Puppet, a centralized management system

Andrea Veri

Open Source Day 2014, Udine

Puppet?

Da wikipedia:

Puppet è uno strumento Open Source di gestione centralizzata delle configurazioni di un server. Ad oggi funzionante sia su sistemi Unix-like che Microsoft Windows. Include un suo **linguaggio dichiarativo** per descrivere le configurazioni dei sistemi che si andranno a configurare.

Linguaggio dichiarativo?

- Rappresenta lo stato a cui l'entità finale dovrà assomigliare
- Il *come* tale entità verrà creata non è rilevante ai fini dello stesso linguaggio.

Nel caso di Puppet

- Viene reso disponibile agli amministratori di sistema un particolare linguaggio a cui essi possono far riferimento durante l'espletamento delle loro funzioni
- Tale linguaggio è molto simile ad una serie di istruzioni che verranno interpretate ed eseguite in ordine casuale
- Le istruzioni vengono inserite all'interno di file testuali chiamati **manifesti**.
- Le istruzioni all'interno dei **manifesti** verranno poi richieste dai client di ciascun target che procederà poi a verificare ed applicare le nuove eventuali modifiche nei servizi di riferimento secondo il modello di distribuzione convergente. (**Convergent Distributed Model**)

Amministratori di sistema?

Istruzioni su manifesti?

Modello di distribuzione convergente?

Alcune premesse

Qual'è il ruolo di un amministratore di sistema?

- Si occupa di mantenere e gestire le risorse informatiche (software, hardware) all'interno di un determinato ambito lavorativo o domestico
- Monitora costantemente le risorse ed i servizi evitando o minimizzando qualsiasi tipo di downtime
- Risponde alle query degli utenti ed esegue il troubleshooting di eventuali disservizi per ripristinarne le funzionalità
- Gestisce gli account utente dell'intera infrastruttura nonchè il monitoraggio di operazioni pianificate quali i backup notturni
- **Automatizza i processi per evitare di ripetere i medesimi comandi ad ogni successiva installazione di una nuova macchina fisica, virtuale o di un semplice servizio**

Alcune premesse

Automatizzare

In passato

- Nei periodi che precedono l'incremento esponenziale dei dispositivi a disposizione di una azienda o di un ente e della successiva informatizzazione di molteplici contesti della vita quotidiana il numero dei sistemi era molto esiguo
- Replicare eventuali configurazioni di sistema poteva e veniva svolto manualmente e singolarmente su ogni sistema in possesso

Ad oggi

- Drastico aumento dei computer a disposizione delle aziende fino a contare diverse centinaia di migliaia nelle organizzazioni di maggior rilievo
- Necessità di automatizzare i processi ed i procedimenti volti alla installazione di macchine fisiche ex novo ed al deploy di macchine virtuali con le relative configurazioni di sistema pre-impostate

Back on track

Gli elementi che lo compongono

- Un master server denominato **Puppet Master**
- Un client denominato **Puppet Client**

Come si interfacciano

- Il Puppet Client si connette periodicamente al master ed ottiene il **catalogo** dei cambiamenti previsti per quel nodo
- A connessione avvenuta il client verifica eventuali modifiche alle configurazioni di sistema per quel target
- Qualora vengano riscontrate differenze tra la copia della configurazione all'interno del Puppet Master e quella presente all'interno del sistema di riferimento, il client applica le modifiche in locale

Un passo in avanti, caso pratico

Organizzazione dell'infrastruttura ed esigenze

- Infrastruttura composta da 100 server
- Utilizzo massiccio di tecnologie che permettono la virtualizzazione
- Conseguente aumento radicale del numero dei sistemi a disposizione
- Necessità di effettuare il deploy di nuove macchine virtuali
- Necessità di assegnare ogni eventuale macchina virtuale creata ad un pool già esistente di sistemi per favorirne il failover, ridondanza o semplicemente il load balancing

Un passo in avanti, caso pratico

Nell'era precedente alla creazione di Puppet

- Necessità di ricreare le esatte configurazioni dei servizi su ogni singolo sistema
- Dispendio di energie, tempo e fondi non indifferente
- Possibilità di compiere degli errori nel ricreare esattamente un determinato sistema ed i relativi servizi associati
- Impossibilità di prescindere dall'intervento manuale di un amministratore di sistema

Un passo in avanti, caso pratico

Nell'era post creazione di Puppet

- Server centrale contenente tutte le configurazioni di sistema e dei servizi
- Possibilità di assegnare ad ogni nodo determinate classi a seconda dei task per esso individuati
- Risparmio notevole in termini di tempo, costi e risorse impiegate
- Certezza matematica che i file di configurazione modificati saranno esattamente i medesimi rispetto a quelli disponibili all'interno del Puppet Master tramite controllo degli MD5SUM ad ogni run del Puppet Client
- Puppet, in combinazione con strumenti quali Cobbler od altri provisioner di macchine virtuali, diviene uno strumento potentissimo

Un passo indietro, i cataloghi

- Il Puppet Master contiene al suo interno un web server basato sulla libreria Ruby WEBRick che fornisce, inoltre, l'autenticazione dei client stessi
- Il Puppet Client instaura una connessione con il Puppet Master ogni 30 minuti e si occupa, in primis, di eseguire lo scaricamento del **catalogo** relativo a quel nodo
- Ogni **catalogo** conterrà tutte le istruzioni che il Puppet Client dovrà eseguire nel nodo per portarlo dallo stato di partenza allo stato desiderato (from current to desired state)
- I singoli nodi non conoscono nulla degli altri nodi presenti all'interno del Puppet Master (ciò per motivi di sicurezza)
- Il Puppet Client procederà, quindi, ad analizzare il **manifesto** del nodo, il cuore delle istruzioni che il client analizzerà per modificare lo stato del sistema
- In questa situazione il Puppet Client sta ancora ricavando ed analizzando le informazioni necessarie ad eseguire una run completa. Da qui la possibilità di eseguire una **Dry Run** (tramite la flag **no-op**) senza applicare i cambiamenti previsti nel nodo

Un po' di codice per iniziare!

manifests/example.pp:

```
class example {
    package { httpd:
        ensure => present,
    }

    service { httpd:
        enable => true,
        require => Package['httpd'],
    }
}
```

Elementi presenti:

- Definizione di installazione del pacchetto **httpd** (Apache)
- Definizione di abilitazione del servizio **httpd** ad ogni riavvio
- Presenza di **require**, elemento che crea una dipendenza funzionale tra le due definizioni

Ancora codice!

manifests/example.pp:

```
    package { httpd:
        ensure => present,
    }

    service { httpd:
        enable => true,
        require => Package['httpd']
    }

    file { "/etc/httpd/sites.d":
        ensure => directory,
        recurse => true,
        purge => true,
        force => true,
        owner => "root",
        group => "root",
        mode => 0644,
        notify => Service["httpd"];
     }
```

Ancora codice, nuovi elementi:

- Definizione che crea e gestisce una directory
- Molteplici variabili che definiscono le caratteristiche che la directory dovrà avere nel suo stato finale:
  - recurse
  - purge
  - force
  - owner
  - group
  - mode
  - **notify**

Handlers!

- L'elemento **notify** introduce la tematica degli **handlers**
- Permettono di lanciare un determinato comando ogni qual volta un file od una directory a cui essi sono collegati viene modificato
- L'handler diviene a sua volta una dipendenza della risorsa che lo chiama in causa

+AndreaVeri

dragonsreach.it

@andrea_veri

av@gnome.org