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.

Permessi e proprietà

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.
`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 del-

le 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.