Capitolo 1

Per iniziare

Questo libro vi insegnerà a realizzare sistemi eccezionali utilizzando la versione più recente di Hadoop. Prima di cambiare il mondo, però, vi serviranno un po' di basi. In queste pagine introduttive affronteremo quanto segue.

- Un breve ripasso su Hadoop.
- Una panoramica sull'evoluzione di Hadoop.
- Gli elementi chiave di Hadoop 2.
- Le distribuzioni Hadoop che utilizzeremo in questo libro.
- Il dataset che impiegheremo per gli esempi.

Una nota sulle versioni

In Hadoop 1, la storia delle versioni è piuttosto complessa, con più diramazioni nel range 0.2x che portano a situazioni insolite, in cui una versione 1.x potrebbe, in alcune situazioni, avere meno funzioni di una 0.23. Nella base di codice della versione 2 è tutto molto più diretto, per fortuna, ma è importante chiarire esattamente quale versione utilizzeremo in questo libro.

Hadoop 2.0 è stato rilasciato nelle versioni alfa e beta, e nel tempo sono state introdotte alcune modifiche non compatibili. In particolare, si è assistito a uno sforzo di stabilizzazione dell'API principale tra la versione beta e la release finale.

In questo capitolo

Hadoop 2.2.0 è stata la prima release *general availability* (GA) della base di codice di Hadoop 2, e le sue interfacce sono ormai dichiarate stabili e compatibili per il futuro. In questo libro utilizzeremo quindi la versione e le interfacce 2.2.
Sebbene gli stessi principi siano applicabili a una 2.0 beta, ci saranno delle incompatibilità con le API. La cosa è particolarmente importante perché MapReduce v2 è stata oggetto di backporting su Hadoop 1 da parte di molti produttori della versione, ma questi prodotti si basavano sulla beta e non sulle API GA. Se utilizzate uno di questi prodotti, noterete l'incompatibilità delle modifiche. Consigliamo di utilizzare una release basata su Hadoop 2.2 o versioni successive sia per lo sviluppo sia per la distribuzione di qualsiasi carico di lavoro di Hadoop 2.

Panoramica su Hadoop

In questo libro diamo per scontato che la maggior parte dei lettori abbia un minimo di familiarità con Hadoop, o almeno con i sistemi di elaborazione dei dati. Non spiegheremo quindi i dettagli del suo successo né affronteremo il tipo di problemi che aiuta a risolvere. Considerati però alcuni degli aspetti di Hadoop 2 e di altri prodotti che impiegheremo nei vari capitoli, è utile dare un'idea di come Hadoop rientra nel panorama tecnologico e quali sono le aree problematiche specifiche in cui può essere vantaggioso utilizzarlo.
Una volta, prima che il termine *big data* facesse la sua comparsa (quindi più o meno dieci anni fa), erano poche le possibilità di elaborare dataset nell'ordine dei terabyte e oltre. Alcuni database commerciali potevano essere scalati a questi livelli con dei setup hardware molto specifici e costosi, ma le competenze e gli investimenti necessari lo rendevano un'opzione praticabile solo per le organizzazioni più grandi. In alternativa, si poteva costruire un sistema personalizzato mirato al problema contingente, ma questo non eliminava gli inconvenienti legati alle competenze e ai costi, senza considerare i rischi insiti in ogni sistema all'avanguardia. D'altra parte, se un sistema era ben costruito, probabilmente si sarebbe adattato alla perfezione alle esigenze per cui era nato.
Alcune società di piccole e medie dimensioni si preoccupavano per quanto riguardava lo spazio, non solo perché le soluzioni non erano alla loro portata, ma anche perché in genere i loro volumi di dati non raggiungevano i requisiti richiesti da tali soluzioni. Con il crescere della capacità di generare grossi database, cresceva anche la necessità di elaborare i dati.
La diffusione di grandi quantità di dati, non più esclusiva di pochi, portò con sé l'esigenza di alcune modifiche rilevanti nell'architettura dei sistemi di elaborazione anche per le aziende più piccole. La prima modifica importante fu la riduzione dell'investimento anticipato di capitale sul sistema, quindi niente hardware di alto livello né costose licenze software. In precedenza, si sarebbero utilizzati hardware high-end in un numero relativamente piccolo di server e sistemi di storage molto grandi, ciascuno dei quali aveva approcci diversi per evitare i crash. Per quanto

impressionanti, questi sistemi erano costosissimi, e spostarsi verso un numero più esteso di server di livello più basso sarebbe stato il modo più rapido per ridurre drasticamente il costo dell'hardware di un nuovo sistema. Il passaggio a un hardware di base invece che a un'attrezzatura aziendale tradizionale avrebbe anche comportato una riduzione delle capacità in termini di recupero e tolleranza ai guasti, responsabilità che sarebbero passate al livello software. (Software più intelligente, hardware più sciocco.)

Google diede via al cambiamento che sarebbe poi diventato noto come Hadoop quando, nel 2003 e nel 2004, rilasciò due documenti accademici che descrivevano il *Google File System* (GFS) (`http://research.google.com/archive/gfs.html`) e MapReduce (`http://research.google.com/archive/mapreduce.html`). I due documenti fornivano una piattaforma per l'elaborazione dei dati su larga scala in un modo eccezionalmente efficiente. Google aveva seguito l'approccio "fai da te", ma invece di costruire qualcosa di mirato alla risoluzione di un problema specifico o di un determinato dataset, aveva creato una piattaforma sulla quale potevano essere implementate più applicazioni di elaborazione. In particolare, utilizzava numerosi server di base e costruiva il GFS e MapReduce in modo che presumessero che i guasti dell'hardware fossero comuni e quindi qualcosa con cui il software avrebbe avuto spesso a che fare.

Nello stesso tempo, Doug Cutting stava lavorando sul crawler web open source chiamato Nutch, e in particolare su alcuni elementi nel sistema che ricoprirono una notevole rilevanza nel momento in cui i documenti su GFS e MapReduce furono pubblicati. Doug aveva iniziato a lavorare sulle implementazioni open source delle idee di Google, e presto nacque Hadoop, inizialmente un progetto derivato di Lucene e poi un progetto di alto livello indipendente sotto l'egida dell'Apache Software Foundation.

Yahoo! assunse Doug Cutting nel 2006 e divenne rapidamente tra i primi sostenitori del progetto Hadoop. Oltre a pubblicizzare in tutto il mondo alcune delle più grandi distribuzioni Hadoop, Yahoo! consentì a Doug e ad altri ingegneri di contribuire ad Hadoop durante il periodo del loro impiego, per non parlare di tutti i miglioramenti e le estensioni ad Hadoop sviluppati internamente.

Componenti di Hadoop

Hadoop è costituito da una serie di progetti secondari, molti dei quali verranno affrontati nel corso del libro. Di base, Hadoop fornisce due servizi: lo storage e il calcolo. Un flusso di lavoro tipico di Hadoop implica il caricamento dei dati nell'*Hadoop Distributed File System* (HDFS) e la loro elaborazione tramite l'API MapReduce o i numerosi strumenti che si basano su MapReduce come framework di esecuzione.

Entrambi i livelli sono implementazioni dirette delle tecnologie GFS e MapReduce di Google.

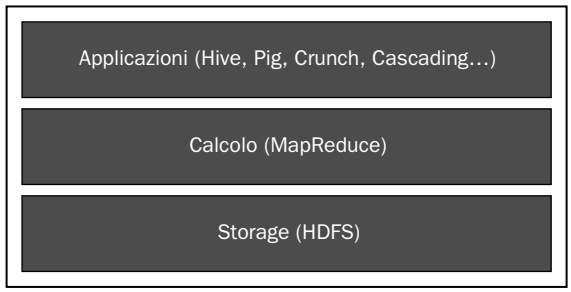


Figura 1.1 Hadoop 1: HDFS e MapReduce.

Componenti comuni

Sia HDFS sia MapReduce adottano molti dei principi architetturali descritti nel paragrafo precedente, e in particolare quelli che seguono.

- Entrambi sono concepiti per l'esecuzione su cluster di server di base (cioè con specifiche da medie a basse).
- Entrambi scalano la loro capacità aggiungendo altri server (*scale-out*) rispetto all'abitudine precedente di utilizzare un hardware più grande (*scale-up*).
- Entrambi hanno meccanismi per identificare e risolvere i problemi.
- Entrambi forniscono la maggior parte dei loro servizi in modo trasparente, consentendo all'utente di concentrarsi sul problema del momento.
- Entrambi hanno un'architettura in cui un cluster software risiede sui server fisici e gestisce aspetti come il bilanciamento del carico di un'applicazione e la tolleranza ai guasti, senza affidarsi all'hardware high-end per applicare queste capacità.

Storage

HDFS è un file system, sebbene non compatibile con POSIX. Questo significa che non ha le stesse caratteristiche di un file system ordinario ma altre peculiarità.

- Salva i file in blocchi di almeno 64 MB o, ancora più spesso, di 128 MB, dimensioni ben superiori ai 4-32 KB della maggior parte dei file system.
- È ottimizzato per il *throughput* a sfavore della latenza; è molto efficiente nella lettura in streaming di file molto grossi ma scadente quando si tratta di cercarne di piccoli.
- È ottimizzato per carichi di lavoro del tipo "scrivi una volta e leggi più volte".
- Invece di gestire i guasti del disco tramite le ridondanze fisiche nelle serie di dischi o strategie analoghe, HDFS utilizza la replica. Ciascuno dei blocchi che costituisce un file viene salvato su più nodi nel cluster, e il servizio chiamato NameNode li monitora costantemente per garantire che gli eventuali errori

o problemi non abbiano cancellato qualche blocco al di sotto del fattore di replica desiderato. Se accade, NameNode programma la creazione di un'altra copia all'interno del cluster.

Calcolo

MapReduce è un'API, un motore di esecuzione e un paradigma; rende possibile una serie di trasformazioni da una sorgente in un dataset di risultati. Nel caso più semplice, i dati di input vengono immessi tramite una funzione `map`, mentre i dati temporanei risultanti vengono forniti attraverso una funzione `reduce`.
MapReduce lavora al meglio su dati non strutturati o semi-strutturati. Non serve che i dati siano conformi a schemi rigidi; il requisito è che possano essere forniti alla funzione `map` come una serie di coppie chiave-valore. L'output della funzione `map` è un set di altre coppie chiave-valore, mentre la funzione `reduce` esegue l'aggregazione per assemblare il set finale di risultati.
Hadoop offre una specifica (cioè un'interfaccia) per le fasi di `map` e `reduce`, la cui implementazione è in genere chiamata *mapper* e *reducer*. Una tipica applicazione MapReduce comprenderà un certo numero di mapper e reducer, e non è insolito che diversi di questi siano molto semplici. Lo sviluppatore si focalizza sulla trasformazione tra i dati sorgente e i dati risultanti, mentre il framework di Hadoop gestisce tutti gli aspetti dell'esecuzione e del coordinamento del lavoro.

Meglio se insieme

HDFS e MapReduce possono essere utilizzati singolarmente, ma quando lavorano insieme fanno emergere il meglio l'uno dell'altro, e questa interrelazione è stato il fattore principale del successo e dell'adozione di Hadoop 1.
Quando si progetta un job di MapReduce, Hadoop deve decidere su quale host eseguire il codice per poter elaborare il dataset nel modo più efficiente. Non conta molto se gli host dei cluster di MapReduce traggono i loro dati da un unico host o array di storage, perché il sistema è una risorsa condivisa. Se il sistema di storage fosse più trasparente e consentisse a MapReduce di manipolare i suoi dati più direttamente, ci sarebbe l'opportunità di eseguire l'elaborazione dei dati più da vicino, in base al principio secondo cui è meno costoso spostare l'elaborazione che spostare i dati.
Il modello più comune di Hadoop vede la distribuzione dei cluster HDFS e MapReduce sullo stesso gruppo di server. Ogni host che contiene i dati e il componente HDFS che li gestisce ospita anche un componente MapReduce che può programmare ed eseguire l'elaborazione. Quando un job viene inviato ad Hadoop, questo può utilizzare l'ottimizzazione della posizione per programmare il più possibile i dati sugli host in cui risiedono i dati, riducendo così il traffico di rete e ottimizzando le prestazioni.

Hadoop 2: dov'è l'affare?

Se consideriamo i due componenti principali della distribuzione Hadoop, lo storage e il calcolo, vediamo che Hadoop 2 ha un impatto diverso su ciascuno di essi. Laddove l'HDFS in Hadoop 2 è un prodotto più ricco di funzionalità e più resiliente di quello in Hadoop 1, le modifiche per MapReduce sono più profonde, e hanno cambiato di fatto il modo in cui Hadoop viene percepito come piattaforma di elaborazione in generale. Vediamo prima HDFS in Hadoop 2.

Storage in Hadoop 2

Discuteremo l'architettura HDFS nel dettaglio nel Capitolo 2; per ora è sufficiente pensare a un modello master-slave. I nodi slave (i *DataNode*) contengono i dati veri e propri del file system. In particolare, ogni host che esegue un DataNode ha solitamente uno o più dischi su cui sono scritti i file che contengono i dati per ogni blocco HDFS. Il DataNode di per sé non sa nulla del file system globale; il suo ruolo è quello di memorizzare, servire ed assicurare l'integrità dei dati di cui è responsabile.

Il nodo master (il *NameNode*) deve sapere quale dei DataNode contiene un determinato blocco e come quei blocchi sono strutturati a formare il file system. Quando un client considera il file system per recuperare un file, è attraverso una richiesta al NameNode che ottiene l'elenco dei blocchi richiesti.

Questo modello funziona bene ed è stato scalato su cluster con decine di migliaia di nodi in realtà come quella di Yahoo!. Per quanto scalabile, tuttavia, c'è un rischio di resilienza; se il NameNode diventa non disponibile, allora l'intero cluster diventa inutile. Nessuna operazione HDFS può essere svolta, e poiché la maggioranza delle installazioni usa HDFS come livello di storage dei servizi, come MapReduce, anche questi diventano non disponibili, anche se sono in piena esecuzione senza problemi. Ancora peggio, il NameNode memorizza i metadati del file system in un file persistente sul suo file system locale. Se l'host del NameNode va in crash in un modo tale per cui i dati non sono recuperabili, allora tutti i dati sul cluster sono irrimediabilmente perduti. Continueranno a esistere sui vari DataNode, ma la mappatura di quali blocchi contenevano quali file non è più disponibile. Ecco perché in Hadoop 1 la best practice era quella che il NameNode scrivesse i dati del suo file system contemporaneamente sui dischi locali e almeno su un volume di rete remoto (solitamente tramite NFS).

Alcuni produttori di terze parti offrono diverse soluzioni *high-availability* (HA) di NameNode, ma il prodotto Hadoop centrale non fornisce questa resilienza nella Versione 1. Considerati il singolo punto di fallimento architetturale e il rischio di perdita dei dati, non sarà una sorpresa scoprire che la *NameNode HA* è una delle funzioni principali di HDFS in Hadoop 2, come vedremo nei prossimi capitoli. Questa caratteristica offre un NameNode in standby che può essere promosso au-

tomaticamente a soddisfare tutte le richieste qualora il NameNode attivo fallisse, e garantisce un'ulteriore resilienza per i dati critici del file system.
HDFS in Hadoop 2 è un file system non ancora compatibile con POSIX; ha una dimensione di blocchi molto grande e baratta ancora la latenza con il throughput. Tuttavia, ha ora alcune capacità che possono farlo assomigliare di più a un file system tradizionale. In particolare, l'HDFS core in Hadoop 2 può essere montato da remoto su un volume NFS, un'altra funzione che era prima offerta come proprietaria da fornitori di terzi parti ma che ora è parte integrante della base di codice principale di Apache.
Complessivamente, l'HDFS in Hadoop 2 è più resiliente e può essere integrato più facilmente nei processi e nei flussi di lavoro esistenti. È una grande evoluzione del prodotto che era in Hadoop 1.

Calcolo in Hadoop 2

Il lavoro su HDFS 2 è iniziato prima che la direzione di MapReduce fosse stabilita definitivamente. Questo soprattutto perché funzioni come la NameNode HA erano una strada talmente ovvia che la community conosceva già gli ambiti più critici da affrontare. Tuttavia, MapReduce non contemplava altrettante aree di miglioramento, ed ecco perché non fu subito chiaro lo scopo di un'iniziativa come quella di MRv2.
L'obiezione principale a MapReduce in Hadoop 1 riguardava il fatto che il suo modello di elaborazione in batch mal si adattava ai domini problematici in cui erano necessari tempi di risposta più rapidi. Hive, per esempio, che vedremo nel Capitolo 7, fornisce un'interfaccia del tipo SQL sui dati HDFS, ma dietro le quinte le istruzioni vengono convertite in job di MapReduce che vengono poi eseguiti come tutti gli altri. Altri prodotti o strumenti adottavano un approccio simile, fornendo un'interfaccia utente specifica che nascondeva il livello di traduzione di MapReduce.
Sebbene questo approccio ebbe successo, e nonostante la realizzazione di prodotti notevoli, rimane il fatto che il più delle volte c'è una discrepanza, poiché tutte queste interfacce, alcune delle quali si aspettano un certo tipo di reattività, dietro le quinte vengono eseguite su una piattaforma di elaborazione in batch. E tali discrepanze rimanevano anche se potevano essere apportati a MapReduce dei miglioramenti a favore di una corrispondenza più precisa. Questa situazione portò a un cambiamento significativo del focus dell'iniziativa MRv2. Forse non era MapReduce ad aver bisogno di modifiche; la vera necessità era quella di consentire diversi modelli di elaborazione sulla piattaforma Hadoop. Fu così che nacque *Yet Another Resource Negotiator* (YARN).
MapReduce in Hadoop 1 faceva due cose piuttosto diverse: forniva il framework di elaborazione per eseguire i calcoli di MapReduce, ma gestiva anche l'allocazione della computazione sul cluster. Non solo indirizzava i dati a e tra operazioni specifiche di `map` e `reduce`, ma determinava anche dove ogni attività sarebbe stata

eseguita, e gestiva l'intero ciclo di vita del job, monitorando la salute di ogni attività e nodo, riprogrammando in caso di fallimenti e così via.
Non è un'operazione banale, e la parallellizzazione automatizzata dei carichi di lavoro è sempre stato uno dei vantaggi di Hadoop. Se consideriamo MapReduce in Hadoop 1, dopo che l'utente definisce i criteri chiave per il job, qualsiasi altra cosa è di responsabilità del sistema. Da un punto di vista della scala, lo stesso job di MapReduce può essere applicato a dataset di qualsiasi volume sui cluster di qualsiasi dimensione. Se abbiamo 1 GB di dati su un unico host, allora Hadoop programmerà l'elaborazione di conseguenza, e farà lo stesso anche se abbiamo 1 PB di dati su mille macchine. Dal punto di vista dell'utente, la scala effettiva dei dati e dei cluster è trasparente, e al di là del tempo necessario a elaborare il job, l'interfaccia con cui si interagisce con il sistema non cambia.
In Hadoop 2, il ruolo di programmazione dei job e di gestione delle risorse è separato da quello dell'esecuzione dell'applicazione vera e propria, ed è svolto da YARN.
YARN è responsabile della gestione delle risorse del cluster, quindi MapReduce esiste in quanto applicazione che gira sul framework di YARN. In Hadoop 2 l'interfaccia di MapReduce è completamente compatibile con quella in Hadoop 1, sia semanticamente sia praticamente. Tuttavia, dietro le quinte, MapReduce è diventata un'applicazione ospitata sul framework YARN.
Il senso di questa discrepanza è che possono essere scritte altre applicazioni che forniscono modelli di elaborazione centrati sul problema contingente scaricando al contempo su YARN le responsabilità di gestione delle risorse e di programmazione. Le versioni più recenti di molti motori di esecuzione sono state portate su YARN, sia in uno stato pronto per la produzione sia sperimentale; tale approccio permette che un singolo cluster Hadoop esegua tutto, dai job di MapReduce orientati al batch attraverso query SQL a risposta rapida fino a stream di dati continui, oltre a implementare modelli come l'elaborazione dei grafici e la *Message Passing Interface* (MPI) del mondo dell'*High Performance Computing* (HPC). La Figura 1.2 mostra l'architettura di Hadoop 2.

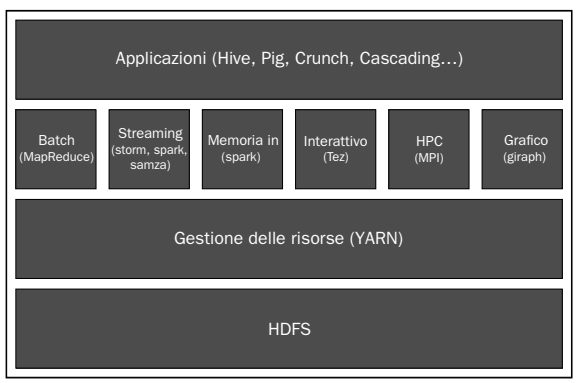


Figura 1.2 Hadoop 2.

Ecco perché gran parte dell'attenzione e dell'entusiasmo su Hadoop 2 si è concentrata su YARN e sui framework che vi risiedono, come Apache Tez e Apache Spark. Con YARN, il cluster Hadoop non è più solo un motore di elaborazione in batch; è una singola piattaforma sulla quale possono essere applicate varie tecniche di elaborazione alle enormi quantità di dati salvate in HDFS. Inoltre le applicazioni possono essere costruite su questi paradigmi computazionali e modelli di esecuzione. Si può pensare a YARN come al kernel di elaborazione su cui possono essere costruite applicazioni specifiche. Affronteremo YARN nel dettaglio nei Capitoli 3, 4 e 5.

Distribuzioni di Apache Hadoop

Agli albori di Hadoop, il peso dell'installazione (spesso dalla sorgente) e la gestione di ogni componente e delle sue dipendenze ricadevano sull'utente. Con la diffusione del sistema e dell'ecosistema degli strumenti e delle librerie di terze parti, la complessità dell'installazione e della gestione di una distribuzione Hadoop aumentò drasticamente, fino al punto che fornire un'offerta coerente di package software, documentazione e formazione attorno all'Apache Hadoop core è diventato un modello di business. Entriamo allora nel mondo delle distribuzioni per Apache Hadoop.

Le distribuzioni Hadoop sono concettualmente simili al modo in cui le distribuzioni Linux forniscono un set di software integrato attorno a un core comune. Si accollano il compito di assemblare e raccogliere il software e di fornire all'utente una modalità per installare, gestire e distribuire Apache Hadoop e un numero selezionato di librerie di terze parti. In particolare, le release forniscono una serie di versioni del prodotto che sono certificate come mutuamente compatibili. Storicamente, assemblare una piattaforma basata su Hadoop era un'operazione resa complessa dalle varie interdipendenze delle versioni.

Cloudera (`http://www.cloudera.com`), Hortonworks (`http://www.hortonworks.com`) e MapR (`http://www.mapr.com`) sono tra le prime ad aver raggiunto il mercato, ognuna con approcci e punti di vendita specifici. Hortonworks si posiziona come player open source; anche Cloudera è rivolta all'open source ma aggiunge elementi proprietari per la configurazione e la gestione di Hadoop; MapR fornisce una distribuzione Hadoop ibrida open source/proprietaria caratterizzata da un livello NFS proprietario invece che HDFS e un focus sulla fornitura di servizi.

Un altro player importante nell'ecosistema distribuito è Amazon, che offre una versione di Hadoop chiamata *Elastic MapReduce* (EMR) sull'infrastruttura *Amazon Web Services* (AWS).

Con l'avvento di Hadoop 2, il numero di distribuzioni disponibili per Hadoop è aumentato esponenzialmente, ben oltre le quattro che abbiamo citato. Un elenco non completo delle offerte software che include Apache Hadoop si trova all'indirizzo `http://bit.ly/1MnahAV`.

Un doppio approccio

In questo libro, tratteremo la costruzione e la gestione di cluster Hadoop locali e illustreremo come portare l'elaborazione sul cloud attraverso EMR.
La ragione è duplice: sebbene EMR renda Hadoop molto più accessibile, ci sono aspetti della tecnologia che diventano palesi solo con l'amministrazione manuale del cluster. Per quanto sia possibile utilizzare EMR in un modo più manuale, in genere per tali esplorazioni si utilizza un cluster locale. In secondo luogo, molte organizzazioni usano un insieme di capacità a metà tra l'in-house e il cloud, a volte per il timore di affidarsi a un unico provider esterno, anche se, in termini pratici, spesso è conveniente sviluppare e testare su piccola scala la capacità locale e poi distribuire il prodotto su vasta scala sul cloud.
In uno degli ultimi capitoli in cui vedremo altri prodotti che si integrano con Hadoop, mostreremo alcuni esempi di cluster locali e vedremo che, a prescindere da dove vengono distribuiti, non c'è differenza tra come i vari prodotti lavorano.

AWS: infrastruttura on demand di Amazon

AWS è un set di servizi di cloud computing offerto da Amazon. Nel libro ne utilizzeremo molti.

Simple Storage Service (S3)

Simple Storage Service (S3) di Amazon (`http://aws.amazon.com/s3/`) è un servizio di storage che fornisce un semplice modello di memorizzazione chiave-valore. Usando interfacce web, a riga di comando o di programma per creare oggetti – da file di testo, a immagini, a MP3 –, potete memorizzare e recuperare i dati in base a un modello gerarchico in cui create dei *bucket* che contengono gli oggetti. Ogni bucket ha un identificatore unico, e all'interno di ciascun bucket ogni oggetto ha un nome univoco. Questa strategia elementare abilita un servizio potentissimo di cui Amazon si assume la totale responsabilità (per scalare il servizio e per l'affidabilità e disponibilità dei dati).

Elastic MapReduce (EMR)

Elastic MapReduce di Amazon (`http://aws.amazon.com/elasticmapreduce/`) non è altro che Hadoop sul cloud. Usando una qualsiasi delle varie interfacce (console web, riga di comando o API), viene definito un flusso di lavoro Hadoop con attributi come il numero di host Hadoop richiesti e la posizione dei dati sorgente. Viene fornito il codice Hadoop che implementa i job di MapReduce, e viene premuto il pulsante virtuale *Vai*.

Nella sua modalità più potente, EMR può trarre i dati sorgente da S3, elaborarli su un cluster Hadoop che crea sul servizio di host virtuale on demand EC2 di Amazon, riportare i dati in S3 e terminare il cluster Hadoop e le macchine virtuali EC2 che lo ospitano. Ovviamente ognuno di questi servizi ha un costo (solitamente in base ai GB memorizzati e al tempo di utilizzo del server), ma la capacità di accedere a queste funzionalità così elevate di elaborazione dei dati senza che occorra un hardware dedicato non è da trascurare.

Come iniziare

Descriveremo ora i due ambienti che utilizzeremo nel libro. La macchina virtuale di QuickStart Cloudera sarà il nostro punto di riferimento su cui mostreremo tutti gli esempi; tuttavia, alcuni casi particolarmente interessanti che vale la pena eseguire su un servizio on demand li illustreremo su EMR di Amazon
Sebbene il codice e gli esempi forniti siamo il più possibile generici e portabili, quando si tratta di cluster locali, la nostra configurazione di riferimento sarà quella di Cloudera eseguita su CentOS Linux.
La maggior parte delle volte ci rifaremo a esempi che utilizzano o che vengono eseguiti dal prompt del terminale. Per quanto le interfacce grafiche di Hadoop siano molto migliorate negli anni (vedi per esempio, gli ottimi HUE e Cloudera Manager), quando si tratta di sviluppo, automazione e accesso programmatico al sistema, la riga di comando rimane ancora lo strumento più potente per lavorare.
Tutti gli esempi e il codice sorgente presentati in questo libro possono essere scaricati all'indirizzo `https://github.com/learninghadoop2/book-examples`. Inoltre è disponibile un piccolo sito web (in inglese) dedicato a questo libro dove trovare aggiornamenti e materiale correlato: l'indirizzo è `http://learninghadoop2.com`.

Cloudera QuickStart VM

Uno dei vantaggi delle distribuzioni Hadoop è che consentono l'accesso a package software facili da installare. Cloudera va anche oltre, e fornisce un'istanza di Virtual Machine scaricabile gratuitamente, nota come CDH QuickStart VM, distribuita su CentOS Linux.
Nel resto del libro utilizzeremo la CDH5.0.0 VM come riferimento e come sistema di base per eseguire gli esempi e il codice sorgente disponibile per i sistemi di virtualizzazione VMware (`http://www.vmware.com/nl/products/player/`), KVM (`http://www.linux-kvm.org/page/Main_Page`) e VirtualBox (`https://www.virtualbox.org/`).

Amazon EMR

Prima di utilizzare Elastic MapReduce, dobbiamo impostare un account AWS e registrarci per i servizi necessari.

Creare un account AWS

Amazon ha integrato i suoi account generali con AWS; se quindi avete già un account per uno qualsiasi dei siti di vendita online di Amazon, lo utilizzerete anche per i servizi AWS.

> **NOTA**
>
> I servizi AWS hanno un costo; dovrete quindi aver associata all'account una carta di credito attiva su cui possano essere effettuati gli addebiti.

Se richiedete un nuovo account Amazon, andate su `http://aws.amazon.com`, selezionate *Create a new AWS account* e seguite le istruzioni. Amazon ha aggiunto un livello gratuito (*Free Tier*) per alcuni servizi, quindi nei primi giorni di prova ed esplorazione le vostre attività rientreranno in questa versione. L'ambito del livello gratuito si sta ampliando, quindi verificate quello che volete, e non vi faranno pagare niente.

Sottoscrivere i servizi necessari

Una volta ottenuto un account Amazon, dovrete registrarlo per poterlo utilizzare con i servizi AWS necessari, cioè *Simple Storage Service* (S3), *Elastic Compute Cloud* (EC2) ed *Elastic MapReduce*. L'adesione è gratuita; la procedura serve solo per rendere disponibile i servizi al vostro account.
Aprite le pagine di S3, EC2 ed EMR da `http://aws.amazon.com`, fate clic sul pulsante *Sign up* in ogni pagina e seguite le istruzioni.

Utilizzare Elastic MapReduce

Una volta creato un account con AWS e dopo aver sottoscritto i servizi necessari, possiamo procedere a configurare l'accesso programmatico a EMR.

Rendere Hadoop operativo

> **ATTENZIONE**
>
> Costa soldi veri!

Prima di proseguire, è fondamentale tenere presente che l'uso dei servizi AWS implica il pagamento di una tariffa che avverrà addebitata sulla carta di credito associata all'account di Amazon. In genere le cifre sono basse, ma aumentano con l'aumentare dell'entità dell'infrastruttura consumata; lo storage di 10 GB di dati in S3 costa dieci volte più di 1 GB, ed eseguire 20 istanze di EC2 costa venti volte una sola istanza. Va poi considerato che i costi effettivi tendono a subire degli aumenti marginali più piccoli a livelli più elevati. In ogni caso, prima di utilizzare un servizio, leggete con attenzione le sezioni riguardanti i prezzi. Considerate anche che i dati che vengono trasferiti all'esterno dei servizi AWS, come C2 e S3,

sono addebitabili, mentre i trasferimenti tra servizi non lo sono. Questo significa che spesso è più conveniente progettare l'uso degli AWS in modo da mantenere i dati al loro interno per la maggior parte dell'elaborazione. Per informazioni su AWS ed EMR, consultate la pagina `http://aws.amazon.com/elasticmapreduce/#pricing`.

Come utilizzare EMR

Amazon fornisce interfacce sia web sia a riga di comando per EMR. Entrambi i tipi di interfaccia sono solo un front-end del sistema vero e proprio; un cluster creato da riga di comando può essere esplorato e gestito con gli strumenti web e viceversa. In genere utilizzeremo strumenti a riga di comando per creare e manipolare i cluster in modo programmatico, mentre torneremo all'interfaccia web quando ha senso farlo.

Credenziali AWS

Prima di utilizzare gli strumenti programmatici o a riga di comando, dovremo capire come il possessore di un account si autentica sugli AWS per le richieste.
Ogni account AWS ha numerosi identificatori, come quelli elencati di seguito, che vengono utilizzati quando si accede ai vari servizi.

- ID dell'account: ogni account AWS ha un ID numerico.
- Chiave di accesso: la chiave di accesso associata viene usata per identificare l'account che effettua la richiesta.
- Chiave di accesso segreta: fa il paio con la chiave di accesso. La chiave di accesso normale non è segreta e può essere esposta nelle richieste, mentre quella segreta è quella che utilizzate per validarvi come possessori dell'account. Trattatela con la stessa cura con cui trattate la vostra carta di credito.
- Coppie di chiavi: sono utilizzate per il login agli host EC2. È possibile generare coppie di chiavi pubbliche/private in EC2 o importare nel sistema le chiavi generate all'esterno.

Le credenziali e i permessi degli utenti sono gestiti tramite un servizio web chiamato *Identity and Access Management* (IAM), che dovrete sottoscrivere per poter ottenere le chiavi di accesso e segreta.
Sembra tutto un po' confuso, e lo è, almeno all'inizio. Quando si usa uno strumento per accedere a un servizio AWS, solitamente viene subito richiesto di aggiungere le credenziali corrette a un file configurato, dopodiché tutto funziona. Se però decidete di esplorare gli strumenti programmatici o a riga di comando, vale la pena investire un po' di tempo per leggere la documentazione relativa a ciascun servizio per capire come funziona sotto l'aspetto della sicurezza. Trovate ulteriori informazioni sulla creazione di un account AWS e sull'ottenimento delle credenziali di accesso alla pagina `http://docs.aws.amazon.com/iam`.

L'interfaccia AWS a riga di comando

Ogni servizio AWS ha da sempre il proprio set di strumenti a riga di comando. Tuttavia, di recente, Amazon ha creato un unico strumento unificato che consente l'accesso alla maggior parte dei servizi, l'Amazon CLI (*Command Line Interface*), che si trova all'indirizzo `http://aws.amazon.com/cli`.
Può essere installata da un tarball o tramite i package manager `pip` o `easy_install`. Sulla CDH QuickStart VM, possiamo installare `awscli` usando il seguente comando:

```
$ pip install awscli
```

Per accedere all'API, dobbiamo configurare il software per autenticarci per gli AWS usando le nostre chiavi di accesso e segreta.
È anche il momento giusto per impostare una copia di chiavi EC2 seguendo le istruzioni fornite all'indirizzo `https://console.aws.amazon.com/ec2/home?region=us-east-1#c=EC2&s=KeyPairs`. Per quanto una coppia di chiavi non sia strettamente necessaria per eseguire un cluster EMR, ci da là possibilità di effettuare un login da remoto al nodo master e di ottenere un accesso di basso livello al cluster.
Il prossimo comando ci guiderà attraverso una serie di passi di configurazione e poi di salvataggio della configurazione definitiva nel file `.aws/credential`:

```
$ aws configure
```

Una volta impostata la CLI, possiamo interrogare AWS con `aws <service> <arguments>`. Per creare e interrogare un bucket S3 usate un comando come quello che segue. Notate che i bucket S3 devono essere univoci tra tutti gli account AWS, quindi i nomi più comuni come `s3://mybucket` non saranno disponibili:

```
$ aws s3 mb s3://learninghadoop2
$ aws s3 ls
```

Possiamo dotare un cluster EMR di cinque nodi `m1.xlarge` usando i comandi seguenti:

```
$ aws emr create-cluster --name "EMR cluster" \
--ami-version 3.2.0 \
--instance-type m1.xlarge  \
--instance-count 5 \
--log-uri s3://learninghadoop2/emr-logs
```

Qui `--ami-version` è l'ID di un template Amazon Machine Image (`http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html`) e `--log-uri` dice a EMR di raccogliere i log e di memorizzarli nel bucket S3 `learninghadoop2`.

> **NOTA**
>
> Se non avete specificato una regione predefinita quando avete impostato l'AWS CLI, dovrete aggiungerne una alla maggior parte dei comandi EMR nella CLI usando l'argomento `--region`. Per esempio, `region eu-west-1` è relativo all'area dell'Irlanda nell'Unione Europea. Trovate dettagli sulle regioni AWS disponibili all'indirizzo `http://docs.aws.amazon.com/general/latest/gr/rande.html`.

Possiamo inviare i flussi di lavoro aggiungendo dei passi a un cluster in esecuzione tramite questo comando:

```
$ aws emr add-steps --cluster-id <cluster> --steps <steps>
```

Per terminare il cluster, usate questa riga di comando:

```
$ aws emr terminate-clusters --cluster-id <cluster>
```

Negli ultimi capitoli vi mostreremo come aggiungere dei passi per eseguire job di MapReduce e script Pig.
Trovate altre informazioni sull'AWS CLI alla pagina `http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-manage.html`.

Eseguire gli esempi

All'indirizzo `https://github.com/learninghadoop2/book-examples` trovate il codice di tutti gli esempi. Vengono forniti gli script e le configurazioni Gradle (`http://www.gradle.org/`) per compilare la maggior parte del codice Java. Lo script `gradlew` incluso nell'esempio caricherà Gradle e lo utilizzerà per recuperare le dipendenze e il codice di compilazione.
I file JAR possono essere creati invocando l'attività `jar` tramite uno script `gradlew`, così:

```
./gradlew jar
```

I job vengono solitamente eseguiti inviando un file JAR usando il comando `hadoop jar`:

```
$ hadoop jar example.jar <MainClass> [-libjars $LIBJARS] arg1 arg2 … argN
```

Il parametro facoltativo `-libjars` specifica le dipendenze di runtime di terze parti da inviare ai nodi remoti.

> **NOTA**
>
> Alcuni dei framework con cui lavoreremo, come Apache Spark, hanno strumenti propri di build e di gestione dei package. Informazioni e risorse specifiche verranno segnalate per questi casi particolari.

L'attività `copyJar` Gradle può essere usata per scaricare le dipendenze di terze parti in `build/libjars/<example>/lib`, come segue:

```
./gradlew copyJar
```

Per comodità, forniamo un'attività `fatJar` Gradle che accorpa le classi di esempio e le loro dipendenze in un unico file JAR. Sebbene questo approccio sia sconsigliato a favore dell'uso di `-libjar`, può diventare comodo quando si devono gestire le questioni legate alle dipendenze.

Il prossimo comando genera `build/libs/<example>-all.jar`:

```
$ ./gradlew fatJar
```

Elaborazione dei dati con Hadoop

Nei prossimi capitoli del libro presenteremo i componenti principali dell'ecosistema di Hadoop, oltre ad alcuni strumenti e librerie di terze parti che renderanno la scrittura di codice robusto e distribuito un'attività accessibile e divertente. Leggendo, imparerete a raccogliere, elaborare, memorizzare ed estrarre informazioni da grandi quantità di dati strutturati o meno.
Ci serviremo di un dataset generato dal firehose in tempo reale di Twitter (`http://www.twitter.com`). Questo approccio ci permetterà di fare qualche esperimento locale con dataset relativamente piccoli e, una volta pronti, di scalare gli esempi verso l'alto a un livello di produzione.

Perché Twitter?

Grazie alle sue API programmatiche, Twitter fornisce un modo semplice per generare dataset di dimensioni arbitrarie e per immetterli nei nostri cluster Hadoop locali o sul cloud. Il dataset che utilizzeremo avrà alcune proprietà che si adattano a numerosi casi di modellazione ed elaborazione dei dati.
I dati di Twitter hanno le seguenti proprietà.

- Non sono strutturati: ogni aggiornamento dello stato è un messaggio testuale che può contenere riferimenti a un contenuto multimediale, come URL e immagini.
- Sono anche strutturati: i tweet sono record consecutivi con una data e un'ora.
- Sono rappresentabili graficamente: le relazioni come le risposte e le menzioni possono essere modellate come una rete di interazioni.
- Sono gelolocalizzabili: si conosce la posizione da cui un tweet è stato inviato o dove un utente risiede.
- Sono in tempo reale: tutti i dati generati su Twitter sono disponibili attraverso un flusso in tempo reale (*firehose*).

Queste proprietà si rifletteranno nel tipo di applicazione che possiamo costruire con Hadoop, e includono esempi di sentiment e trend analysis e social network.

Creare il primo dataset

Le condizioni d'uso di Twitter vietano la ridistribuzione dei dati generati dall'utente in qualsiasi forma; per questa ragione, non possiamo rendere disponibile un dataset comune. Utilizzeremo allora uno script di Python per accedere in modo

programmatico alla piattaforma e creare un deposito di tweet degli utenti raccolti da uno stream live.

Un servizio, più API

Gli utenti di Twitter condividono oltre 200 milioni di tweet al giorno, noti anche come *aggiornamenti dello stato*. La piattaforma offre l'accesso a questo corpus di dati attraverso quattro tipi di API, ciascuna delle quali rappresenta una sfaccettatura di Twitter e mira a soddisfare casi d'uso specifici, come il collegamento e l'interazione con il contenuto di Twitter da fonti di terze parti (*Per prodotto*), l'accesso programmatico al contenuto di utenti o siti specifici (*REST*), le funzionalità di ricerca tra le timeline di utenti o siti (*Cerca*) e l'accesso a tutto il contenuto creato sulla rete di Twitter in tempo reale (*API Streaming*).
L'API Streaming consente un accesso diretto allo stream di Twitter, tenendo traccia delle parole chiave, recuperando i tweet geotaggati da una determinata regione e altro ancora. In questo libro useremo questa API come sorgente di dati per illustrare le capacità sia in batch sia in tempo reale da Hadoop. Tuttavia non interagiremo direttamente con essa; utilizzeremo invece librerie di terze parti per scaricarci di compiti come la gestione dell'autenticazione e delle connessioni.

Anatomia di un tweet

Ogni oggetto restituito da una chiamata all'API in tempo reale è rappresentato da una stringa JSON serializzata che contiene un set di attributi e metadati oltre al messaggio di testo.
Questo contenuto aggiuntivo include un ID numerico che identifica in modo univoco il tweet, la posizione da cui è stato condiviso, l'utente che l'ha condiviso (l'oggetto utente), se è stato ripubblicato da altri utenti (cioè se è stato ritiwittato) e quante volte (conteggio dei tweet), il linguaggio macchina del testo, se il tweet è stato inviato in risposta a qualcuno e, in questo caso, gli ID dell'utente e del tweet a cui è stato inviato e così via.
La struttura di un tweet e degli altri oggetti eventualmente esposti dall'API è in costante evoluzione. Trovate una guida aggiornata alla pagina `https://dev.twitter.com/docs/platform-objects/tweets`.

Credenziali di Twitter

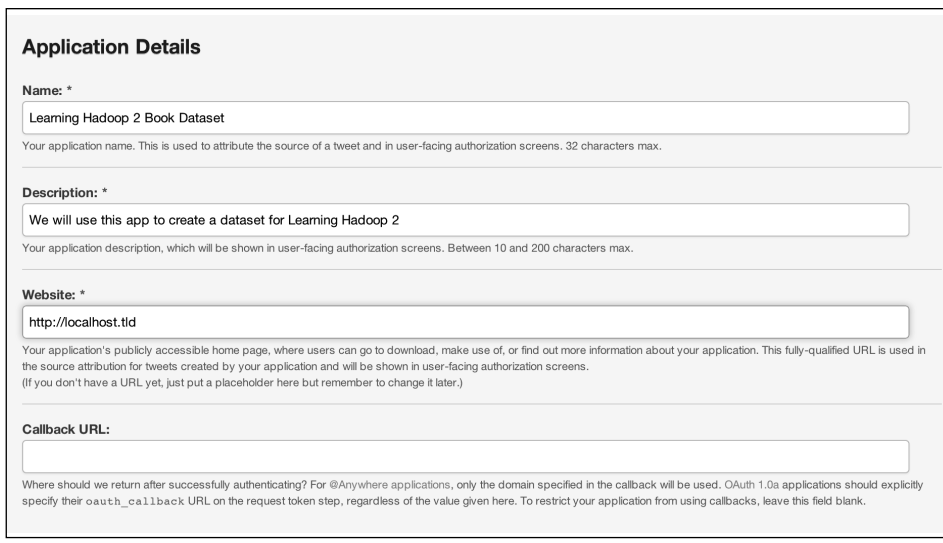
Twitter si serve del protocollo OAuth per autenticare e autorizzare l'accesso alla sua piattaforma da software di terze parti.
L'applicazione ottiene tramite un canale esterno, per esempio un form web, le seguenti coppie di codici:

- un *consumer key*;
- un *consumer secret*.

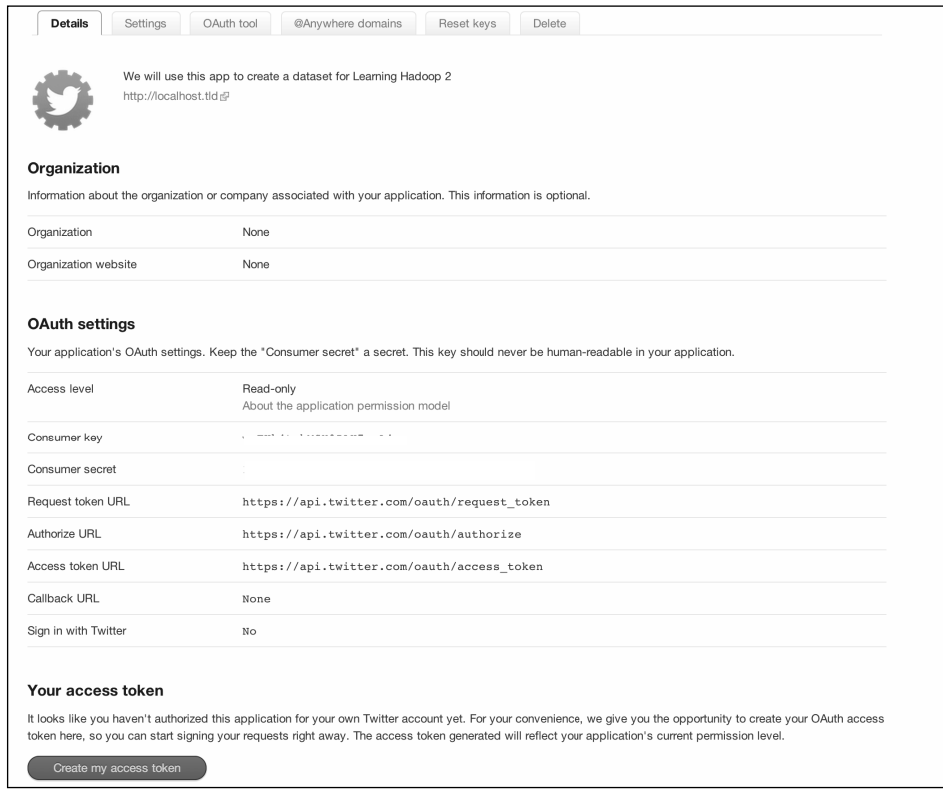
Il consumer secret non viene mai trasmesso direttamente alla terza parte perché viene usato per firmare ogni richiesta.
L'utente autorizza l'applicazione ad accedere al servizio tramite un processo a tre vie che, una volta completato, fornisce all'applicazione un token costituito da:

- un *access token*;
- un *access secret*.

Analogamente al codice consumer, l'access secret non viene mai trasmesso direttamente alla terza parte e viene usato per firmare ogni richiesta.
Per usare l'API Streaming, dovremo prima registrare un'applicazione e ottenere per essa l'accesso programmatico al sistema. Se richiedete un nuovo account Twitter, accedete alla pagina `https://twitter.com/signup` e inserite le informazioni richieste. A seguire, dovremo creare un'applicazione d'esempio che accederà all'API per nostro conto assegnandole i permessi opportuni. Per farlo ci serviremo del form web alla pagina `https://dev.twitter.com/apps`.
Quando si crea una nuova app, ci viene chiesto di darle un nome, di fornire una descrizione e un URL. La schermata che segue mostra le impostazioni di un'applicazione d'esempio chiamata `Learning Hadoop 2 Book Dataset`. Per gli scopi di questo libro non occorre specificare un URL valido, quindi utilizzeremo un segnaposto.



Una volta completato il form, controlliamo e accettiamo le condizioni d'uso e facciamo clic sul pulsante *Create Application* nell'angolo inferiore sinistro. Comparirà una pagina che riassume i dettagli della nostra applicazione, come mostrato nella figura. Le credenziali di autenticazione e di permesso si trovano nella scheda *OAuth Tool*.



Ed eccoci pronti a generare il nostro primo vero dataset di Twitter.

Accesso programmato con Python

In questo paragrafo utilizzeremo Python e la libreria `tweepy` (`https://github.com/tweepy/tweepy`) per raccogliere i dati di Twitter. Il file `stream.py` nella directory *ch1* dell'archivio di codice del libro istanzia un listener al firehose in tempo reale, cattura un campione di dati e ripete il testo di ciascun tweet nell'output standard. La libreria `tweepy` può essere installata usando sia i package manager `easy_install` o `pip` sia clonando il repository all'indirizzo `https://github.com/tweepy/tweepy`.
Sulla CDH QuickStart VM, possiamo installare `tweepy` con il seguente comando:

```
$ pip install tweepy
```

Quando è invocato con il parametro `-j`, lo script genera un tweet JSON nell'output standard; `-t` estrae e visualizza il campo di testo. Specifichiamo quanti tweet visualizzare con `-n <num tweets>`. Quando `-n` non è specificato, lo script verrà eseguito a tempo indeterminato. L'esecuzione può essere interrotta premendo Ctrl+C.
Lo script si aspetta che le credenziali OAuth vengano memorizzate come variabili d'ambiente della shell; le credenziali che seguono dovranno essere impostate nella sessione del terminale da cui `stream.py` verrà eseguito:

```
$ export TWITTER_CONSUMER_KEY="your_consumer_key"
$ export TWITTER_CONSUMER_SECRET="your_consumer_secret"
$ export TWITTER_ACCESS_KEY="your_access_key"
$ export TWITTER_ACCESS_SECRET="your_access_secret"
```

Dopo che la dipendenza richiesta è stata installata e i dati OAuth nell'ambiente della shell sono stati impostati, possiamo eseguire il programma come segue:

```
$ python stream.py -t -n 1000 > tweets.txt
```

Ci affidiamo all'I/O della shell di Linux per reindirizzare l'output con l'operatore > di `stream.py` a un file chiamato `tweets.txt`. Se tutto è stato eseguito correttamente, dovreste vedere una sequenza di testo in cui ogni riga è un tweet.
Notate che in questo esempio non abbiamo mai usato Hadoop. Nei prossimi capitoli vi mostreremo come importare un dataset generato dall'API Streaming in Hadoop e ne analizzeremo il contenuto su un cluster locale e su EMR.
Per ora, diamo un'occhiata al codice sorgente di `stream.py`, che trovate all'indirizzo `https://github.com/learninghadoop2/book-examples/blob/master/ch1/stream.py`:

```
import tweepy
import os
import json
import argparse

consumer_key = os.environ['TWITTER_CONSUMER_KEY']
consumer_secret = os.environ['TWITTER_CONSUMER_SECRET']
access_key = os.environ['TWITTER_ACCESS_KEY']
access_secret = os.environ['TWITTER_ACCESS_SECRET']

class EchoStreamListener(tweepy.StreamListener):
    def __init__(self, api, dump_json=False, numtweets=0):
        self.api = api
        self.dump_json = dump_json
        self.count = 0
        self.limit = int(numtweets)
        super(tweepy.StreamListener, self).__init__()

    def on_data(self, tweet):
        tweet_data = json.loads(tweet)
        if 'text' in tweet_data:
            if self.dump_json:
                print tweet.rstrip()
            else:
                print tweet_data['text'].encode("utf-8").rstrip()

            self.count = self.count+1
            return False if self.count == self.limit else True

    def on_error(self, status_code):
        return True
```

```
    def on_timeout(self):
        return True
…
if __name__ == '__main__':
    parser = get_parser()
    args = parser.parse_args()

    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_key, access_secret)
    api = tweepy.API(auth)
    sapi = tweepy.streaming.Stream(
        auth, EchoStreamListener(
            api=api,
            dump_json=args.json,
            numtweets=args.numtweets))
    sapi.sample()
```

Come prima cosa, importiamo tre dipendenze: `tweepy` e i moduli `os` e `json`, presenti nell'intepreter di Python versione 2.6 o successiva.
Definiamo poi la classe `EchoStreamListener`, che eredita ed estende `StreamListener` da `tweepy`. Come il nome può far intuire, `StreamListener` ascolta gli eventi e i tweet che vengono pubblicati nello stream in tempo reale e agisce di conseguenza.
Ogni volta che individua un nuovo evento, innesca una chiamata a `on_data()`. In questo metodo, estraiamo il campo `text` da un oggetto tweet e lo mostriamo nell'output standard con una codifica UTF-8. In alternativa, se lo script è invocato con `-j`, mostriamo l'intero tweet JSON. Quando lo script viene eseguito, istanziamo un oggetto `tweepy.OAuthHandler` con le credenziali OAuth che identificano il nostro account Twitter, e poi usiamo questo oggetto per l'autenticazione con i codici access e secret key dell'applicazione. Utilizziamo poi l'oggetto `auth` per creare un'istanza della classe `tweepy.API` (`api`).
Se l'autenticazione ha successo, diciamo a Python di ascoltare gli eventi sullo stream in tempo reale usando `EchoStreamListener`.
Una richiesta http GET all'endpoint `statuses/sample` viene eseguita da `sample()` e restituisce un campione casuale di tutti gli stati pubblici.

ATTENZIONE

Di default, `sample()` viene eseguita a tempo indeterminato. Ricordate di terminare esplicitamente la chiamata al metodo premendo Ctrl+C.

Riepilogo

In questo capitolo abbiamo compiuto una rapida panoramica sulla storia di Hadoop: da dove viene, la sua evoluzione e perché il rilascio della versione 2 è stato così fondamentale. Abbiamo anche descritto il mercato emergente delle distribuzioni

di Hadoop e abbiamo spiegato come nel libro utilizzeremo una combinazione tra distribuzioni locali e cloud.
Infine abbiamo descritto come configurare il software necessario, gli account e gli ambienti richiesti per i prossimi capitoli, oltre a illustrare come trarre dallo stream di Twitter quei dati che ci serviranno per gli esempi.
Fissate queste basi, passiamo a esaminare nel dettaglio il livello dello storage in Hadoop.