Parte xx

Cfengine

Introduzione a Cfengine 749
- Primo approccio con la configurazione 750
- Sezioni e classi predefinite 753
- Classi più in dettaglio 758
- Variabili e stringhe 762
- Espressioni regolari 768

Cfengine: sezioni di uso comune 771
- Permessi e proprietà 772
- Sezione «control» 774
- Sezione «classes» o «groups» 776
- Sezione «copy» 777
- Sezione «directories» 782
- Sezione «disable» 783
- Sezione «files» 784
- Sezione «links» 786
- Sezione «processes» 788
- Sezione «shellcommands» 791
- Sezione «tidy» 791

Cfengine attraverso la rete 795
- Configurazione e avvio del demone 795
- Filosofia del sistema di distribuzione di Cfengine 800

Introduzione a Cfengine

Primo approccio con la configurazione 750
La variabile CFINPUTS 752
Simulazione 752
Sezioni e classi predefinite 753
Classi più in dettaglio 758
Variabili e stringhe 762
Elenchi 767
Espressioni regolari 768

Cfengine [1] è uno strano sistema di amministrazione di elaboratori Unix, la cui importanza si apprende solo con il tempo e con l'utilizzo. Il suo scopo è quello di facilitare l'amministrazione di tali sistemi operativi, soprattutto quando si dispone di un gruppo eterogeneo di questi su diversi elaboratori. Questi capitoli dedicati a Cfengine non pretendono di esaurire l'argomento, cercando piuttosto di semplificare il suo apprendimento che poi può essere approfondito leggendo la documentazione originale.

A prima vista, si può intendere Cfengine come l'interprete di un linguaggio molto evoluto. In questo capitolo si introduce l'uso specifico dell'eseguibile '**`cfengine`**', il cui scopo è interpretare un file di configurazione, ovvero il suo script, agendo di conseguenza.

«

Per funzionare, l'eseguibile '**cfengine**' richiede la presenza di un file di configurazione, che eventualmente può essere trasformato in script, se ciò può essere conveniente. La comprensione, anche elementare, del modo in cui si configura questo programma, è la chiave per capire a cosa può servire in generale Cfengine.

Il file di configurazione di '**cfengine**' ha una struttura speciale, in cui però si possono inserire commenti, preceduti dal simbolo '**#**', e righe vuote o bianche. In particolare, a proposito dei commenti, se questi si collocano alla fine di una direttiva, devono essere staccati da questa con uno o più spazi orizzontali.

Le direttive del file di configurazione vanno inserite all'interno di sezioni ed eventualmente all'interno di classi. In altri termini, il file di configurazione si articola in sezioni, che possono contenere direttive o scomporsi in classi, che a loro volta contengono le direttive. Come si intende, la suddivisione in classi è facoltativa, ma si tratta comunque di una caratteristica fondamentale di Cfengine, in quanto consente di selezionare le direttive da prendere in considerazione in base all'appartenenza o meno dell'elaboratore alle classi stesse.

Dal momento che il problema non è semplice da esporre, conviene iniziare subito con un esempio che possa essere verificato senza troppi problemi anche da un utente comune:

```
# Esempio di partenza

control:
    actionsequence = ( links )

links:
    /var/tmp/altra -> /tmp
```

Se questo file si chiama ‘`cfengine.conf`’ e si trova nella directory corrente, qualunque essa sia, se non è stata impostata la variabile di ambiente ‘**`CFINPUTS`**’, si può avviare l’interpretazione di tale file semplicemente avviando l’eseguibile ‘**`cfengine`**’:

`$` **`cfengine`** [*Invio*]

Quello che si ottiene è soltanto la creazione del collegamento simbolico ‘`/var/tmp/altra`’ che punta in realtà alla directory ‘`/tmp/`’.

Se il file di configurazione fosse collocato altrove, eventualmente con un’altra denominazione, si potrebbe ottenere lo stesso risultato con il comando seguente, dove il nome del file viene aggiunto nella riga di comando:

`$` **`cfengine -f`** ***file_di_configurazione*** [*Invio*]

Infine, per realizzare uno script dalla configurazione, basta inserire all’inizio una riga simile a quella seguente (ammesso che l’eseguibile si trovi effettivamente in ‘`/usr/bin/`’):

```
#!/usr/bin/cfengine -f
```

In altri termini, lo script completo dell’esempio precedente sarebbe:

```
#!/usr/bin/cfengine -f
# Esempio di partenza

control:
    actionsequence = ( links )

links:
    /var/tmp/altra -> /tmp
```

La variabile CFINPUTS

«

La variabile di ambiente '**CFINPUTS**' serve per definire un percorso di ricerca per il file di configurazione. In generale, se si utilizza l'opzione '**-f**' specificando un percorso assoluto, a partire dalla radice (qualcosa che inizia con '**/**'), si tratta esattamente di quel file, altrimenti, se è disponibile la variabile '**CFINPUTS**', questa viene preposta al nome del file indicato. Per esempio, il comando

```
$ cfengine -f prova[Invio]
```

fa riferimento precisamente al file di configurazione '`$CFINPUTS/prova`', ovvero al file '`./prova`' se la variabile '**CFINPUTS**' non è disponibile.

Quando non si indica il file di configurazione, si fa implicitamente riferimento al nome '`cfengine.conf`'. In tal caso si tratta precisamente di '`$CFINPUTS/cfengine.conf`', ovvero del file '`./cfengine.conf`' in mancanza della variabile '**CFINPUTS**'.

Simulazione

«

Cfengine è un sistema molto potente, i cui script definiscono operazioni molto complesse con poche direttive. Di fronte a direttive distruttive occorre essere sicuri del risultato che si ottiene effettivamente. Per verificare cosa farebbe Cfengine con la configurazione stabilita, senza eseguire realmente la cosa, si può usare l'opzione '**-n**', abbinata a '**-v**': la prima simula l'esecuzione; la seconda mostra nel dettaglio cosa succede o cosa dovrebbe succedere.

Finché non si è sicuri del proprio script o della propria configurazione, occorre ricordare di fare tutte le prove utilizzando l'opzione '**-n**'.

Realizzando uno script con questo intento, basta modificare la prima riga nel modo seguente:

```
#!/usr/bin/cfengine -n -v -f
```

Sezioni e classi predefinite

Le direttive del file di configurazione vanno inserite all'interno di sezioni, che a loro volta possono suddividersi in classi. Le sezioni rappresentano dei tipi di azione e i loro nomi sono già stabiliti.

sezione_ovvero_tipo_di_azione **:**

definizione_della_classe **: :**

direttiva_o_azione

...

...

Negli esempi visti fino a questo punto, sono state mostrate le sezioni '**control**' e '**links**'. Nella sezione '**control**' è stata inserita la direttiva '**actionsequence**', che ha l'aspetto di un assegnamento a una variabile:

```
control:
    actionsequence = ( links )
```

Le direttive, ovvero le istruzioni che possono apparire all'interno di classi o di sezioni non suddivise in classi, possono occupare una o più righe, senza bisogno di simboli di continuazione e senza bisogno di simboli per la conclusione delle istruzioni stesse.

In questo caso particolare, si tratta di assegnare uno o più nomi, che rappresentano altrettante sezioni, alla sequenza di esecuzione. In pratica, la direttiva dell'esempio stabilisce che deve essere eseguita la sezione '**links**'. Se non venisse specificata in questo modo, la sezione '**links**' non verrebbe presa in considerazione. Pertanto, la configurazione seguente non produrrebbe alcunché:

```
# Non fa nulla

control:
    actionsequence = ( )

links:
    /var/tmp/altra -> /tmp
```

Il prossimo esempio dovrebbe chiarire definitivamente questo particolare. Si osservi il fatto che si vuole eseguire prima la sezione '**tidy**' e poi la sezione '**links**', anche se l'ordine in cui sono mostrate poi le sezioni è inverso.

```
control:
    actionsequence = ( tidy links )

links:
    /var/tmp/altra -> /tmp

tidy:
    /var/tmp pattern=* age=30 recurse=inf
```

In questo caso, la sezione '**tidy**' serve a programmare la cancellazione di file e directory. Per la precisione, la direttiva che si vede cancella tutti i file e le directory a partire da '`/var/tmp/`', purché

la data di accesso sia trascorsa da almeno 30 giorni. Si osservi anche l'opzione '**recurse=inf**', che richiede una ricorsione infinita nelle sottodirectory. In condizioni normali, questa ricorsione non dovrebbe attraversare i collegamenti simbolici, mentre per ottenere tale comportamento occorrerebbe aggiungere l'opzione '**-l**'. Pertanto, anche se dovesse esistere già il collegamento simbolico '`/var/tmp/altra`', che punta a '`/tmp/`', questa directory non verrebbe scandita se non richiesto espressamente.

Le classi sono la caratteristica fondamentale di Cfengine, perché consentono di distinguere le direttive di una sezione in base a una sottoclassificazione che serve a selezionare un gruppo ristretto di elaboratori. In pratica, consente di indicare direttive differenti in base alla «classificazione» a cui appartengono gli elaboratori presi in considerazione. Si osservi l'esempio seguente:

```
control:
    actionsequence = ( links )

links:
    linux_2.2.15::
        /var/tmp/altra -> /tmp
    linux_2.2.16::
        /var/tmp/altre -> /tmp
        /var/tmp/altri -> /tmp
```

Anche se poco significativo, l'esempio è abbastanza semplice e dovrebbe permettere di comprendere il senso della distinzione in classi. In questo caso, la sezione '**links**' si articola in due classi, denominate '**linux_2.2.15**' e '**linux_2.2.16**'. Se viene usato questo file in un elaboratore con un sistema GNU/Linux avente un kernel 2.2.15, si ottiene il collegamento simbolico '`/var/tmp/altra`', mentre con un kernel 2.2.16 si otterrebbero due collegamenti simbolici: '`/var/tmp/altre`' e '`/var/tmp/altri`'. Naturalmente, que-

sta operazione può non avere molto significato in generale, ma l'esempio serve a mostrare la possibilità di indicare direttive diverse in base alla classe a cui appartiene l'elaboratore.

La classe serve principalmente a individuare il sistema operativo (nel caso di GNU/Linux si tratta del nome del kernel), in modo da cambiare azione in funzione delle consuetudini di ogni ambiente. In questo caso, volendo selezionare un sistema GNU/Linux senza specificare la versione del kernel sarebbe stato sufficiente indicare la classe '**`linux`**'. Tuttavia, come si vede nell'esempio, esistono delle classi più dettagliate che permettono di raggiungere anche altre caratteristiche. Per conoscere quali sono le classi valide nell'elaboratore che si utilizza in un certo momento, basta il comando seguente:

```
$ cfengine -p -v[Invio]
```

A titolo di esempio, ecco cosa potrebbe comparire:

```
GNU Configuration Engine -
cfengine-1.5.3
Free Software Foundation 1995, 1996, 1997
Donated by Mark Burgess, Centre of Science and Technology
Faculty of Engineering, Oslo College, 0254 Oslo, Norway

-------------------------------------------------------------------------

Host name is: dinkel
Operating System Type is linux
Operating System Release is 2.2.15
Architecture = i586


Using internal soft-class linux for host dinkel

The time is now Tue Oct 24 16:11:18 2000


-------------------------------------------------------------------------
```

```
Additional hard class defined as: 32_bit
Additional hard class defined as: linux_2.2.15
Additional hard class defined as: linux_i586
Additional hard class defined as: linux_i586_2.2.15
Additional hard class defined as:
 linux_i586_2_2_15__1_Thu_Aug_31_15_55_32_CEST_2000

GNU autoconf class from compile time: linux-gnu

  Careful with this - it might not be correct at run time if you have
  several OS versions with binary compatibility!

Address given by nameserver: 192.168.1.1
dinkel: No preconfiguration file
Accepted domain name: undefined.domain


Defined Classes = ( any debian linux dinkel undefined_domain Tuesday Hr16 Min11
 Min10_15 Day24 October Yr2000 32_bit linux_2_2_15 linux_i586 linux_i586_2_2_15
 linux_i586_2_2_15__1_Thu_Aug_31_15_55_32_CEST_2000 linux_gnu 192_168_1
 192_168_1_1 )

Negated Classes = ( )

Installable classes = ( )

Global expiry time for locks: 120 minutes

Global anti-spam elapse time: 0 minutes

Extensions which should not be directories = ( )
Suspicious filenames to be warned about = ( )
```

Le classi disponibili sono quindi quelle elencate nell'insieme '**`Defined Classes`**'. Si può osservare che è accessibile anche una classe con il nome della distribuzione GNU/Linux (in questo caso è Debian), oltre agli indirizzi IP abbinati all'interfaccia di rete.

Classi più in dettaglio

«

Le classi non sono necessariamente nomi singoli; possono essere delle espressioni composte da più nomi di classe, uniti tra loro attraverso operatori booleani opportuni. Prima di arrivare a descrivere questo, è bene riassumere le classi più comuni e vedere come si possono definire delle classi nuove. Una classe elementare può essere:

- la parola chiave '**`any`**', che rappresenta tutti gli elaboratori;
- il nome del sistema operativo o del kernel, assieme a una serie di varianti che includono altre caratteristiche dell'architettura del sistema;
- il nome finale dell'elaboratore (senza il dominio eventuale a cui appartiene);
- il nome che identifica una componente del tempo (giorno, ora, minuto, ecc.), come si vede nella tabella u56.10;
- il nome di un gruppo di classi definito per comodità dell'utilizzatore;
- il nome di una classe libera definito per comodità dell'utilizzatore.

Tabella u56.10. Elenco delle classi di Cfengine riferite al tempo.

Nome	Descrizione
Monday, Tuesday, Wednesday,...	Giorni della settimana.
Hr00, Hr01,... Hr23	Ore del giorno.
Min00, Min01,... Min59	Minuti di un'ora.

Nome	Descrizione
Min00_05, Min05_10,... Min55_00	Intervalli di cinque minuti.
Day1, Day2,... Day31	Giorni del mese.
January, February,... December	Mesi dell'anno.
Yr1999, Yr2000, Yr2001,...	Anni.

Si può definire un gruppo di classi attraverso la sezione '**`classes`**' o '**`groups`**', in cui le direttive servono per definire delle classi nuove raggruppando più classi preesistenti:

```
classes:|groups:
    gruppo_di_classi = ( classe_1 classe_2... )
    ...
```

Per esempio, la dichiarazione seguente serve a raggruppare in due classi nuove le ore del mattino e le ore della sera, supponendo che ciò possa avere un significato pratico di qualche tipo:

```
classes:
    OreDelMattino = ( Hr06 Hr07 Hr08 Hr09 )
    OreDellaSera  = ( Hr18 Hr19 Hr20 Hr21 )
```

Inoltre si possono definire delle classi in base al risultato soddisfacente di un programma o di uno script. In altri termini, se un programma restituisce *Vero*, questo fatto può essere preso in considerazione come motivo valido per generare una classe. L'esempio seguente crea la classe '**`miashell`**' se è presente il file '`/bin/bash`' oppure il file '`/bin/zsh`':

```
classes:
    miashell = ( "/bin/test -f /bin/bash" "/bin/zsh" )
```

Si possono dichiarare anche delle classi fittizie, il cui significato si può comprendere solo in un secondo momento. Queste classi fittizie si dichiarano nella sezione '**control**', con la direttiva '**addclasses**':

```
control:
    ...
    addclasses = ( classe_fittizia... )
    ...
```

L'esempio seguente crea due classi fittizie, denominate '**bianchi**' e '**rossi**':

```
    addclasses = ( bianchi rossi )
```

Avendo più chiaro in mente cosa possa essere una classe elementare, si può iniziare a descrivere la definizione di espressioni legate alle classi. Le espressioni in questione sono booleane, dal momento che le classi, di per sé, rappresentano degli insiemi di elaboratori. In questo senso, la logica booleana si intende correttamente come la logica degli insiemi. Gli operatori di queste espressioni sono elencati nella tabella u56.14.

Tabella u56.14. Operatori logici delle espressioni riferite alle classi di Cfengine.

Operatore	Descrizione
`( )`	Le parentesi tonde hanno la precedenza nella valutazione.
`!`	NOT, ovvero insieme complementare.
`.`	AND, ovvero intersezione.

Operatore	Descrizione
`|`	OR, ovvero unione.
`||`	Modo alternativo di indicare OR.

Per esempio, per indicare una classe complessiva che rappresenta indifferentemente un elaboratore con sistema operativo GNU/Linux o GNU/Hurd, si può usare l'espressione '**`linux|hurd`**'. In pratica, si scrive così:

```
    linux|hurd::
```

Per indicare una classe che rappresenti tutti gli elaboratori che non abbiano un sistema operativo GNU/Linux, si potrebbe usare l'espressione '**`!linux`**', ovvero:

```
    !linux::
```

A questo punto diventa più facile comprendere il senso delle classi fittizie che si possono dichiarare con la direttiva '**`addclasses`**'. Si osservi l'esempio seguente:

```
control:
    actionsequence = ( links )
    addclasses = ( primo )

links:
    any.primo::
        /var/tmp/altra -> /tmp
```

L'espressione '**`any.primo`**' si avvera solo quando la classe elementare '**`primo`**' è stata dichiarata come nell'esempio; infatti, '**`any`**' è sempre vera. In questo modo, anche se l'esempio non richiederebbe tanta raffinatezza, basterebbe controllare la dichiarazione della direttiva '**`addclasses`**' per abilitare o meno la classe sottostante. In altri

termini, è facile modificare un file di configurazione che richiama in più punti la classe fittizia '**primo**', modificando solo una riga di codice nella sezione '**control**'.

Il controllo sulla definizione di classi fittizie può avvenire anche al di fuori del file di configurazione attraverso le opzioni '**-D*classe_fittizia***' e '**-N*classe_fittizia***'. Nel primo caso, si ottiene la dichiarazione di una classe fittizia, mentre nel secondo si ottiene l'eliminazione di una classe già dichiarata nel file di configurazione. Per esempio, il comando seguente serve ad annullare l'effetto della dichiarazione della classe fittizia '**primo**', dell'esempio precedente.

```
$ cfengine -Nprimo -f prova.conf [Invio]
```

Variabili e stringhe

«

Cfengine gestisce le variabili di ambiente, oltre ad altre variabili, in modo simile a quanto fanno le shell. Queste variabili vengono espanse usando una delle due notazioni seguenti:

```
$(nome_variabile)
${nome_variabile}
```

Per la precisione, le variabili di Cfengine possono essere state ereditate dall'ambiente, possono essere state definite nella sezione '**control**', oppure possono essere variabili predefinite di Cfengine. L'esempio seguente mostra la dichiarazione della variabile '**percorso**' nella sezione '**control**':

```
control:
    actionsequence = ( tidy links )
    percorso = ( "/var/tmp" )

tidy:
    $(percorso) pattern=* age=30 recurse=inf

links:
    $(percorso)/altra -> /tmp
```

Si intuisce che potrebbe essere più interessante dichiarare la variabile in questione all'interno di classi diverse, in modo da aggiornare automaticamente il percorso di conseguenza. L'esempio seguente mostra due classi inventate, '**bianco**' e '**nero**', che non esistono in realtà:

```
control:
    actionsequence = ( tidy links )
    bianco::
        percorso = ( "/var/tmp" )
    nero::
        percorso = ( "/temp" )

tidy:
    $(percorso) pattern=* age=30 recurse=inf

links:
    $(percorso)/altra -> /tmp
```

Si può osservare in particolare che la direttiva '**actionsequence**', non appartenendo ad alcuna classe, viene presa sempre in considerazione.

Le variabili predefinite di Cfengine sono tali perché sono gestite automaticamente e servono a rendere disponibili delle informazioni, oppure perché servono a definire delle informazioni specifiche. In altri termini, le prime vanno solo lette, mentre le altre vanno impostate opportunamente se richiesto. La tabella u56.20 mostra le variabili destinate alla sola lettura, mentre la tabella u56.21 mostra le

variabili da impostare.

Tabella u56.20. Variabili interne di Cfengine, destinate alle sola lettura.

Variabile	Descrizione
`allclasses`	Elenca le classi attive.
`arch`	Architettura in modo dettagliato.
`binserver`	Servente NFS predefinito per dati binari.
`class`	Classe essenziale riferita al sistema operativo.
`date`	La data attuale.
`fqhost`	Il nome a dominio completo.
`ipaddress`	Un indirizzo IP significativo dell'elaboratore.
`year`	L'anno attuale.

Per quanto riguarda la variabile '**`domain`**', se questa non viene impostata espressamente, occorre considerare che potrebbe trattarsi del dominio che compone il nome dell'elaboratore, ovvero ciò che si legge e si imposta con il comando '**`hostname`**' dei sistemi Unix. In pratica, se il nome dell'elaboratore è stato impostato senza l'aggiunta del dominio di appartenenza, questa variabile restituisce probabilmente la stringa '**`undefined.domain`**'. Lo stesso discorso vale per la variabile '**`fqhost`**': se non si dispone del dominio finale nel nome restituito da '**`hostname`**', si ottiene una cosa simile a '***nome*`.undefined.domain`**'.

Tabella u56.21. Variabili interne di Cfengine, modificabili da parte dell'utilizzatore.

Variabile	Descrizione
`domain`	Il dominio, senza il nome iniziale dell'elaboratore.

`faculty`
 	
`site`
	Nome utilizzabile per definire il luogo.

`maxcfengines`	Numero massimo di processi Cfengine concorrenti.
`repchar`	Carattere usato in sostituzione di '`/`' nei nomi di file.
`split`	Carattere usato per separare gli elenchi nelle variabili.
`sysadm`	Amministratore (nome o indirizzo di posta elettronica).
`checksumdatabase`	File destinato alla raccolta dei codici di controllo.

In generale, i nomi delle variabili sono distinti anche in base all'uso di maiuscole e minuscole; tuttavia, le variabili predefinite possono essere usate con qualunque combinazione di lettere maiuscole e minuscole.

Esiste anche un altro gruppo di variabili speciali, in sola lettura, definite per facilitare l'inserimento di caratteri speciali all'interno di stringhe, quando non è possibile fare altrimenti. Queste variabili sono elencate nella tabella u56.22.

Tabella u56.22. Variabili interne per la rappresentazione di caratteri speciali.

Variabile	Descrizione
`cr`	Ritorno a carrello: <*CR*>.
`dblquote`	Apici doppi: ‘`"`’.
`dollar`	Dollaro: ‘`$`’.
`lf`	Avanzamento di riga: <*LF*>.
`n`	Codice di interruzione di riga secondo l’architettura.
`quote`	Apice singolo: ‘`'`’.
`space`	Spazio singolo: <*SP*>.
`tab`	Tabulazione: <*TAB*>.

Le stringhe sono delimitate indifferentemente attraverso apici doppi e singoli, potendo usare anche gli apici singoli inversi. In pratica, si possono usare le forme seguenti:

```
"stringa"
'stringa'
`stringa`
```

Il significato è lo stesso e l’espansione delle variabili avviene in tutti i casi nello stesso modo. Disponendo di diversi tipi di delimitatori, è più facile includere questi simboli nelle stringhe stesse. In

questo senso va considerato il fatto che non esistono sequenze di escape; al massimo si possono usare le variabili predefinite per la rappresentazione di caratteri particolari.

Le stringhe sono utilizzabili solo in contesti particolari, precisamente la definizione di valori da assegnare a una variabile dichiarata nella sezione '**control**' e i comandi di shell nella sezione '**shellcommands**' (che non è ancora stata mostrata).

Elenchi

Le variabili possono essere intese come contenenti un elenco di sottostringhe. In questi casi, la loro espansione può richiedere una valutazione ulteriore. Tutto ha inizio dalla variabile interna '**split**', che normalmente contiene il carattere ':'. In questo senso, si osservi l'esempio seguente:

```
control:
    actionsequence = ( tidy )
    elenco = ( "primo:secondo:terzo" )

tidy:
    /var/tmp/$(elenco) pattern=* age=0
```

Assegnando alla variabile '**elenco**' la stringa '**primo:secondo:terzo**', si ottiene l'indicazione di un elenco di tre sottostringhe: '**primo**', '**secondo**' e '**terzo**'. A questo punto, la direttiva contenuta nella sezione '**tidy**', si traduce nella cancellazione dei file '`/var/tmp/primo`', '`/var/tmp/secondo`' e '`/var/tmp/terzo`'. Volendo cambiare il simbolo di separazione delle sottostringhe si agisce nella variabile '**split**', come si vede nell'esempio seguente, che ottiene lo stesso risultato.

```
control:
    actionsequence = ( tidy )
    split = ( " " )
    elenco = ( "primo secondo terzo" )

tidy:
    /var/tmp/$(elenco) pattern=* age=0
```

Naturalmente, si può ottenere l'espansione di variabili del genere solo nei contesti in cui questo può avere significato.

Espressioni regolari

«

In contesti ben determinati, si possono indicare delle espressioni regolari. Cfengine utilizza le espressioni regolari ERE secondo le convenzioni GNU. Sono disponibili gli operatori riassunti nella tabella u56.25.

Tabella u56.25. Elenco degli operatori delle espressioni regolari.

Operatore	Descrizione
`\`	Protegge il carattere seguente da un'interpretazione diversa da quella letterale.
`^`	Ancora dell'inizio di una stringa.
`.`	Corrisponde a un carattere qualunque.
`$`	Ancora della fine di una stringa.
`|`	Indica due possibilità alternative alla sua sinistra e alla sua destra.
`( )`	Definiscono un raggruppamento.
`[ ]`	Definiscono un'espressione tra parentesi quadre.

Operatore	Descrizione
`[`*xy*`...]`	Un elenco di caratteri alternativi.
`[`*x*`-`*y*`]`	Un intervallo di caratteri alternativi.
`[^...]`	I caratteri che non appartengono all'insieme.
x`*`	Nessuna o più volte *x*. Equivalente a '*x*`{0,}`'.
x`?`	Nessuna o al massimo una volta *x*. Equivalente a '*x*`{0,1}`'.
x`+`	Una o più volte *x*. Equivalente a '*x*`{1,}`'.
x`{`*n*`}`	Esattamente *n* volte *x*.
x`{`*n*`,}`	Almeno *n* volte *x*.
x`{`*n*`,`*m*`}`	Da *n* a *m* volte *x*.
`\b`	La stringa nulla all'inizio o alla fine di una parola.
`\B`	La stringa nulla interna a una parola.
`\<`	La stringa nulla all'inizio di una parola.
`\>`	La stringa nulla alla fine di una parola.
`\w`	Un carattere di una parola, praticamente '`[[:alnum:]_]`'.
`\W`	L'opposto di '`\w`', praticamente '`[^[:alnum:]_]`'.

Le espressioni regolari GNU includono anche le classi di carat-

teri (nella forma ‘**[:*nome*:]**’, come prescrive lo standard POSIX, mentre mancano i simboli di collazione e le classi di equivalenza..

[1] **CFengine** GNU GPL

Cfengine: sezioni di uso comune

Permessi e proprietà 772
Sezione «control» 774
Impostazione delle variabili 775
Direttiva «actionsequence» 775
Direttiva «addclasses» 775
Sezione «classes» o «groups» 776
Sezione «copy» 777
Opzione «dest» 778
Opzione «recurse» 779
Opzione «type» 779
Opzione «purge» 781
Opzione «server» 781
Sezione «directories» 782
Sezione «disable» 783
Opzione «rotate» 783
Sezione «files» 784
Opzione «checksum» 785
Opzione «recurse» 785
Sezione «links» 786
Opzione «type» 787
Opzione «recurse» 788

Sezione «processes» 788
Opzione «signal» 789
Opzione «restart» 789
Opzione «matches» 790
Opzione «action» 791
Sezione «shellcommands» 791
Sezione «tidy» 791
Opzione «pattern» 792
Opzione «recurse» 792
Opzione «age» 792
Opzione «type» 793
Opzione «rmdirs» 794

Una volta compresa a grandi linee l'impostazione della configurazione di Cfengine, bisogna entrare nell'analisi specifica di ogni sezione che si voglia prendere in considerazione, dal momento che ognuna può avere le sue caratteristiche e le sue direttive specifiche. In questo capitolo si descrivono solo alcune sezioni tipiche, in modo superficiale, allo scopo di consentire un utilizzo elementare di Cfengine.

Si osservi che in generale non conta l'ordine in cui sono indicate le sezioni e le direttive all'interno delle sezioni; inoltre, le direttive possono utilizzare più righe senza bisogno di simboli di continuazione.

In più sezioni differenti si usano delle direttive che contengono opzioni con lo stesso nome e con lo stesso significato. Si tratta in particolare di quelle opzioni che definiscono le caratteristiche dei permessi e delle proprietà di file e directory. È il caso di mostrare queste opzioni una volta sola per tutte:

```
mode=modalità
```

```
owner=utente_proprietario
```

```
group=gruppo_proprietario
```

La modalità è un numero ottale oppure una stringa di permessi. La stringa di permessi può essere espressa come avviene con il comando '**chmod**'. Si osservino gli esempi seguenti.

Opzione	Descrizione
`mode=0775`	Imposta la modalità 0775_8 in modo preciso.
`mode=u+rwx`	Assegna sicuramente all'utente proprietario i permessi di lettura, scrittura ed esecuzione (o attraversamento).
`mode=o-rwx`	Toglie agli utenti diversi dall'utente proprietario e dal gruppo proprietario qualunque permesso di accesso.

Opzione	Descrizione
`user=root` `group=root`	Stabilisce che l'utente e il gruppo proprietario deve essere '**`root`**', ammesso che Cfengine stia funzionando con i privilegi necessari per poter modificare la proprietà di file e directory.

Sezione «control»

La sezione '**`control`**' è quella fondamentale di ogni configurazione di Cfengine, dal momento che è attraverso questa, assieme alla direttiva '**`actionsequence`**', che si stabilisce l'utilizzo e l'ordine delle altre sezioni. In generale, la sintassi specifica di questa sezione è la seguente:

```
control:
      [espressione_classe::]
            nome = ( valore... )
            ...
      ...
      ...
```

È essenziale che, nelle direttive di assegnamento tipiche di questa sezione, le parentesi tonde siano spaziate sia all'interno che all'esterno.

Impostazione delle variabili

In questa sezione si dichiarano le variabili e si impostano quelle predefinite che richiedono un intervento. L'esempio seguente definisce la variabile predefinita '**domain**':

```
control:
    ...
    domain = ( brot.dg )
    ...
```

Direttiva «actionsequence»

Al nome '**actionsequence**' viene assegnato l'elenco di nomi di sezioni e di altre azioni da eseguire, in base all'ordine in cui si trovano in questo elenco:

```
control:
    …
    actionsequence = ( azione_1  azione_2… )
    …
```

A livello di utilizzo elementare, si fa riferimento sempre solo a nomi di sezione, mentre sono previsti altri nomi che identificano azioni particolari che non fanno capo a una sezione.

Direttiva «addclasses»

La direttiva '**addclasses**' è utilizzata per creare delle classi fittizie aggiuntive. L'esempio seguente aggiunge le classi '**bianco**' e '**nero**':

```
control:
    ...
    addclasses = ( bianco nero )
    ...
```

Si possono aggiungere delle classi anche con l'opzione '**-D*nome***' e si possono eliminare delle classi con l'opzione '**-N*nome***'.

Sezione «classes» o «groups»

«

La sezione '**classes**', ovvero anche '**groups**', è un po' anomala nella logica di Cfengine, dal momento che non rappresenta un'azione vera e propria, ma la dichiarazione di un raggruppamento di classi. Intuitivamente si comprende che questa cosa dovrebbe essere compito della sezione '**control**'. In effetti, questa sezione viene presa in considerazione comunque e non va annotata nella direttiva '**actionsequence**' della sezione '**control**'. La sintassi della dichiarazione di una classe nell'ambito di questa sezione, può essere di tre tipi:

```
classes:|groups:

    ...
    gruppo_di_classi = ( classe_1 classe_2... )
    ...
```

```
classes:|groups:

    ...
    classe = ( +dominio_nis )
    ...
```

```
classes:|groups:
    ...
    classe = ( "comando_di_shell" )
    ...
```

Nel primo caso si crea una classe che riproduce la somma di quelle indicate tra parentesi; nel secondo si ha una classe che rappresenta l'insieme degli elaboratori appartenenti al dominio NIS indicato; nel terzo si ottiene una classe se il comando indicato (delimitato tra virgolette) termina con successo, ovvero restituisce *Vero*.

Sezione «copy»

La sezione '**copy**' serve a copiare file nell'ambito dello stesso file system, oppure tra elaboratori differenti, attraverso il demone '**cfd**'. La copia viene fatta preparando prima un file con estensione '`.cfnew`', che alla fine viene rinominato nel modo previsto. Questa accortezza serve nella copia tra elaboratori, per evitare il danneggiamento dei file nel caso di interruzione della comunicazione nella rete. Salvo diversa indicazione, quando viene rimpiazzato un file attraverso la copia, quello vecchio viene conservato temporaneamente aggiungendogli l'estensione '`.cfsaved`'.

```
copy:
    [espressione_classe::]
        origine dest=destinazione [altre_opzioni]
        ...
    ...
    ...
```

Il contenuto della sezione '**copy**', può essere ovviamente suddiviso in classi, se ciò è utile. Alla fine, le direttive che possono essere contenute sono di un tipo solo, dove la prima informazione indica il nome del file, o il modello di file da copiare, mentre il resto sono delle opzioni nella forma '***nome=valore***'. Le opzioni di queste direttive sono numerose; qui ne vengono descritte solo alcune.

Opzione «dest»

«

```
dest=destinazione
```

Con questa opzione si definisce la destinazione della copia. Deve trattarsi di un oggetto dello stesso tipo dell'origine: se l'origine è un file normale, la destinazione deve essere un file normale; se l'origine è un collegamento simbolico la destinazione si riferisce a un collegamento simbolico; se l'origine è una directory, la destinazione deve essere una directory, in cui vengono copiati tutti i file che si trovano in quella originale (senza riprodurre le sottodirectory eventuali).

```
copy:
    /etc/passwd
        dest=/home/tizio/users
```

L'esempio mostra una situazione molto semplice, dove si vuole copiare il file '`/etc/passwd`' nel file '`/home/tizio/users`', oppure si vuole mantenere aggiornata la copia.

Opzione «recurse»

«

```
recurse={n_livelli|inf}
```

La copia di una directory può avvenire anche ricorsivamente, attraverso le sue sottodirectory, specificando il livello di ricorsione con questa opzione. Si assegna un numero intero, oppure la parola chiave '**`inf`**'. Il numero rappresenta la quantità di livelli di ricorsione da considerare, mentre la parola '**`inf`**' richiede espressamente una ricorsione infinita.

```
copy:
    /etc
        dest=/home/tizio/copia_etc
        recurse=1
```

L'esempio mostra la copia del contenuto della directory '`/etc/`' nella directory '`/home/tizio/copia_etc/`'. Dal momento che la ricorsione è limitata a un solo livello, si ottengono solo i file e le sottodirectory vuote (nel senso che non viene copiato anche il loro contenuto).

Opzione «type»

«

```
type={ctime|mtime|checksum|sum|byte|binary}
```

L'opzione '**`type`**' definisce in che modo Cfengine può determinare se il file va copiato o meno. Normalmente si fa riferimento alla

data di «creazione», intesa come quella in cui vengono modificati i permessi o comunque viene cambiato inode, nel senso che solo se il file di origine ha una data più recente viene fatta la copia. Intuitivamente si comprende il senso delle parole chiave '**ctime**' e '**mtime**': la prima fa riferimento esplicito a questa data di creazione, mentre la seconda fa riferimento alla data di modifica. In alternativa, le parole chiave '**checksum**' o '**sum**' richiedono un controllo attraverso un codice di controllo (una firma MD5) per determinare se il file originale è diverso e se è richiesta la copia.

Si osservi che nella copia tra elaboratori distinti, è l'elaboratore di destinazione che genera la firma MD5 del suo file e la invia all'elaboratore di origine per il confronto. Pertanto è nell'elaboratore di origine che avviene la comparazione delle firme e in caso di diversità avviene la trasmissione del file di origine.

Le parole chiave '**byte**' e '**binary**' richiedono un confronto completo dei file byte per byte.

```
copy:
    /etc/passwd
        dest=/home/tizio/users
        type=checksum
```

L'esempio mostra il caso della copia del file '`/etc/passwd`' nel file '`/home/tizio/users`', verificando la necessità di aggiornare la copia attraverso un codice di controllo.

Opzione «purge»

«

```
purge={true|false}
```

L'opzione '**purge**', se attivata assegnando la parola chiave '**true**', abilita l'eliminazione dei file che nell'origine non sono più presenti.

```
copy:
    /etc
        dest=/home/tizio/copia_etc
        recurse=inf
        purge=true
```

L'esempio mostra la copia del contenuto della directory '`/etc/`' nella directory '`/home/tizio/copia_etc/`', con ricorsione infinita, specificando anche che nella destinazione devono essere eliminati i file e le directory che non sono più presenti nell'origine.

Opzione «server»

«

```
server=nodo_remoto
```

Questa opzione si usa quando si vuole copiare un file remoto; per la precisione, serve a specificare che l'origine si trova presso un altro elaboratore. Per riuscire a copiare attraverso elaboratori, è necessario che il nodo servente sia stato predisposto con il demone '**cfd**'; inoltre, è necessario specificare la variabile '**domain**' nella sezione di controllo ('**control**').

Sezione «directories»

«

```
directories:
      [espressione_classe : : ]
            directory   [opzioni ]
            ...
      ...
      ...
```

Le direttive della sezione ‘**directories**’ servono a richiedere la presenza di directory determinate, specificando eventualmente le caratteristiche necessarie. Se le directory in questione mancano, vengono create; se le caratteristiche non corrispondono, queste vengono modificate.

Le opzioni più importanti sono quelle che definiscono i permessi e la proprietà, come descritto nella sezione u0.1.

```
directories:
    /           mode=0755 owner=root group=root
    /etc        mode=0755 owner=root group=root
    /bin        mode=0755 owner=root group=root
    /dev        mode=0755 owner=root group=root
    /sbin       mode=0755 owner=root group=root
    /lib        mode=0755 owner=root group=root
    /usr        mode=0755 owner=root group=root
    /tmp        mode=1777 owner=root group=root
```

L’esempio mostra una serie di direttive della sezione ‘**directories**’ con lo scopo di salvaguardare la presenza, i permessi e la proprietà di alcune directory importanti.

Sezione «disable»

```
disable:
      [espressione_classe::]
            file   [opzioni]
            ...
      ...
      ...
```

La sezione ‘**disable**’ serve a elencare un gruppo di file che si vogliono «disabilitare». L’idea è che questi file non devono essere presenti, ma non si vogliono nemmeno cancellare. In pratica, se vengono trovati, si aggiunge loro l’estensione ‘`.cfdisabled`’.

```
disable:
    /etc/hosts.equiv
```

L’esempio mostra una situazione tipica, in cui si vuole evitare che esista il file ‘`/etc/hosts.equiv`’, pur lasciando la possibilità di verificare il suo contenuto originale.

Opzione «rotate»

```
rotate={n|empty}
```

Eccezionalmente, se si utilizza l’opzione ‘**rotate**’, si fa riferimento implicitamente a file di registrazioni (*log*), che conviene spezzare periodicamente. Assegnando un numero intero all’opzione, si specifica la quantità di livelli da conservare. Per esempio, assegnando il valore ‘**2**’, si fa in modo che il file venga rinominato aggiungendo

l'estensione '`.1`', mentre un eventuale file preesistente con lo stesso nome verrebbe rinominato sostituendo l'estensione '`.1`' con '`.2`'.

Se si assegna la parola chiave '**`empty`**', non si salvano le versioni precedenti, annullando semplicemente il contenuto del file.

```
disable:
    /var/log/wtmp rotate=7
```

L'esempio mostra la richiesta di mettere da parte il file '`/var/log/wtmp`', in modo da ricominciare con un file vuoto, mantenendo sette copie precedenti, da '`/var/log/wtmp.1`' a '`/var/log/wtmp.7`'.

Sezione «files»

«

```
files:
    [espressione_classe : :]
        file  [opzioni]
        ...
    ...
    ...
```

Le direttive della sezione '**`files`**' servono a richiedere la presenza di file determinati, specificando eventualmente le caratteristiche necessarie. Se i file in questione mancano, vengono creati (vuoti); se le caratteristiche non corrispondono, queste vengono modificate per quanto possibile.

Le opzioni più importanti sono quelle che definiscono i permessi e la proprietà, come descritto nella sezione u0.1. Tuttavia è importante anche la possibilità di controllare che i file in questione non siano stati modificati.

Opzione «checksum»

«

```
checksum=md5
```

L'opzione '**checksum**' (a cui può essere assegnato solo il valore '**md5**') consente di richiedere la verifica dei file attraverso un codice di controllo. Inizialmente, questo codice di controllo deve essere accumulato da qualche parte e precisamente si tratta del file dichiarato nella variabile '**checksumdatabase**' nella sezione di controllo ('**control**').

Opzione «recurse»

«

```
recurse={n_livelli|inf}
```

Anche se si sta facendo riferimento principalmente a file, è consentito indicare directory intere, specificando il livello di ricorsione attraverso l'opzione '**recurse**', già descritta in precedenza.

La sezione '**files**' è orientata ai file. Tuttavia, se si richiede l'impostazione di permessi specifici, questi potrebbero interferire con quelli delle directory, nel momento in cui si fa riferimento a queste. Per risolvere il problema, Cfengine aggiunge i permessi di attraversamento necessari alle directory.

Sezione «links»

«

```
links:
     [espressione_classe : : ]
          collegamento  {- | +}> [ ! ]  oggetto_originale  [opzioni ]
          ...
     ...
     ...
```

La sezione ‘**links**’ serve per creare, aggiornare e sistemare dei collegamenti simbolici. In generale si distingue tra collegamenti singoli o collegamenti multipli. La differenza sta nell’uso dell’operatore ‘**->**’ oppure ‘**+>**’.

I collegamenti singoli riguardano un solo collegamento simbolico. Se il collegamento esiste già, viene verificato che corrisponda a quanto descritto nella direttiva, altrimenti si ottiene una segnalazione di errore.

```
/usr/local      -> /mia_dir/usr/local
```

L’esempio mostra una direttiva in cui si vuole che sia creato e mantenuto il collegamento simbolico ‘`/usr/local`’, che deve puntare alla directory reale ‘`/mia_dir/usr/local/`’.

Se il collegamento simbolico esiste già ma non corrisponde, oppure si tratta di un file, si può imporre la sua correzione con l’aggiunta del punto esclamativo:

```
/usr/local      ->! /mia_dir/usr/local
```

In tal caso, se ‘`/usr/local`’ fosse un file, il suo nome verrebbe modificato in ‘`/usr/local.cfsaved`’ e il collegamento potrebbe

così essere sistemato.

I collegamenti multipli si fanno indicando una directory di destinazione e una directory di origine, come nell'esempio seguente:

```
/usr/local/bin  +> /mia_dir/usr/local/bin
```

In questi casi si vogliono generare nella directory '`/usr/local/bin/`' tanti collegamenti simbolici quanti sono i file nella directory '`/mia_dir/usr/local/bin/`'.

Anche nel caso di collegamenti multipli si può usare il punto esclamativo per richiedere la correzione necessaria al completamento dell'operazione.

In generale, non vengono eliminati i collegamenti riferiti a file o directory non più esistenti. Per ottenere questo risultato, che potrebbe essere particolarmente utile in presenza di collegamenti multipli, occorre usare l'opzione '**`-L`**' nella riga di comando dell'eseguibile '**`cfengine`**'.

La creazione di un collegamento simbolico può richiedere la creazione delle directory che lo precedono. In condizioni normali ciò avviene automaticamente, senza bisogno di preoccuparsene.

Opzione «type»

«

```
type={hard|relative|absolute}
```

La sezione '**`links`**' è pensata fondamentalmente per gestire collegamenti simbolici. Tuttavia, con questa opzione è possibile richiedere

la creazione di collegamenti fisici, oltre alla possibilità di specificare il tipo di collegamento simbolico che si vuole ottenere: assoluto o relativo.

Indipendentemente dalle possibilità del sistema operativo, Cfengine non può creare dei collegamenti fisici che puntano a directory.

Opzione «recurse»

«

```
recurse={n_livelli|inf}
```

Dal momento che è consentita la generazione di collegamenti multipli, diventa opportuna la possibilità di specificare il livello di ricorsione attraverso l'opzione '**recurse**', già descritta in precedenza.

Sezione «processes»

«

```
processes:
      [espressione_classe::]
            "espressione_regolare"   [opzioni]
            ...
      ...
      ...
```

La sezione '**processes**' serve a individuare dei processi in funzione, attraverso un'analisi di quanto restituito dal comando '**ps**' del

sistema operativo. Lo scopo può essere di due tipi: inviare un segnale al processo o ai processi individuati, oppure eseguire un comando per riavviarli se questi risultano mancanti.

Si deve osservare che ogni direttiva individua uno o più processi in base a un'espressione regolare, delimitata tra virgolette. In tal modo si può fare riferimento a tutto ciò che appare nel rapporto generato dal comando '**ps**', non soltanto il nome del processo. Tuttavia, ciò significa anche che occorre predisporre bene queste espressioni regolari, per non incorrere in errori.

Opzione «signal»

«

```
signal=nome_del_segnale
```

attraverso l'opzione '**signal**' si richiede espressamente l'invio del segnale specificato. In mancanza di questa, non si invia alcun segnale. Il nome da assegnare dipende dal sistema operativo utilizzato, anche se in generale si tratta di nomi abbastanza standardizzati.

```
processes:
    "inetd" signal=hup
```

L'esempio mostra il caso in cui si cerchi il processo individuato dalla stringa '**inetd**', per inviargli il segnale SIGHUP.

Opzione «restart»

«

```
restart "comando_di_shell"
```

Se non si trova una corrispondenza con l'espressione regolare indicata, si può ottenere l'avvio di un comando che presumibilmente

serve ad avviare il processo relativo. Per questo si utilizza l'opzione '**`restart`**' che, come si vede dal modello sintattico, **non utilizza** il simbolo '**=**' per l'assegnamento.

Opzione «matches»

«

```
matches=[>|<]n
```

È possibile individuare una quantità di processi che corrispondono all'espressione regolare di partenza, definendo la quantità attesa. Se si usano gli operatori '**<**' e '**>**', ci si aspettano più di ***n*** processi, o meno di ***n*** processi, perché la condizione si avveri. Diversamente si attendono esattamente ***n*** processi.

Di solito, questa opzione si abbina soltanto alla richiesta di un avvertimento, attraverso l'opzione '**`action=warn`**'.

```
processes:
    "telnetd" matches=<7 action=warn
```

Questo esempio mostra il caso in cui si voglia essere avvisati se si trovano sette o più processi corrispondenti alla stringa '**`telnetd`**'.

Può sembrare strana l'interpretazione dei simboli '**>**' e '**<**'. In realtà si deve vedre la cosa dal lato opposto: con '**>**' ci si aspetta che i processi siano meno della quantità indicata, perché non debba essere eseguita l'azione; nello stesso modo, con '**<**' ci si aspetta che i processi siano di più di quanto indicato perché l'azione non sia eseguita.

Opzione «action»

```
action={signal|do|warn}
```

L'opzione '**action**' stabilisce il da farsi, quando si verificano le condizioni richieste per intervenire. Le parole chiave '**signal**' o '**do**' richiedono espressamente l'invio del segnale stabilito con l'opzione '**signal**'; la parola chiave '**warn**' richiede solo una segnalazione di avvertimento.

Sezione «shellcommands»

```
shellcommands:
    [espressione_classe::]
        "comando"  [opzioni]
        ...
    ...
    ...
```

La sezione '**shellcommands**' serve a inserire dei comandi di shell, debitamente delimitati tra virgolette. Di solito, non si utilizzano le opzioni, anche se in situazioni particolari possono essere utili.

Sezione «tidy»

```
tidy:
    [espressione_classe::]
        directory pattern=modello [altre_opzioni]
        ...
    ...
    ...
```

La sezione '**tidy**' è fatta per eliminare dei file non desiderati. Come si può osservare dal modello sintattico, le direttive iniziano dalla definizione di una directory di partenza, per cui diventa necessario specificare i file da eliminare attraverso un modello con l'opzione '**pattern**'.

Opzione «pattern»

«

```
pattern=modello
```

Attraverso l'opzione '**pattern**' si specifica il file o i file da prendere in considerazione nella directory di partenza. Il modello si può realizzare utilizzando i soliti simboli speciali, '*****' e '**?**', con il significato consueto: qualunque stringa, oppure un solo carattere.

Opzione «recurse»

«

```
recurse={n_livelli|inf}
```

È consentita la cancellazione di file anche attraverso le sottodirectory, utilizzando l'opzione '**recurse**', come già è stato mostrato in precedenza.

Opzione «age»

```
age=n_giorni
```

L'opzione '**age**' consente di specificare quanto tempo devono avere i file per poter essere cancellati. Se il tempo è stato raggiunto o superato, si ottiene la cancellazione. Questo tempo si riferisce in modo predefinito alla data di accesso, ma può essere cambiato con l'opzione '**type**'.

```
tidy:
    /tmp      pattern=*  recurse=inf  age=1
    /var/tmp  pattern=*  recurse=inf  age=7
```

L'esempio mostra un caso molto semplice, in cui si vuole ripulire il contenuto delle directory '`/tmp/`' e '`/var/tmp/`', per i file che sono vecchi rispettivamente un giorno e sette giorni.

Opzione «type»

```
type=[ctime|mtime|atime]
```

La vecchiaia di un file può essere valutata in base alla data di «creazione», intesa come cambiamento di inode, di modifica o di accesso, assegnando rispettivamente le parole chiave '**ctime**', '**mtime**' o '**atime**' all'opzione '**type**'. Questo serve per stabilire il modo corretto di interpretazione del valore assegnato all'opzione '**age**'.

Opzione «rmdirs»

«

```
rmdirs=[true|false|all|sub]
```

In condizioni normali, non si ottiene la cancellazione delle directory. Per questo, occorre usare l'opzione '**rmdirs**', a cui si assegnano le parole chiave che si vedono nel modello sintattico. In condizioni normali, è come se fosse assegnata la parola chiave '**false**', che impedisce la cancellazione. Se si richiede la cancellazione, si elimina anche la directory di partenza, corrispondente al modello richiesto. Al contrario, assegnando la parola chiave '**sub**', si preserva la directory di partenza.

Cfengine attraverso la rete

Configurazione e avvio del demone 795
Configurazione essenziale di «cfd.conf» 797
Filosofia del sistema di distribuzione di Cfengine 800

Cfengine consente anche un utilizzo attraverso la rete, per mezzo del demone '**cfd**', che viene avviato nell'elaboratore che offre il proprio servizio.

L'utilizzo più semplice di questa possibilità di Cfengine sta nella copia di file attraverso la rete, per sincronizzare gli elaboratori clienti. Per la precisione, questo particolare è l'unica cosa che viene mostrata qui, in questo capitolo.

Configurazione e avvio del demone

Il servizio relativo al demone '**cfd**' prevede l'accesso alla porta TCP 5308, che pertanto non è privilegiata e consente l'avvio del demone anche senza i privilegi dell'utente '**root**', se questo può essere utile per qualche motivo. Nel file '`/etc/services`' dovrebbe esserci pertanto una riga simile a quella seguente:

```
cfengine        5308/tcp
```

Per funzionare, il demone '**cfd**' richiede la presenza del file '`cfd.conf`', nella directory corrente nel momento dell'avvio del demone, che ha una struttura simile a quella di '`cfengine.conf`'.

Oltre a questo file essenziale, occorre tenere presente che il demone tiene in considerazione anche il contenuto dei file '`/etc/hosts.allow`' e '`/etc/hosts.deny`', per controllare gli accessi.

Una volta predisposto il sistema di configurazione, basta avviare il demone '**`cfd`**', con i privilegi dell'utente '**`root`**', se necessario, oppure con i privilegi di un utente comune.

```
# cfd[Invio]
```

Alcune opzioni del demone '**`cfd`**' sono molto utili per consentire l'analisi del file di configurazione e per poter tenere sotto controllo ciò che avviene effettivamente durante la connessione. Queste opzioni sono riepilogate nella tabella u58.2.

Tabella u58.2. Elenco delle opzioni essenziali di '**`cfd`**'.

Opzione	Descrizione
`-h` `--help`	Elenca brevemente le opzioni disponibili.
`-d` `--debug`	Rimane in primo piano e mostra ciò che accade.
`-v` `--verbose`	Mostra informazioni dettagliate.
`-p` `--parse-only`	Si limita a scandire il file di configurazione.

Può essere interessante il controllo della configurazione attraverso l'opzione '**-p**', unita opportunamente all'opzione '**-v**'. Inoltre, per verificare le connessioni, soprattutto alla ricerca delle motivazioni per cui qualcosa non funziona come si vorrebbe, conviene utilizzare l'opzione '**-d**', sempre in combinazione con '**-v**'.

```
# cfd -d -v[Invio]
```

Configurazione essenziale di «cfd.conf»

Il file '`cfd.conf`' ha una vaga somiglianza con il file di configurazione di un cliente normale di Cfengine. In particolare, ci sono le sezioni e possono essere presenti le classi, solo che hanno valore esclusivamente nei confronti dell'elaboratore in cui si trova a funzionare il demone.

Generalmente, è probabile che non si faccia uso di classi in un file '`cfg.conf`' e qui non si mostrano esempi in tal senso. La sintassi semplificata ed essenziale di questo file, viene mostrata dal modello seguente. Si tenga presente che non vengono mostrate tutte le direttive, ma solo quelle che devono essere conosciute necessariamente.

```
control:
     [domain = ( dominio )]
     [maxconnections = ( numero_massimo_di_connessioni_indipendenti )
]
     [allowconnectionsfrom = ( numero_ip [numero_ip]... )]
     [denyconnectionsfrom = ( numero_ip [numero_ip]... )]
     [allowmultipleconnectionsfrom = ( numero_ip [numero_ip]
... )]
     [logallconnections = ( true|false )]

admit:|grant:
     file_o_directory          nodi_indicati_con_caratteri_jolly
     ...

deny:
     file_o_directory          nodi_indicati_con_caratteri_jolly
     ...
```

Si può osservare la presenza di una sezione di controllo, simile a quella dei clienti Cfengine. Questa sezione può anche risultare vuota.

Le sezioni '**admit**' (o '**grant**') e '**deny**', permettono di stabilire l'accessibilità di file e di directory, a degli elaboratori identificati per nome, anche in modo parziale attraverso caratteri jolly. Si intende che la sezione '**admit**' o '**grant**' serva a elencare i file e le directory accessibili, mentre la sezione '**deny**' serve a escludere successivamente parte di quanto precedentemente concesso.

Nella sezione di controllo, le direttive '**maxconnections**',

‘**allowconnectionsfrom**’, ‘**denyconnectionsfrom**’ e ‘**allowmultipleconnectionsfrom**’, limitano o concedono gli accessi attraverso l’indicazione di un elenco di indirizzi IP. In generale, questo può essere un mezzo ulteriore di controllo di sicurezza per gli accessi, dal momento che spesso è sufficiente l’uso delle sezioni ‘**admit**’ e ‘**deny**’. In particolare, ogni cliente Cfengine che accede, ha la possibilità di aprire una sola connessione, mentre con la direttiva ‘**allowmultipleconnectionsfrom**’ è possibile autorizzare un accesso multiplo agli indirizzi indicati.

L’uso delle altre direttive indicato dovrebbe essere intuitivo; inoltre, nella sezione di controllo è possibile dichiarare delle variabili, nello stesso modo della configurazione dei clienti Cfengine.

È importante ricordare che i percorsi di cui si concede l’accesso, devono essere reali, perché i collegamenti simbolici non vengono presi in considerazione. Questo tipo di errore lo si può individuare utilizzando l’opzione ‘**-d**’ quando si avvia ‘**cfd**’.

A titolo di esempio viene mostrato un caso molto semplice di configurazione, in cui si concede l’accesso alle directory ‘`/usr/local/file_pubblici1/`’ e ‘`/usr/local/file_pubblici2/`’, creando appositamente due variabili per semplificarne l’indicazione; inoltre si concede l’accesso anche ai file ‘`/etc/passwd`’ e ‘`/etc/group`’. Per la precisione, la directory ‘`/usr/local/file_pubblici1/`’ risulta accessibile a tutti, mentre ‘`/usr/local/file_pubblici2/`’ è accessibile solo ai domini **.brot.dg* e **.mehl.dg*; inoltre, i due file ‘`/etc/passwd`’ e ‘`/etc/group`’ sono accessibili esclusivamente dal dominio **.brot.dg*. Infine, per

qualche motivo, si esclude l'accesso alla directory '`/usr/local/file_pubblici2/particolare/`' al dominio **.mehl.dg*. Ogni cliente può aprire una sola connessione e sono consentiti un massimo di 10 accessi simultanei.

```
control:
    pubblici1 = ( /usr/local/file_pubblici1 )
    pubblici2 = ( /usr/local/file_pubblici2 )
    maxconnections = ( 10 )

admit:
    $(pubblici1) *
    $(pubblici2) *.brot.dg *.mehl.dg
    /etc/passwd  *.brot.dg
    /etc/group   *.brot.dg

deny:
    $(pubblici2)/particolare *.mehl.dg
```

Filosofia del sistema di distribuzione di Cfengine

«

È il caso di osservare che, contrariamente a Rsync, il cliente Cfengine contatta il servente per ottenere qualcosa e non per inviare lì un file.

Quando la trasmissione di un file è sottoposta al confronto di un codice di controllo, è il cliente Cfengine che invia il suo codice di controllo al servente, il quale verifica la necessità o meno di trasmettere il file aggiornato.