare

Ad esempio nel **Software**:

- Version control tools: GIT, Mercurial, etc
- Continuous integration tools: Jenkins, Travis CI (github), etc
  - build
  - test
- Packaging
- Distribution

Nel software è semplice… alla fine si tratta di software che gestisce altro software

E nell'**hardware**?

- cosa c'entrano GIT o Mercurial con un disco rigido da aggiungere ad un server?
- il server funziona…
  - perché dovrei aver di un sistema di integrazione continua?
  - basta un tool di monitoring
- Packaging e distribution?

… a maggior ragione se il server è fisico e non virtuale...

Infrastructure as Code

Infrastructure as code (IaC) is the **process of managing and provisioning** computer data centers through **machine-readable definition files**, rather than physical hardware configuration or interactive configuration tools.

Both physical equipment such as **bare-metal servers** as well as **virtual machines** and associated configuration resources are called "infrastructure", although they have nothing to do with actual infrastructure.

The definitions may be in a **version control system**. It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.

https://en.wikipedia.org/wiki/Infrastructure_as_Code

Alcuni esempi di Configuration Management Tools:

- CFEngine
  - prima release nel 1993
  - gestiva solo workstation Unix
  - proprio linguaggio (DSL)
- Puppet:
  - prima release nel 2005
  - scritto in Ruby
  - configurabile con un proprio DSL
- Chef:
  - prima release nel 2009
  - scritto in Ruby ed Erlang
  - proprio DSL
- Saltstack:
  - prima release nel 2011
  - scritto in Python

- Caratteristiche principali dei software CFM:
  - definiscono un proprio linguaggio di configurazione
  - richiedono uno più server dove essere installati
  - richiedono un agente installato nei server controllati (o slave)
  - idempotenti
    - *Un'operazione che può essere ripetuta più volte senza cambiarne il risultato*

- Ansible:
  - prima release nel 2012
  - scritto in Python
  - file di configurazione in YAML
  - agent-less

	puppet labs	CHEF	SALTSTACK	ANSIBLE
Prima release	2005	2009	2011	2012
Linguaggio	Ruby	Ruby	Python	Python
Configurazione	DSL	DSL	DSL	YAML
Setup	Master/Slave	Master/Slave	Master/Slave	Agentless
	4654	4998	8150	25801

(*) Grazie a Rubens Souza per l'idea: @Pycon 2016 e 2017

Perché scegliere Ansible?

- **No agent**:
  - Non è necessario installare agenti sui server da installare
    - Usa SSH per comunicare con l'infrastruttura
- **No master**:
  - Non c'è necessità di un server dove installare il software controllore
  - Si può lanciare da qualsiasi PC che possa raggiungere l'infrastruttura
    - requisito: linux o mac
- **No file temporanei o di configurazione su server remoti**
- **Semplice da installare ed imparare:**
  - no requisiti particolari, solo
    - ssh
    - python (2.6 e 2.7, in preview 3.0)
- **Configurazione dichiarativa via YAML**:
  - si scrive cosa fare (ad esempio *"installa nodejs"*)
  - non come farlo

Perché scegliere Ansible?

- **Moduli:**
  - libreria con più di 1300
  - possono essere scritti in Python
  - comandi di shell
  - ansible-galaxy
- **Sicurezza:**
  - comunicazione via ssh
  - diverse metodologie di autenticazione:
    - sudo, su
    - pbrun, pfexec, doas, dzdo, ksu
  - possibilità di criptare le password (vault)
- **Configurazione per host o gruppi di host**
- **Gestione del cambio di stato**
  - ogni task riporta dopo l'esecuzione se è cambiato o meno

PostGreSQL installation… for dummies

```
$ sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/
$(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

$ sudo apt-get install wget ca-certificates

$ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc \
  | sudo apt-key add -

$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install postgresql-9.6 postgresql-contrib-9.6 \
  python-psycopg2

$ update-rc.d postgresql enable
```

PostGreSQL installation … for geeks

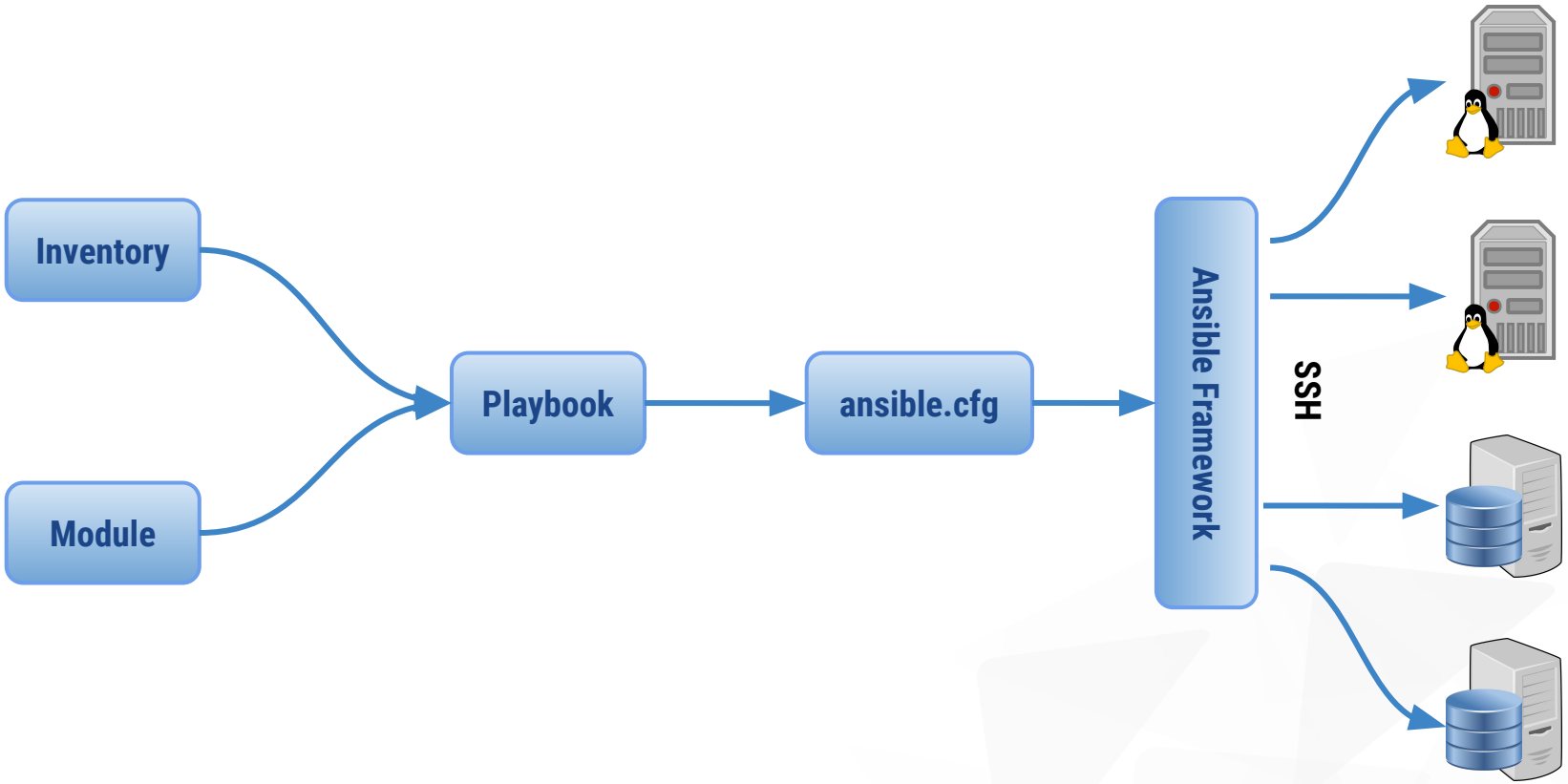
```
#!/bin/sh

# script to add apt.postgresql.org to sources.list

# from command line
CODENAME="$1"
# lsb_release is the best interface, but not always available
if [ -z "$CODENAME" ]; then
    CODENAME=$(lsb_release -cs 2>/dev/null)
fi
# parse os-release (unreliable, does not work on Ubuntu)
if [ -z "$CODENAME" -a -f /etc/os-release ]; then
    . /etc/os-release
    # Debian: VERSION="7.0 (wheezy)"
    # Ubuntu: VERSION="13.04, Raring Ringtail"
    CODENAME=$(echo $VERSION | sed -ne 's/.*(\(.*\)).*/\1/')
fi
# guess from sources.list
if [ -z "$CODENAME" ]; then
    CODENAME=$(grep '^deb ' /etc/apt/sources.list | head -n1 | awk '{ print $3 }')
fi
# complain if no result yet
if [ -z "$CODENAME" ]; then
    cat <<EOF
Could not determine the distribution codename. Please report this as a bug to
pgsql-pkg-debian@postgresql.org. As a workaround, you can call this script with
the proper codename as parameter, e.g. "$0 squeeze".
EOF
    exit 1
fi

# errors
set -e

cat <<EO
This scr
your system. The distribution codename used will be $CODENAME-pgdg.

EOF

---------------------------------- CUT CUT CUT CUT CUT ----------------------------------

echo "Running apt-get update ..."
apt-get update

cat <<EOF

You can now start installing packages from apt.postgresql.org.

Have a look at https://wiki.postgresql.org/wiki/Apt for more information;
most notably the FAQ at https://wiki.postgresql.org/wiki/Apt/FAQ
EOF
```

Lo script completo è disponibile su:
https://anonscm.debian.org/cgit/pkg-postgresql/postgresql-common.git/plain/pgdg/apt.postgresql.org.sh

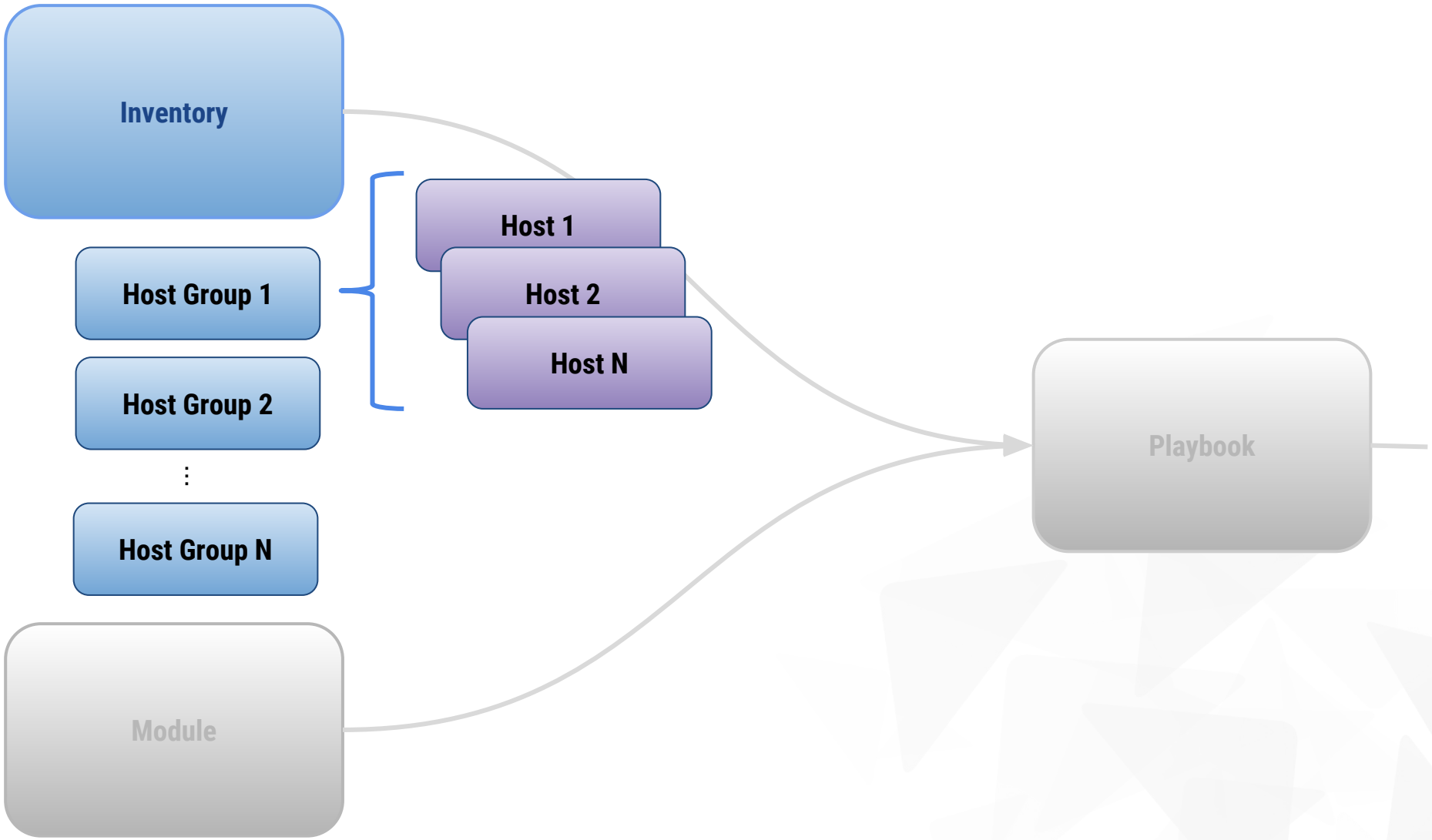
Keep calm… with Ansible c'est plus facile :-D

```
---
  - hosts: all
    become: true
    tasks:
    - name: add apt postgresql repo key
      apt_key: url=https://www.postgresql.org/media/keys/ACCC4CF8.asc

    - name: add apt postgresql repo
      apt_repository: "repo='deb http://apt.postgresql.org/pub/repos/apt/ jessie-pgdg main'"

    - name: install postgresql
      apt: name="{{ item }}" state=present
      with_items:
        - postgresql-9.6
        - postgresql-contrib-9.6
        - python-psycopg2

    - name: Start Postgresql
      service: name=postgresql state=started enabled=yes
```

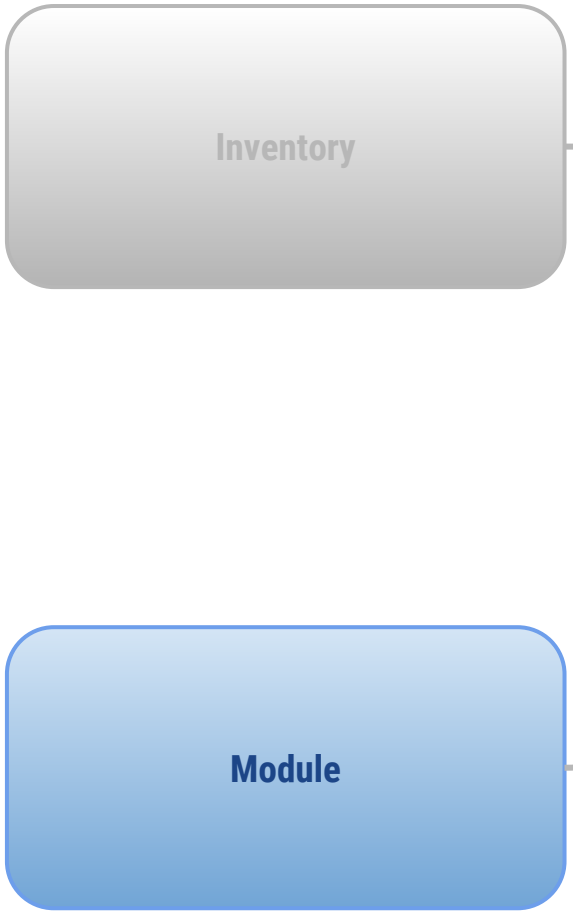
Le componenti di Ansible



Le componenti di Ansible - Inventory



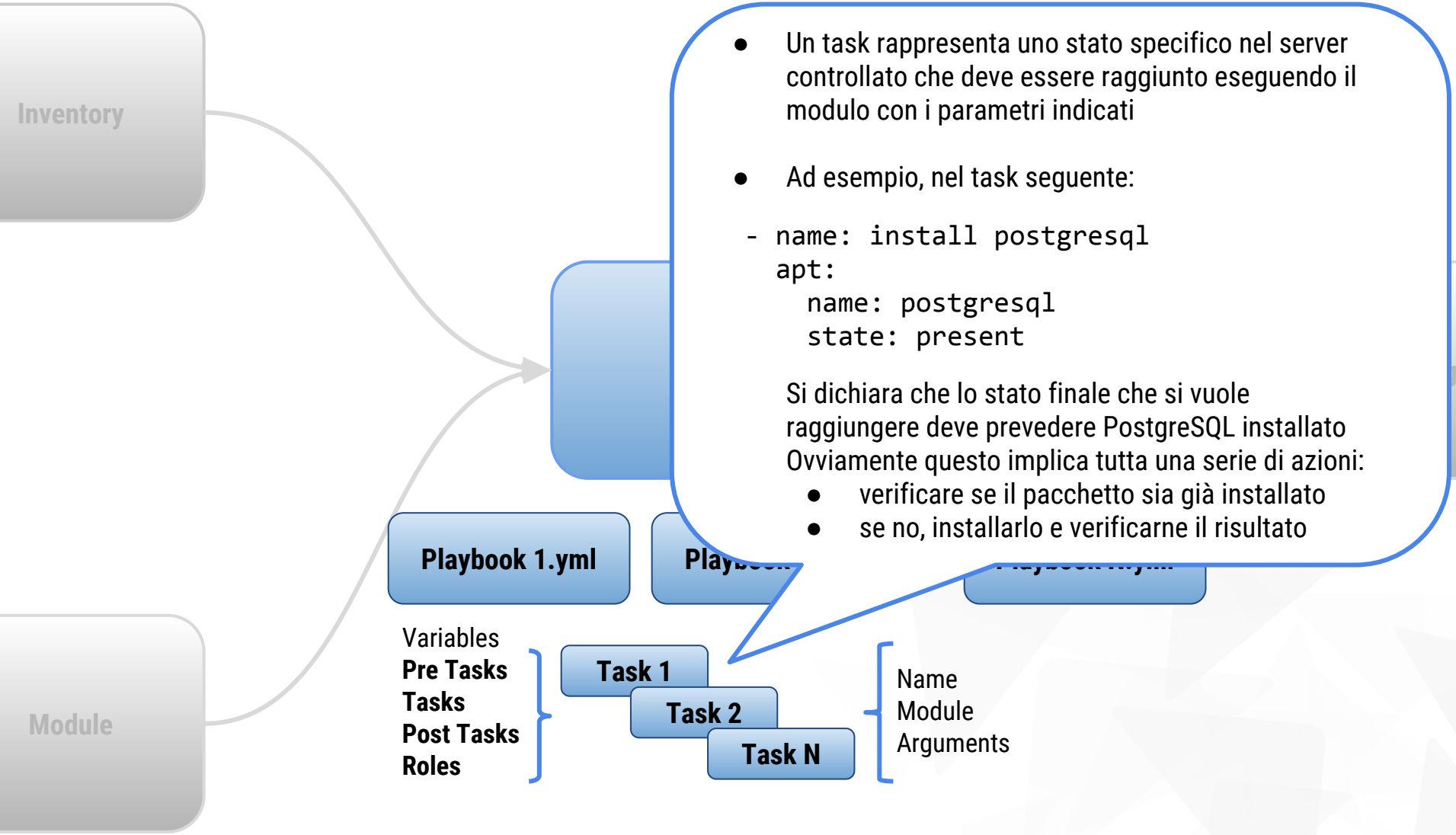
Le componenti di Ansible - Module



- Un modulo rappresenta un blocco specifico di codice eseguito sul sistema controllato al fine di eseguire un task.
- I moduli in Ansible sono più di 1000 e se ne possono scrivere sempre di nuovi. Ecco alcuni esempi:
  - **Package Management**: apt, yum, etc
  - **Source Control**: git, hg, ...
  - **Commands**: shell, command, ...
  - **Monitoring**: zabbix, nagios, ...
  - **Cloud**: amazon, azure, docker, …
  - **Database**: postgresql_database, ...

http://docs.ansible.com/ansible/modules_by_category.html

Le componenti di Ansible - Playbook



- Un task rappresenta uno stato specifico nel server controllato che deve essere raggiunto eseguendo il modulo con i parametri indicati
- Ad esempio, nel task seguente:

```
- name: install postgresql
  apt:
    name: postgresql
    state: present
```

Si dichiara che lo stato finale che si vuole raggiungere deve prevedere PostgreSQL installato
Ovviamente questo implica tutta una serie di azioni:

- verificare se il pacchetto sia già installato
- se no, installarlo e verificarne il risultato

Esempio di playbook - file YAML


```
---

name: Martin
job: Developer
skill: Elite
employed: True

foods:
    - Apple
    - Orange
    - Strawberry
    - Mango

languages:
    perl: Elite
    python: Elite
    pascal: Lame

education: |
    4 GCSEs
    3 A-Levels
    BSc in the Internet of Things
```

This denotes the start of a YAML file (optional)

We assign the value "`Martin`" to "`name`", etc, etc.<
To "`employed`" is assigned a boolean value

We assign a list (an array) of 4 items to "`foods`"

We assign a dictionary to "`languages`". The dictionary has three keys (`perl`, `python`, `etc`) and three values (`Elite`, …)

Example of multiline string

Esempio di playbook


```
---
- hosts: dbservers
  vars:
    postgresql_version: 9.6
  pre_tasks:
  - name: print a message
    msg: "I'm going to install PostGreSQL {{postgresql_version}}"
  tasks:
     ...
     - name: install postgresql
       apt: name="{{ item }}" state=present
       with_items:
         - "postgresql-{{postgresql_version}}"
         - "postgresql-contrib-{{postgresql_version}}"
  post_tasks:
  - name: print a message
    msg: Yeah, PostGreSQL kicking and alive :-)
  roles:
  - rtp
  - locale
```

Play

...qualche altra curiosità!

```
---
  - hosts: all
    become: true
    tasks:
    - name: add apt postgresql repo key
      apt_key: url=https://www.postgresql.org/media/keys/ACCC4CF8.asc

    - name: add apt postgresql repo
      apt_repository: "repo='deb http://apt.postgresql.org/pub/repos/apt/ jessie-pgdg main'"

    - name: install postgresql
      apt: name="{{ item }}" state=present
      with_items:
        - postgresql-9.5
        - postgresql-contrib-9.5
        - python-psycopg2

    - name: Start Postgresql
      service: name=postgresql state=started enabled=yes
```

Necessario: per installare pacchetti dobbiamo essere root

- possiamo definire dei loop
- esempio di uso di:
  - template jinja2
  - variabili

Variabili

Possono essere di tre tipi:

1. Host variables:
   - definite per host o per gruppi di host all'interno dell'inventory file

```
test_server ansible_host=192.168.33.10
[test_server:vars]
postgresql_version=9.5
```

2. Facts:
   - definite per host o per gruppi mediante:
     - la sezione vars nel playbook
     - il modulo `set_fact`

```
- name: Set needed PostgreSQL version
  set_fact: postgresql_version="9.5"
```

3. Dynamic variables
   - variabili definite come esito di un task

```
- stat:  path="/etc/foo.conf"
  register: foo_config_file
```

Esempio di utilizzo delle variabili

Inventory

```
db1 ansible_host=192.168.33.10
[db1:vars]
postgresql_version=9.5
debian=wheezy

db2 ansible_host=192.168.33.11
[db2:vars]
postgresql_version=9.6
debian=jessie

[dbservers]
db1
db2
```

Esempio di utilizzo delle variabili

Playbook

```
---
  - hosts: dbservers
    become: true
    tasks:
    - name: add apt postgresql repo key
      apt_key: url=https://www.postgresql.org/media/keys/ACCC4CF8.asc

    - name: add apt postgresql repo
      apt_repository: "repo='deb http://apt.postgresql.org/pub/repos/apt/
{{debian}}-pgdg main'"

    - name: install postgresql
      apt: name="{{ item }}" state=present
      with_items:
        - "postgresql-{{postgresql_version}}"
        - "postgresql-contrib-{{postgresql_version}}"
        - python-psycopg2

    - name: Start Postgresql
      service: name=postgresql state=started enabled=yes
```

Moduli per interazione con il database PostGreSQL

- `postgresql_db`:
  - Add or remove PostgreSQL databases from a remote host
- `postgresql_schema`:
  - Add or remove PostgreSQL schema from a remote host

- `postgresql_ext` :
  - Add or remove PostgreSQL extensions from a database
- `postgresql_lang`:
  - Adds, removes or changes procedural languages with a PostgreSQL database

- `postgresql_user`:
  - Adds or removes a users (roles) from a PostgreSQL database.
- `postgresql_privs`:
  - Grant or revoke privileges on PostgreSQL database objects.

http://docs.ansible.com/ansible/latest/list_of_database_modules.html#postgresql

Esempio di creazione di un database e utenti

```
---
- hosts: dbservers
  vars:
    pgsql_user: my_superuser
    pgsql_pwd: my_supersecret_password
  tasks:
    - name: create "foo_user"
      postgresql_user:
        login_host: "{{ansible_host}}"
        login_user: "{{pgsql_user}}"
        login_password: "{{pgsql_pwd}}"
        name: foo_user
        password: foo_pwd

    - name: create "foo" database
      postgresql_db:
        login_host: "{{ansible_host}}"
        login_user: "{{pgsql_user}}"
        login_password: "{{pgsql_pwd}}"
        name: foo
        owner: foo_user
        template: my_template
```

Esempio per impostare uno slave in hot standby

```
- hosts: slave_dbservers
  tasks:
    - name: configure slave server for hot standby
      ini_file:
        path: "/etc/postgresql/{{version}}/{{cluster}}/postgresql.conf"
        option: "{{item.param}}"
        value: "{{item.value}}"
        section: null
      with_items:
        - {param: port, value: 5433}
        - {param: shared_buffers, value: 10GB}
        - {param: hot_standby, value: on}
      register: pgsql_conf

    - name: Restart Postgresql
      service: name=postgresql state=restarted
      when: pgsql_conf.changed
```

Ruolo rtshome.pgsql

- Aggiunge quattro nuovi moduli per PostGreSQL:
  - postgresql_table: crea/elimina una tabella
  - postgresql_row: crea/elimina una riga in una tabella
  - postgresql_query: esegue una query e ne ritorna le righe
  - postgresql_command: esegue un comando

- Disponibile attraverso Ansible Galaxy

```
$ ansible-galaxy install rtshome.pgsql
```

- Opensource e disponibile su Github:

  https://github.com/rtshome/ansible_pgsql

Modulo postgresql_table

- Argomenti principali:
  - schema: Schema where to add/drop the table
  - name: Name of the table
  - state: The table state
  - owner: Owner of the table
  - columns: List of objects with name, type and null keys
  - primary_key: List with column names composing the primary key

```
# Ensure config(parameter, value) table is present in database

- postgresql_table:
    database: my_app
    name: config
    state: present
    columns:
      - { name: parameter, type: text, null: False }
      - { name: value, type: text, null: False }
    primary_key:
      - parameter
```

Modulo postgresql_row

- Argomenti principali:
  - schema: Schema where to add/drop the table
  - table: Name of the table
  - state: The row state
  - row: Dictionary with the fields of the row

```
# Ensure row with fields parameter = "environment" and value =
# "production" is present in db

- postgresql_row:
    database: my_app
    table: config
    row:
        parameter: environment
        value: production
    state:
        present
```

Modulo postgresql_query

- Argomenti principali:
  - query: Query to execute
  - parameters: Parameters of the query as:
    - list if positional parameters are used in query
    - dictionary if named parameters are used

```
# Fetch row from "config" table with parameter name 'environment'
- postgresql_query:
    database: my_app
    query: "SELECT * FROM config WHERE parameter = %(pname)s"
    parameters:
      pname: environment
  register: query_results

# query_results contains:
{
    rows: [{parameter: 'environment', value: 'production'}]
    rowCount: 1
    executed_query: "SELECT * FROM config WHERE parameter = 'environment'"
}
```

Modulo postgresql_command

- Argomenti principali:
  - command: The SQL command to execute
  - parameters: Parameters of the command as:
    - list if positional parameters are used in query
    - dictionary if named parameters are used

```
# Change the environment of the application from 'production' to
# 'staging'
- postgresql_command:
    database: my_app
    command: >
          UPDATE config SET value = %(val)
          WHERE parameter = %(prm)
    parameters:
      val: staging
      prm: env
  register: command_results

# command_results contains:
{executed_command: "UPDATE config SET value='staging' WHERE
parameter='environment'",
rowCount: 1}
```

Esempio: creazione di cronjobs inseriti in tabella

```
- postgresql_query:
    name: "Fetch all the jobs for the sale department"
    query: >
        SELECT name, minute, hour, weekday, month, job, job_user
        FROM job
        WHERE department = %(dep)
    parameters:
      dep: sales
    register: jobs_to_schedule

- cron:
    name: "Schedule job {{name}} for sales department"
    minute: "{{ item.minute }}"
    hour: "{{ item.hour }}"
    weekday: "{{ item.weekday }}"
    month: "{{ item.month }}"
    job: "{{ item.job }}"
    user: "{{ item.job_user }}"
  with_items: jobs_to_schedule.rows
```

Grazie!

Smart Solutions

denis@gasparin.net

@rtshome

http://www.gasparin.net