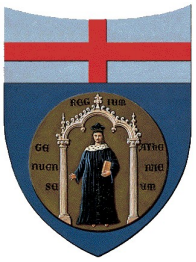
Università Degli Studi Di Genova



Facoltà di Scienze Matematiche Fisiche Naturali

Corso di Laurea in Informatica

# Piattaforma Windows Azure - Lato storage

Relatori
Prof. Maura Cerioli
Dott. Giovanni Lagorio

Prova finale di
Alessio Parma
Matr. N. 3247806

Anno Accademico 2011/2012

# Dedica

Questa relazione è dedicata a (in ordine alfabetico, non in ordine di importanza): Andrea, Artur, Chiara, Federica, Federico, Giuseppe, Guido, Marta, Matteo, Mirko, Niccolò, Renata. Queste sono le persone che mi hanno reso ciò che sono; dire loro semplicemente "grazie" non sarà mai abbastanza, ma lo farò lo stesso: grazie a tutti, davvero. Chissà, magari un giorno mi sdebiterò con una deliziosa torta fatta con il mio grande talento da cuoco!

> "A single rose can be my garden... a single friend, my world." - Leo Buscaglia

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Elenco degli algoritmi

# Capitolo 1

# Introduzione

La parola *cloud*, spesso accompagnata da *computing* e più raramente da *storage*, sembra essere sempre più frequentemente sulle labbra, o meglio, sugli schermi e sulle tastiere, di ogni professionista o ricercatore nel settore IT (Information Technology), nonché sembra essere sempre più l'argomento "protagonista" degli articoli a carattere tecnico. Dare una spiegazione dettagliata del perché questo fenomeno sia in atto esula dagli scopi della relazione; tuttavia, si cercherà di farlo ugualmente, anche se in modo necessariamente sintetico e semplificato.

Tutto questo verrà fatto mentre si cercherà di spiegare quale sia l'offerta della piattaforma Windows Azure (se ne riporta il logo in figura 1.1), gestita da Microsoft e orientata sia al *cloud computing*, sia al *cloud storage*. In particolare, in questa relazione si focalizzerà tutta l'attenzione sulla parte *storage* dell'offerta, descrivendo quali siano i servizi offerti, le modalità d'uso, i limiti e i costi, effettuando confronti con i servizi concorrenti (Amazon Web Services, ad esempio).

Inoltre, si faranno cenni ad altre componenti della piattaforma Windows Azure, soffermandosi più approfonditamente su quelle parti che hanno più affinità con la parte di memorizzazione, come, ad esempio, Windows Azure Marketplace.

Il resto di questo capitolo introduttivo sarà, come suggerisce il nome, dedicato a introdurre al lettore alcuni concetti relativi al mondo del cloud,


Figura 1.1: Logo della piattaforma Windows Azure, argomento principale della relazione.

cioè, le sue origini, i suoi punti di forza e di debolezza, e il suo attuale livello di utilizzo. Per farlo, si cercherà di limitare al minimo il gergo tecnico.

## 1.1 Cosa offre il cloud computing?

Innanzitutto, è necessario fare brevemente chiarezza su cosa si intende con *cloud computing* e come tale termine identifichi il servizio, quando in realtà esso riguarda solo una piccola parte dell'offerta globale. Con il termine *cloud* si denota l'infrastruttura su cui si basano tutti i servizi offerti; per le vere origini del nome, si veda sezione 1.2, ma ai fini della comprensione può risultare più utile pensare che la cloud si chiami così in riferimento a ciò che vede chi osserva una nuvola: egli la vedrà distante ma, soprattutto, la vedrà cambiare forma e posizione rapidamente. Analogamente alla nuvola, una infrastruttura di tipo *cloud* risulta un'entità distante dal cliente, che ha una capacità computazionale e di memorizzazione pressoché infinitamente variabili.

Si sarà notato che, nella frase precedente, si è detto che una cloud offre capacità computazionale e di memorizzazione; la prima offerta viene denominata *cloud computing*, mentre la seconda viene denominata *cloud storage*. Tuttavia, per motivi di praticità, il termine *cloud computing* di solito raccoglie entrambe le offerte, indicando sia la parte di calcolo, sia la parte di conservazione dati.

Ciascun fornitore di servizi di tipo cloud modella la propria offerta in modo diverso, puntando, ad esempio, verso una notevole facilità d'uso a svantaggio della personalizzazione, oppure offrendo solo la parte *storage* del cloud computing. Tuttavia, si può dire che alcuni fattori sono comuni a tutte le offerte: basso livello di manutenzione da parte del cliente, alta disponibilità

del servizio e replicazione dei dati, pagamento a consumo piuttosto che a tariffa fissa.

## 1.2 Un po' di storia

Anche se può sembrare un qualcosa di antitetico cercare di scrivere la *storia* di un fenomeno in piena crescita e sviluppo, si cercherà di dare alcuni cenni di come il cloud computing sia arrivato ad essere quello che attualmente è; in particolare, si noti subito come questa tecnologia, che spesso viene etichettata con aggettivi come "rivoluzionaria", "innovativa" e "recente", sia nata, almeno a livello concettuale, ben cinquant'anni fa, e che sono oramai dieci anni che viene usata *seriamente* in ambiti *enterprise*. Il materiale presentato in questa sezione proviene in gran parte da [1].

L'idea generale dietro alla tecnologia ai servizi di tipo cloud risale agli anni '60, quando John McCarthy[1] scrisse che "computation may someday be organized as a public utility" (un giorno, la capacità di calcolo potrebbe essere organizzata come una pubblica utilità). In seguito, un altro concetto nato agli inizi degli anni '90 ha contribuito al cloud computing: il *grid computing*[2]; esso si proponeva l'obiettivo di rendere la capacità computazionale tanto facile da usufruire quanto lo è collegare un dispositivo alla corrente elettrica (*power grid* in inglese).

Il termine *cloud computing* deriva, con tutta probabilità, dai diagrammi a forma di nuvola usati per rappresentare Internet nei libri di testo, come quello mostrato in figura 1.2. Il concetto tecnologico dietro al cloud computing, invece, deriva dalle compagnie di telecomunicazione, che negli anni '90, hanno

[1] John McCarthy, nato nel 1927 a Boston, è un informatico che ha ricevuto il premio Turing nel 1971 per i suoi notevoli contributi al campo dell'intelligenza artificiale. Egli stesso ha coniato il termine "intelligenza artificiale" ed è l'inventore di Lisp, un linguaggio di programmazione con paradigma prevalentemente funzionale.[4]

[2] Il termine *grid computing* si riferisce alla combinazione di risorse di calcolo, provenienti da molteplici domini amministrativi, con lo scopo di raggiungere un obiettivo comune. La griglia può essere pensata, a livello concettuale, come un sistema distribuito con carichi di lavoro non interattivi che coinvolgono un grande numero di file. Ciò che distingue il grid computing dai convenzionali sistemi per la computazione ad alte prestazioni (come i *cluster*) è che le griglie tendono ad essere accoppiate lascamente, eterogenee e sparse geograficamente.[5]

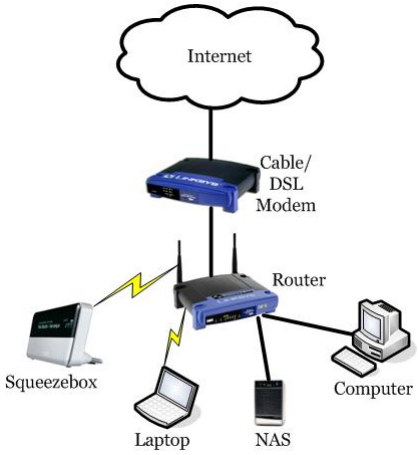



Figura 1.2: Diagramma a "nuvola" da cui il cloud computing ha probabilmente preso il proprio nome.

compiuto un radicale cambiamento passando da circuiti dati punto a punto a servizi di tipo rete virtuale privata (VPN, Virtual Private Network). Ottimizzando l'utilizzo delle risorse attraverso metodi di bilanciamento del carico di lavoro, tali compagnie riuscivano ad eseguire il loro lavoro più efficientemente ed economicamente. La prima volta che il termine *cloud computing* è stato usato nel suo contesto attuale è stato nel 1997, quando Ramnath Chellappa[3] in una lezione lo ha definito come un nuovo "computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone" (paradigma di calcolo dove i limiti di calcolo saranno determinati da motivazioni economiche, piuttosto che soltanto da limiti tecnici).

Uno dei primi attori nel palco del cloud computing è stata la società Salesforce.com, che nel 1999 ha introdotto il concetto di portare applicazioni di livello *enterprise* attraverso un semplice sito web. Amazon è stata la successiva società ad entrare in scena, lanciando nel 2002 Amazon Web Services (AWS); quindi, nel 2006 fu il turno di Google con la suite di strumenti per l'ufficio Google Docs, che ha avuto il merito di portare il cloud computing a

[3]Ramnath Chellappa è attualmente un professore associato nell'area "Information Systems & Operations Management" alla Goizueta Business School, nell'università di Emory.[6]

un più ampio e variegato bacino di utenti. Sempre nel 2006 vi è stata l'introduzione, sempre da parte di Amazon, del servizio Elastic Compute Cloud (EC2), il quale consentiva a piccole società e individui di prendere in affitto computer su cui far "girare" le proprie applicazioni.

In seguito, nel 2008, è stato creato Eucalyptus, la prima piattaforma open source e compatibile con le interfacce di AWS per fare il *deploy* di cloud private[4]; poi, è venuto OpenNebula, il primo software open source per eseguire il *deploy* di cloud private ed ibride[5]. Quindi, con il lancio di Windows Azure, il 2009 ha visto l'ingresso di Microsoft nel mondo del cloud computing.

## 1.3 Punti di forza del cloud computing

Il successo di ogni organizzazione, piccola o grande, è dato in parte dalla capacità (e dalla possibilità) di focalizzarsi sul *core business*, cioè, di investire il maggior numero di risorse sugli elementi pertinenti allo scopo principale dell'organizzazione, e di investire il minor numero di risorse negli elementi di "supporto".

### 1.3.1 Esempio del panificio

Prendiamo, come esempio esplicativo, un panificio. In quel caso, lo scopo principale è produrre diversi tipi di pane, che verranno consumati (e pagati) dai clienti. Per riuscire a produrre il pane, l'azienda di panificazione necessita di elementi di supporto come la farina e i sacchetti per confezionare i prodotti. Affinché la farina sia disponibile all'azienda, essa potrà scegliere due vie: o piantare del grano, o comprare dei sacchi di farina già pronti; ovviamente,

[4]Una cloud "privata" è un'infrastruttura che opera unicamente per una sola organizzazione, sia che sia gestita internamente o da terze parti, sia che sia ospitata internamente o esternamente. Questo tipo di cloud ha attratto diverse critiche, perché gli utenti devono comprarle, costruirle e gestirle, non beneficiando dei minori costi iniziali e della minore gestione di prima mano.[2]

[5]Una cloud ibrida è data dalla composizione di due o più cloud (pubbliche o private), che rimangono entità a se stanti ma sono legate assieme da una tecnologia standardizzata e proprietaria, che consente la portabilità di dati e applicazioni tra di esse. Come si può intuire, il modello ibrido offre i benefici dei molteplici modelli di *deployment*.[2, 3]

nel primo caso, il panificio si allontanerà molto dal suo *core business*, perché coltivare grano non rientra negli obiettivi dell'azienda, ma soprattutto perché tale azione, di per se, non è una soluzione "scalabile". Infatti, se il panificio dovesse raddoppiare il proprio numero di clienti, dovrebbe idealmente raddoppiare i campi di grano (con le conseguenti spese) e se, disgraziatamente, dovesse in seguito perdere quasi tutti i clienti, dovrebbe riadattare la coltivazione alle proprie esigenze. E se, per fortuna, dovesse riacquistare i clienti? Il panificio dovrebbe ricomprare di nuovo i campi, sempre che questo sia possibile. Una soluzione molto semplice sarebbe che il panificio comprasse una grande quantità di terreni, indipendentemente dalla propria produzione di pane; ma questo, a meno di una piccola idea che vedremo in seguito, non è un approccio *cost effective*. In altre parole, l'azienda probabilmente si ritroverebbe a pagare, per la maggior parte del tempo, terreni che non usa.

Quindi, la maggior parte dei panifici si appoggia a un *servizio* che gli fornisca la farina necessaria. Per il panificio, il distributore di farina può idealmente fornire un numero illimitato di sacchi di farina, in qualunque stagione o in qualunque situazione climatica. Ma, cosa più importante, il panificio ha la possibilità di richiedere un numero *variabile* di sacchi di farina, puramente in base alle proprie esigenze: se l'azienda dovesse chiudere, causa malattia, per un lungo periodo, potrebbe smettere di comprare farina, mentre se coltivasse il grano in proprio potrebbe, per vari motivi, non avere la possibilità di smettere di lavorare i campi. Questo approccio sembra molto *cost effective*, perché il panificio paga solo per la farina che consuma, se ne consuma.

### 1.3.2 Dal pane al mondo IT

A questo punto, occorre trasporre l'esempio dell'azienda di panificazione a un'azienda IT, cercando di capire quali elementi hanno il ruolo della farina in un'azienda a carattere informatico e tecnologico. Chiaramente, tale di tipo di organizzazione disporrà di workstation e server, ciascuno dei quali sarà munito di una certa capacità computazionale e di memorizzazione; pertanto, a livello prettamente concettuale ed esemplificativo, potremmo dire

che ciascuna azienda ha una *capacità computazionale totale* (CCT, la somma delle singole capacità computazionali) e una *capacità di memorizzazione totale* (CMT, analogamente la somma delle singole capacità di memorizzazione). Proprio come il panificio, l'azienda operante in campo IT può avere bisogno, a seconda del numero di clienti, di CCT e CMT variabili; in particolare, potrebbe voler avere sempre grandi CCT e CMT, ma limitati costi di gestione (la gestione dell'infrastruttura IT, come prima, non rientra nel *core business*).

Il cloud computing è una soluzione al problema sopra esposto: con misure, costi e modalità diverse, ciascuna cloud offre la possibilità di avere CCT e CMT *infinitamente* variabili, senza costi di gestione. In un certo senso, alcuni costi di gestione permangono: occorre sempre gestire le workstation e la rete interna; tuttavia, i costi relativi ai server possono essere notevolmente ridotti o eliminati del tutto, oltre al fatto che, in casi particolari, anche le workstation, usando la cloud, potrebbero essere depotenziate. Se si vedono le cose da questo punto di vista, la cloud si rivela un fattore chiave di successo per le *startup*, piccole società che, per il semplice fatto di essere piccole, non possono permettersi di investire risorse se non nel *core business*; infatti, a esse la cloud offre la CCT e la CMT necessarie per raggiungere i loro obiettivi. Anche una grande azienda potrebbe ricavare dei vantaggi come farebbe una piccola? In generale, la risposta non è così certa, perché finora si sono presentati solo gli aspetti positivi del cloud computing.

## 1.4 Punti di debolezza del cloud computing

Come tutte le cose di questo mondo, il cloud computing non è perfetto: esso presenta svantaggi la cui "importanza" dipende dal tipo del cliente che usufruisce dell'offerta; ad esempio, il medesimo difetto può essere trascurabile per un tipo di società, ed essere assolutamente bloccante per un'altra. Riprendendo l'esempio dell'azienda di panificazione, si introdurranno ora i principali svantaggi del cloud computing.

### 1.4.1 Nuovamente, l'esempio del panificio

Per introdurre gli aspetti negativi della cloud, ritorniamo all'esempio del panificio, esempio semplice con cui si spera che il lettore abbia oramai preso familiarità perché ricorrerà nuovamente anche in seguito. Precedentemente, si è detto che quasi tutte le aziende di quel tipo si appoggiano a fornitori specializzati per ottenere la farina di cui necessitano. Così facendo, però, creano una forte dipendenza tra se stessi e il fornitore: se questo decidesse di raddoppiare i prezzi, o avesse problemi a soddisfare la richiesta, l'azienda di panificazione potrebbe andare incontro a problemi di produzione. La soluzione più immediata e ovvia, in caso di aumento dei prezzi, è rivolgersi ad un altro distributore: tuttavia, questo non è sempre possibile. Supponiamo, ad esempio, che il fornitore precedente distribuisse la farina in sacchi di una forma particolare e che l'azienda di panificazione abbia costruito un sistema ad hoc per smistare automaticamente i sacchi; se il nuovo fornitore offre la farina in sacchi di forma differente, il sistema andrà ripensato, o gettato via del tutto.

Rimanendo sempre nell'ambito della farina, si pensi all'elevato grado di fiducia che il panificio deve riporre nel distributore; infatti, chi impedisce al distributore di mescolare farina di alta qualità con farina di bassa qualità, o fare altre cose più turpi? E se il grano è coltivato dall'estero, chi assicura che esso non venga trattato con prodotti che, se fosse coltivato localmente, sarebbero banditi dalla legislatura? Di fronte a questi quesiti e a queste incertezze, il panificio, se è riuscito ad espandersi abbastanza da essere una grande azienda, potrebbe seriamente cominciare a fare quanto si era scartato precedentemente, cioè, coltivare da se il grano. In particolare, può riprendere l'idea di avere campi superiori alle proprie necessità, con il piccolo accorgimento di rivendere la farina che avanza, diventando distributore a sua volta.

### 1.4.2 Ritorno al punto di vista IT

Come prima, occorre cercare di ricollegare i concetti semplici appena esposti con i problemi che un'azienda del settore IT potrebbe incontrare. In seguito si

Figura 1.3: Il rapporto tra un'edera e un edificio è simile a quello che si instaura tra un'azienda e un fornitore di servizi di tipo cloud.

vedrà che i fornitori di servizi di tipo cloud, proprio come i fornitori di farina usavano sacchi di forma differente, hanno delle interfacce particolari per i propri servizi; questo fatto crea inevitabilmente una dipendenza dell'azienda nei confronti del fornitore, fenomeno che tecnicamente viene detto *vendor lock-in*. Tale fenomeno, presente non solo in ambito cloud e neppure limitato al settore IT, indica la situazione in cui un'azienda non può "staccarsi" da un fornitore a meno di non interrompere, per un certo tempo e con certi costi, il *core business*. Se una *startup* si appoggia a servizi cloud (cosa che, come si è detto, è più che sensata) e *attorno* ad essi cresce, proprio come un'edera cresce sui muri di un'abitazione (figura 1.3), incontrerà seri problemi nel caso in cui si trovasse a dover reagire a cambiamenti di fornitore.

Sfortunatamente, i problemi non finiscono qui. Nell'esempio si è detto che il panificio potrebbe risentire del fatto che la farina *importata* è gestita da terze parti; in modo analogo, l'azienda IT può avere sia problemi tecnici, sia problemi legali, per lo stesso identico motivo. Si pensi alla gestione dei dati, cosa che verrà trattata diffusamente nel seguito: occorre avere una garanzia sulla loro integrità, nonché la garanzia che, a meno di un permesso esplicito, non vengano trasmessi a terze parti o, similmente, usati dal gestore

stesso della cloud. Inoltre, spesso i centri dati in cui *risiedono* le cloud sono in un'altra nazione rispetto a quella in cui si trova l'azienda; pertanto, la nazione dell'azienda potrebbe imporre certe regolamentazioni sui dati, mentre quella del centro dati ne potrebbe imporre altre, più o meno restrittive.

Quindi, esattamente come il panificio, un'azienda IT si ritrova di fronte a incertezze che, sfortunatamente, sono implicitamente inamovibili. Per aziende di piccola-media scala, non c'è soluzione, se non quella di appoggiarsi alla cloud abbracciando i rischi ad essa collegata, oppure iniziare in modo non competitivo e non *scalabile* usando infrastrutture interne. Ma per le aziende di grande dimensione, con un'infrastruttura interna già consolidata, i rischi sembrano superare i vantaggi, a meno che l'organizzazione decida di costruire da se dei servizi di tipo cloud, e di rivendere la potenza computazione di "avanzo" ad altre società. Per quanto possa sembrare una scelta forse bizzarra, questa è la situazione in cui i più grandi fornitori di servizi basati sulla cloud si ritrovano: il cloud computing sorregge in modo più o meno determinante l'infrastruttura informatica di tali società e, al tempo stesso, viene utilizzata da altre aziende.

## 1.5 Il cloud computing oggi

Come si può notare in figura 1.4, attualmente il mercato legato all'offerta di servizi di tipo cloud è altamente popolato e attivo; si può tranquillamente dire che tutte le grandi società informatiche hanno tra i loro servizi almeno un servizio basato sulla cloud, anche se ognuna li offre in maniera e quantità differenti: alcune offrono piattaforme più o meno complesse per gli sviluppatori software (come Microsoft, Amazon, Google e Salesforce.com), mentre altre si appoggiano a cloud esterne per fornire servizi di backup e sincronizzazione ai propri utenti (come Apple con il prodotto iCloud e Dropbox con l'omonimo prodotto).

Nonostante il cloud computing sia largamente utilizzato e pubblicizzato, tuttavia il suo futuro non è chiaramente definito; infatti, come si è visto nelle sezioni 1.3 e 1.4, esso non ha solo vantaggi, ma anche svantaggi. In particolare, il problema più ricorrente e *bloccante* è la frequenza con cui la


Figura 1.4: Principali attori nel mercato del cloud computing.

cloud, per questioni tecniche interne, diventa non disponibile agli utenti: ad esempio, nell'aprile dell'anno corrente, AWS, il prodotto di Amazon, ha sofferto un fallimento notevole[9], che ha messo offline importanti siti come Reddit. Tuttavia, durante tale fallimento, altri siti, come Netflix, non hanno ceduto, in quanto progettati per essere altamente resistenti alle emergenze.

Pertanto, il cloud computing, come ogni strumento, hai dei punti di debolezza che possono essere in una certa misura aggirati con una adeguata preparazione; ma questa preparazione ha dei costi, che molte società vorrebbero non avere: perché, pur pagando per un servizio di tipo cloud, esso si dimostra non affidabile al cento per cento? Questo dubbio, in un certo senso legittimo, tiene lontane molte grandi società dall'impiegare estensivamente il cloud computing, mentre le piccole società, che non hanno molte altre scelte, si appoggiano ad esso in modo sempre più crescente. Che siano le startup, e non le grandi aziende, a decidere il futuro del cloud computing?

# Capitolo 2

# Glossario

Lo scopo di questo capitolo è di introdurre (o rinfrescare) al lettore alcuni concetti chiave che verranno usati spesso nel corso della relazione. I concetti sono disposti senza alcun particolare ordine, in quanto tutti rivestono eguale importanza nella comprensione di quanto verrà detto in seguito.

## 2.1 IaaS, PaaS, SaaS

Si parlerà ora delle tre idee fondamentali su cui si basa il cloud computing: IaaS, PaaS, SaaS; rispettivamente, i tre acronimi indicano l'offerta come servizio di un'infrastruttura (IaaS), di una piattaforma (PaaS) e di un software (SaaS). Offrire un prodotto come servizio significa dare la possibilità ad un cliente di usare quel prodotto, senza che il cliente si curi della sua manutenzione e attuando un sistema di pagamento a consumo piuttosto che a tariffa fissa.

Pertanto, ora si dedicherà a ciascuno dei tre concetti una breve descrizione, accompagnata dall'oramai immancabile esempio del panificio. Prima di proseguire nella lettura, si consiglia di dare un'occhiata alle figure 2.1 e 2.2, dove vengono sinteticamente raffigurate le idee chiave dietro ai concetti di IaaS, PaaS e SaaS.

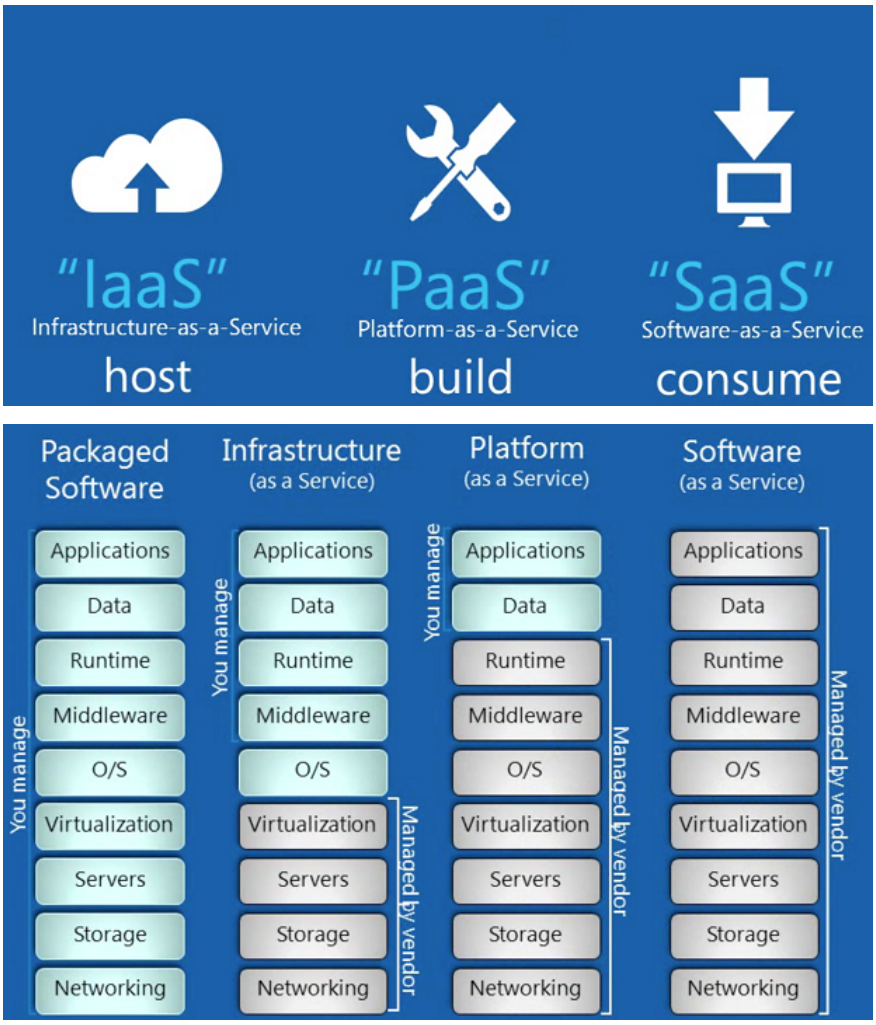



Figura 2.1: Schemi riassuntivi per le differenze tra IaaS, PaaS, SaaS.

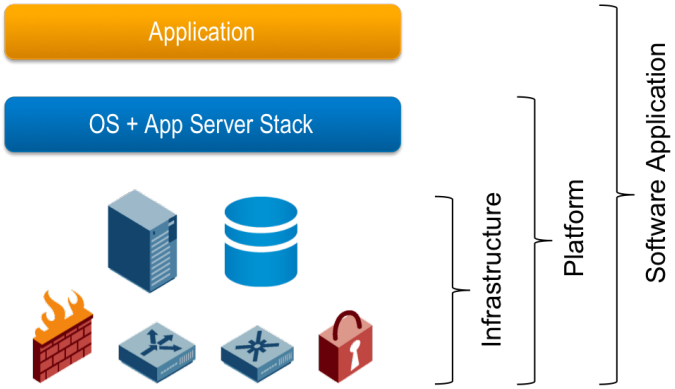



Figura 2.2: Un altro schema che rapporta IaaS, Paas e SaaS.

### 2.1.1 IaaS (Infrastructure as a Service)

Nel caso di un normale panificio il dirigente (o chi per lui) decide, per ciascun tipo di macchinario (forni, impastatrici, celle frigorifere, ecc ecc) quanti esemplari servono, e di quale tipo. Una volta presa questa decisione, però, saranno società di terze parti ad imbastire un'infrastruttura compatibile con le richieste del dirigente, a mantenerla nel tempo e a modificarla in caso di richiesta (necessità di un nuovo forno, potenziamento della rete elettrica, ecc ecc). In particolare, si potrebbe immaginare che i fornitori possano non far pagare il singolo macchinario, ma richiedere un compenso in base all'attività (tempo di lavoro, carico di lavoro, o entrambi) del macchinario. Vedendo le cose in questo modo, sembra evidente che l'*infrastruttura* del forno sia un *servizio* offerto da terze parti, mentre l'azienda di panificazione si occupa di fare il pane utilizzando i macchinari a disposizione (cioè, può scegliere fattori come la temperatura dei forni e la velocità delle impastatrici).

Pertanto, a livello IT, un servizio di tipo IaaS fornisce i centri dati, l'infrastruttura hardware e le risorse software attraverso Internet. In generale, esso è un modello di approvvigionamento nel quale un'organizzazione delega a terze parti la responsabilità degli strumenti a supporto delle operazioni, incluso lo storage, l'hardware, i server e le componenti della rete. Il fornitore del servizio possiede gli strumenti ed è responsabile del loro mantenimento (cosa che, come si vede in figura 2.3, può essere un grande sollievo); il cliente, di solito, paga in base a quanto usa il servizio (concetto di *elastic cloud*).

In alcuni casi, la sigla HaaS (Hardware as a Service) viene usata come sinonimo di IaaS; infatti, questo tipo di servizio assomiglia in una certa misura all'*hosting*. Tuttavia, la differenza principale rispetto a esso sta nel metodo di pagamento e nella flessibilità del servizio. In entrambi i casi, ciò che una società prende in affitto sono macchine, ma, mentre nella cloud prevale il concetto di macchina *virtuale*, nell'hosting prevale quello di macchina *fisica*, con tutti gli svantaggi che tale concetto comporta. Nella cloud è possibile aumentare facilmente il numero di macchine operanti; al contrario, in un servizio di tipo hosting, occorre *aggiungere* fisicamente nuovi server per soddisfare la domanda. Inoltre, sempre per motivi analoghi, in un servizio

Figura 2.3: Non dover gestire l'infrastruttura può risultare in un grande sollievo.

basato sulla cloud è facile impostare un sistema di pagamento a consumo, mentre nell'hosting si paga in base al numero e alla qualità delle macchine affittate, indipendentemente da quanto siano usate.

### 2.1.2 PaaS (Platform as a Service)

Si supponga ora che un individuo maldestro e poco abile in cucina, come lo scrittore di questa relazione, decida di aprire un panificio. Egli non sa nulla di come e quanto il pane vada cotto, per non parlare di come si prepari l'impasto; tuttavia, egli, presa la pasta pronta, potrebbe riuscire a fare il pane, in modo che sia cotto da qualcuno di competente. In pratica, rispetto al caso IaaS, gli operai di questo panificio si occuperebbero soltanto di preparare il pane, lasciando a terze parti la gestione di altri compiti (come l'impastatura e la cottura). Questo comporta che siano le terze parti a gestire l'infrastruttura dei forni, cioè, siano esse a decidere fattori come il numero totale dei forni, e siano sempre esse a stabilire le modalità "operative", come la temperatura e i tempi di cottura. In altre parole, la *piattaforma* su cui si lavora è gestita e mantenuta da terze parti.

Quindi, in generale, un servizio di tipo PaaS fornisce una piattaforma su cui gli sviluppatori di software possono costruire nuove applicazioni o

estenderne di esistenti senza richiedere l'acquisto di materiale e conoscenze per la gestione dei server; in particolare, l'infrastruttura sottostante viene nascosta tramite astrazioni più o meno complesse.

Uno dei punti di debolezza dei servizi di tipo PaaS è il rischio per i clienti di creare una forte dipendenza nei confronti del fornitore del servizio, detta *vendor lock-in*, in particolare se il servizio offre interfacce o linguaggi di sviluppo proprietari. Inoltre, un altro punto debole è dato dalla bassa flessibilità delle offerte, cosa assolutamente comprensibile per come il servizio è impostato, ma che potrebbe rappresentare un grande problema per gli utenti le cui necessità variano rapidamente.

### 2.1.3 SaaS (Software as a Service)

A questo punto, l'esempio del panificio comincia a non tenere più il passo di ciò che dovrebbe esemplificare, ma si proverà lo stesso ad utilizzarlo, visto che il lettore, oramai, si sarà abituato all'analogia. Come per molti altri tipi di business, anche la panificazione può essere gestita in franchising, cioè, più aziende sparse per un dato territorio condividono il marchio e alcune linee guida sulla produzione, ma viene lasciata a loro la decisione di come gestire la produzione. Per chi gestisce il franchising, sarebbe molto interessante se tutte le filiali fossero una "fotocopia" di una filiale ideale, che viene pensata e controllata direttamente dal gestore; cosa più interessante, sarebbe utile che tutte le filiali cambiassero al cambiare del modello ideale. In questo modo, i clienti sfrutterebbero un servizio che è uniforme indipendentemente dal luogo in cui si trovi, e il gestore potrebbe effettuare cambiamenti e migliorie (come l'aggiunta di un nuovo tipo di pane) a una sola filiale (quella ideale), sapendo che anche le altre subiranno le stesse modifiche.

Un software offerto come un *servizio* si comporta esattamente come la filiale ideale nel modello sopra esposto: in generale, offre un controllo centralizzato delle modifiche e un'interfaccia uniforme per tutti i clienti. I servizi di tipo SaaS sono la parte più matura, conosciuta e usata del cloud computing. Può essere definito come un modello di deployment del software in cui le applicazioni sono ospitate dal produttore o da un fornitore di servizi e sono


Figura 2.4: Logo del protocollo OData.

disponibili agli utenti attraverso una rete, di solito Internet. Pertanto, questa è la prima differenza dal tradizionale modello di deployment del software, detto SaaP (Software as a Product), nel quale i clienti acquistano il software e lo installano sui loro personal computer. In particolare, i vantaggi di SaaS rispetto a SaaP sono i seguenti:

- Amministrazione semplificata.
- Gestione automatica degli aggiornamenti e delle correzioni.
- Compatibilità, tutti gli utenti hanno la stessa versione del software.
- Accesso globale.

In generale, SaaS è considerato la parte più matura del cloud computing per via della sua alta flessibilità, comprovati servizi di supporto, scalabilità, ridotta manutenzione verso il cliente.

## 2.2 OData (Open Data Protocol)

OData (si mostra il logo in figura 2.4) è un protocollo web aperto, il cui scopo è consentire l'esecuzione di ricerche su sorgenti dati e la manipolazione dei dati stessi. Il protocollo permette ad un "consumatore" di interagire con una sorgente dati attraverso il protocollo HTTP e di ricevere il risultato in diversi formati, come Atom, JSON o semplice XML; in particolare, tra le richieste inviabili alla sorgente sono contemplate la paginazione, l'ordinamento e il filtraggio dei dati. In figura 2.5 è presente un piccolo esempio di un URI che esegue una query secondo il protocollo OData.

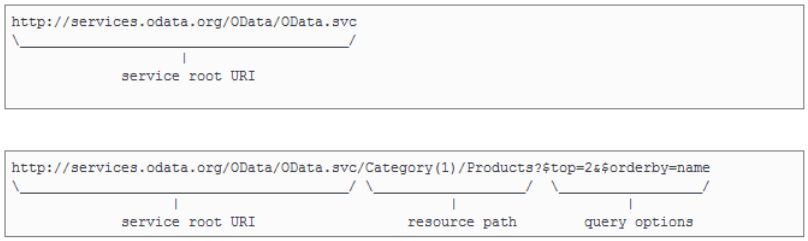



Figura 2.5: Esempio di URI scritto e scomposto secondo il protocollo OData.

Le specifiche del protocollo OData sono disponibili sotto la licenza Microsoft "Open Specification Promise"[1], in modo tale che terze parti, inclusi progetti *open source*, possano costruire client e servizi basati su tale protocollo. In particolare, la scelta di avere una licenza più *libera* è dovuta al fatto che Microsoft vorrebbe che il protocollo OData diventasse lo standard usato dal Web, superando concorrenti come GData, l'equivalente (e molto simile) protocollo di Google.

Microsoft ha rilasciato strumenti atti a facilitare lo sviluppo di software che si appoggia al protocollo in questione per alcuni famosi linguaggi e piattaforme, come .NET, PHP, Java, JavaScript e altri. Molte altre tecnologie di Microsoft supportano il protocollo, come SharePoint Server 2010, Excel 2010, Windows Azure Storage, SQL Server 2008 R2; tramite alcuni plugin è possibile *integrare* l'uso di OData all'interno di Visual Studio, in modo da poter visualizzare graficamente i dati e di poterli far interagire con l'Entity Framework.

Buona parte delle informazioni presentate in questa sezione sono state ricavate da [7].

## 2.3 REST (Representational State Transfer)

REST è uno stile architetturale per software di rete su larga scala che si appoggia alle esistenti tecnologie e protocolli; esso descrive come le risorse

[1] La licenza Microsoft "Open Specification Promise" (o OSP), è una promessa irrevocabile di Microsoft, pubblicata nel settembre 2006, di non far valere diritti legali su alcuni brevetti Microsoft riguardanti le implementazioni di una certa lista di tecnologie.[8]

distribuite possono essere definite e indirizzate, focalizzando l'attenzione sullo scambio delle informazioni e sulla scalabilità.

In altre parole, REST prescrive come definire e indirizzare sorgenti specifiche di informazioni, comunemente note come risorse. Le risorse possono essere riferite individualmente attraverso un URI (Universal Resource Identifier); l'URL (Uniform Resource Locator) è il famoso tipo di URI utilizzato per gli indirizzi Web. Il termine REST spesso indica qualunque semplice interfaccia usata per trasmettere dati di un particolare dominio attraverso il protocollo HTTP senza la necessità di ulteriori strumenti per lo scambio di messaggi o il tracciamento della sessione.

I sistemi costruiti secondo lo stile REST vengono detti "RESTful" e sono costituiti dai seguenti elementi base:

- Elementi relativi ai dati: risorse, identificatori di risorse (indirizzi di rete, URL) e rappresentazioni delle risorse (documenti HTML), e immagini sono acceduti attraverso un'interfaccia standardizzata come HTTP.

- Componenti: server di origine, gateway, proxy e user agent comunicano trasferendo rappresentazioni di risorse attraverso l'interfaccia definita, non operando direttamente sulle risorse stesse. Di solito, questo è realizzato attraverso operazioni ben definite, come le operazioni GET e PUT offerte da HTTP.

- Connettori: client, server e cache, così come i tunnel (connessioni SSL) presentano un'interfaccia astratta per la comunicazione, nascondendo i dettagli implementativi dei meccanismi di comunicazione.

- Interazione priva di stato: tutte le richieste inviate ai connettori devono contenere tutte le informazioni necessarie per comprendere la richiesta, senza dipendere da una qualunque richiesta precedente; in altre parole, tutti i messaggi devono trasportare anche le informazioni necessarie per comprendere il contesto dell'iterazione.

Infine, la più grande e nota applicazione dello stile architetturale REST è il Web stesso, caratterizzato dall'uso del protocollo HTTP e dagli URL come meccanismi di indirizzamento.

# Capitolo 3

# Piattaforma Windows Azure

Si presenterà ora la piattaforma Windows Azure, elencando brevemente tutti i servizi offerti e, successivamente, i vari strumenti messi a disposizione da Microsoft e da terze parti per interagire e sviluppare software su di essa.

Prima di proseguire con le varie descrizioni, occorre affermare che, secondo lo scrittore di questa relazione, Microsoft ha "abusato" del nome Windows Azure. Infatti, esso denota la piattaforma, ma indica anche un suo sottoinsieme di servizi e viene usato come prefisso per altri servizi, come si vedrà in sezione 3.1. Pertanto, al fine di evitare la confusione generata dal solo termine *Windows Azure*, si cercherà di disambiguare ogniqualvolta necessario.

## 3.1 Visione generale

Windows Azure Platform è, come si può intuire facilmente dal nome, un servizio PaaS; pertanto, l'attenzione non è focalizzata tanto sulla configurabilità del servizio, piuttosto sulla sua facilità e rapidità d'uso, nonché su un bassissimo livello di manutenzione. La piattaforma può essere suddivisa nei seguenti servizi, come rappresentato in figura 3.1:

- Windows Azure, comprendente le parti Compute, Storage, Fabric Controller, Content Delivery Network e Connect
- SQL Azure

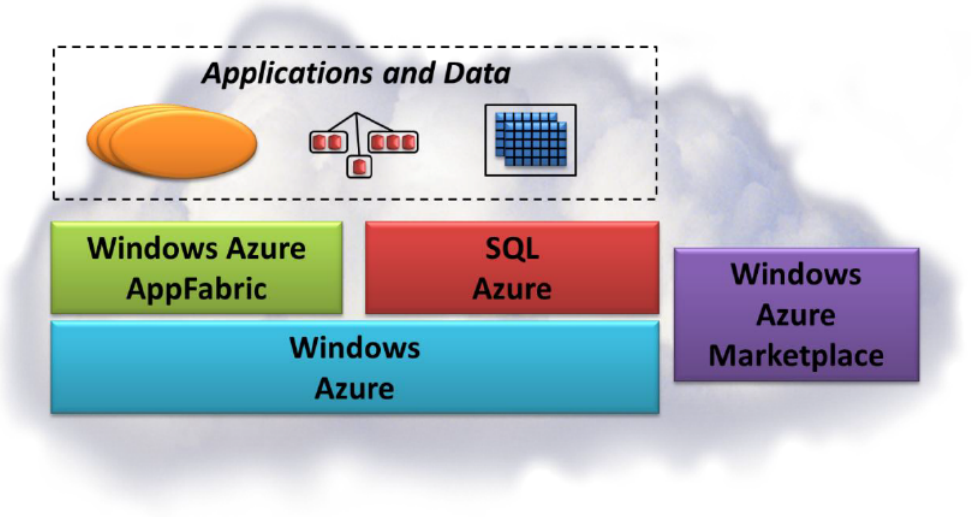



Figura 3.1: I principali servizi di Windows Azure.

- Windows Azure AppFabric
- Windows Azure Marketplace

Per ciascun servizio, segue ora una breve descrizione; nei capitoli successivi si approfondiranno tutti quei servizi che, in misura più o meno forte, riguardano la memorizzazione dei dati, cioè, Azure Storage, Content Delivery Network, SQL Azure e Azure Marketplace. Tutte le figure esposte in questa sezione provengono da [10].

### 3.1.1 Windows Azure

Windows Azure, facilmente confondibile con l'omonima piattaforma, è, come si nota in figura 3.2, un gruppo di servizi atti a rendere utilizzabile al cliente la capacità computazionale (Compute) e di memorizzazione (Storage) della piattaforma, fornendo anche servizi accessori (AppFabric, Content Delivery Network, Connect) per plasmare al meglio l'offerta.

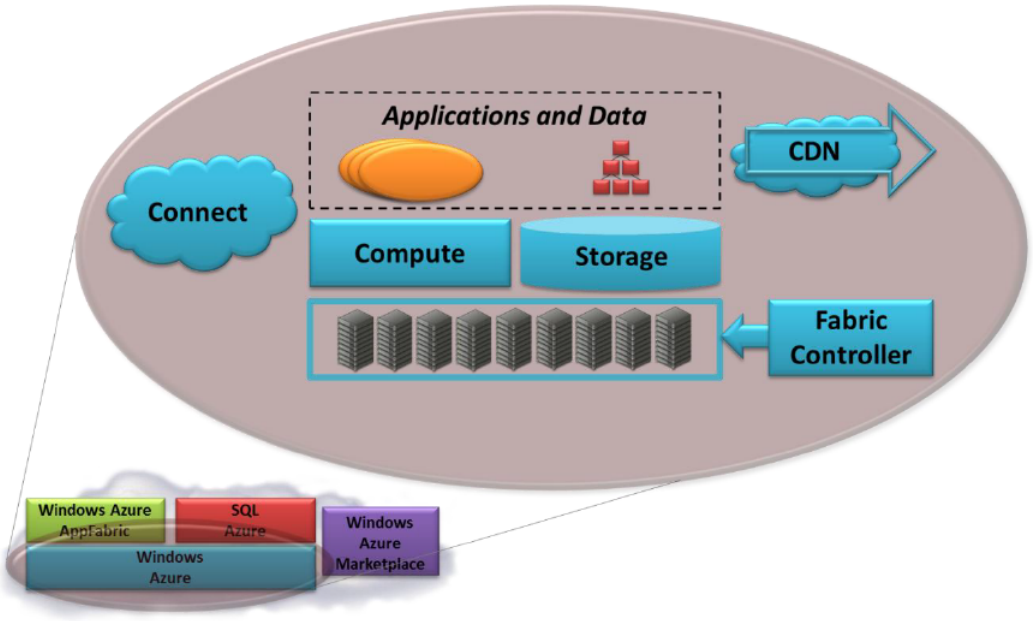



Figura 3.2: Servizi racchiusi sotto il nome "Windows Azure".

### Compute

Come si può immediatamente dedurre dal nome, la parte Compute di Windows Azure consente di sfruttare la capacità computazionale offerta dalla piattaforma; in particolare, questo servizio definisce tre modi, o meglio, *ruoli*, per usufruire di tale capacità: i ruoli *Worker*, *Web* e *VM*. Vi possono essere diverse istanze in esecuzione per ciascun ruolo; ciò non solo significa che più Worker, ad esempio, possono essere in esecuzione, ma anche che più Worker dello stesso tipo possono esserlo.

Un ruolo di tipo Worker, cioè, lavoratore, si occupa proprio di eseguire puramente operazioni di calcolo. Il *lavoro* che il Worker esegue può essere di vario genere; si parte dal calcolo puro, come il processamento di una serie di dati, per poi passare, ad esempio, alla gestione dei messaggi e dei corrispondenti canali per spedirli e riceverli. In generale, la vita di un Worker può essere riassunta nei seguenti passi:

1. Ricezione di un messaggio contenente il lavoro da eseguire;

2. Esecuzione del lavoro, o quello contenuto nel messaggio, o qualcosa di collegato ad esso;

3. Opzionalmente, segnalazione della fine del proprio lavoro;

4. Ritorno al punto 1, in attesa di messaggi in arrivo.

Al contrario, un ruolo di tipo Web ha uno e un solo scopo, eseguire applicazioni basate sul Web, mentre un ruolo di tipo VM può far girare un'immagine fornita dall'utente di Windows Server 2008 R2.

### Storage

L'offerta di Windows Azure per la memorizzazione dei dati è l'argomento principale della relazione; pertanto, qui si darà soltanto una breve descrizione dei servizi, coerentemente con gli altri, lasciando ai capitoli successivi il compito di approfondire l'argomento. Attualmente, il servizio fornisce tre modi differenti per memorizzare i dati: i blob, le tabelle e le code. Essi operano secondo modalità completamente differenti l'uno dall'altro, non lasciando spazio ad ambiguità di scelta tra uno dei tre. Tutti i servizi di memorizzazione possono essere acceduti via HTTP utilizzando un'interfaccia REST; per eseguire ricerche sulle tabelle, è necessario utilizzare il protocollo OData.

Il servizio relativo ai blob è dedicato alla memorizzazione di file testuali e binari; ciascun blob può raggiungere dimensioni notevoli (sino a 1TB) ed al blob stesso possono essere associati proprietà e metadati. Inoltre, i blob possono essere raccolti, per questioni di modularità, dentro dei "contenitori"; ovviamente, sono consentite operazioni di upload e download su ogni blob. Per ulteriori dettagli ed esempi d'uso, si veda capitolo 5.

Le tabelle, contrariamente a quanto si possa supporre dal nome, non sono tabelle di un database relazionale, ma si avvicinano piuttosto all'approccio seguito dai recenti database NoSQL e sono concettualmente simili alle tabelle proposte da software come Microsoft Excel. In pratica, tali tabelle non forzano un certo schema sulle righe che possono contenere, lasciando piena libertà di inserire righe con schemi differenti all'interno di una stessa tabella. Per approfondimenti, si veda capitolo 5.

Infine, le code offrono un meccanismo che segue i principi di persistenza e durabilità per consentire ai *ruoli* della parte Compute di comunicare tra di loro; come ci si può aspettare, le code utilizzano una politica FIFO per la gestione dei messaggi (anche se essa non sempre è garantita). Le code di Windows Azure sono una parte dell'offerta di facile comprensione e di grande utilità, si parlerà nuovamente di esse in capitolo 6.

### Fabric Controller

Il servizio Fabric Controller è la ragione fondamentale per cui la piattaforma Windows Azure viene definita di tipo PaaS: infatti, esso si occupa di portare avanti tutti quei compiti di manutenzione, dei quali, senza di esso, si dovrebbe occupare il cliente. In particolare, il Fabric Controller si assicura che il numero pattuito di istanze di ogni ruolo sia sempre in esecuzione (se un'istanza fallisce, è suo compito crearne un'altra) e, cosa importante per gli amministratori di sistema, esso stesso si occupa di applicare patch al sistema operativo e ad altri software di sistema.

### Content Delivery Network

Un'applicazione web potrebbe utilizzare i blob per memorizzare informazioni che sono accedute da più parti del mondo. Tuttavia, i centri dati Microsoft sarebbero, in quel caso, necessariamente *distanti* dalla maggior parte dei clienti; questo fatto crea un *ritardo* nella consegna dei contenuti richiesti dai clienti.

Risolvere questo problema è il compito del servizio Content Delivery Network, che consente di "avvicinare" i blob contenenti le informazioni richieste ai clienti richiedenti, velocizzando la consegna e migliorando i tempi di risposta per i clienti stessi.

### Connect

Questo servizio è stato pensato per coloro che, per svariati motivi (ad esempio, legali) non possono spostare *interamente* il proprio sistema all'interno della cloud, ma devono tenerne una parte internamente. Per fissare le idee,

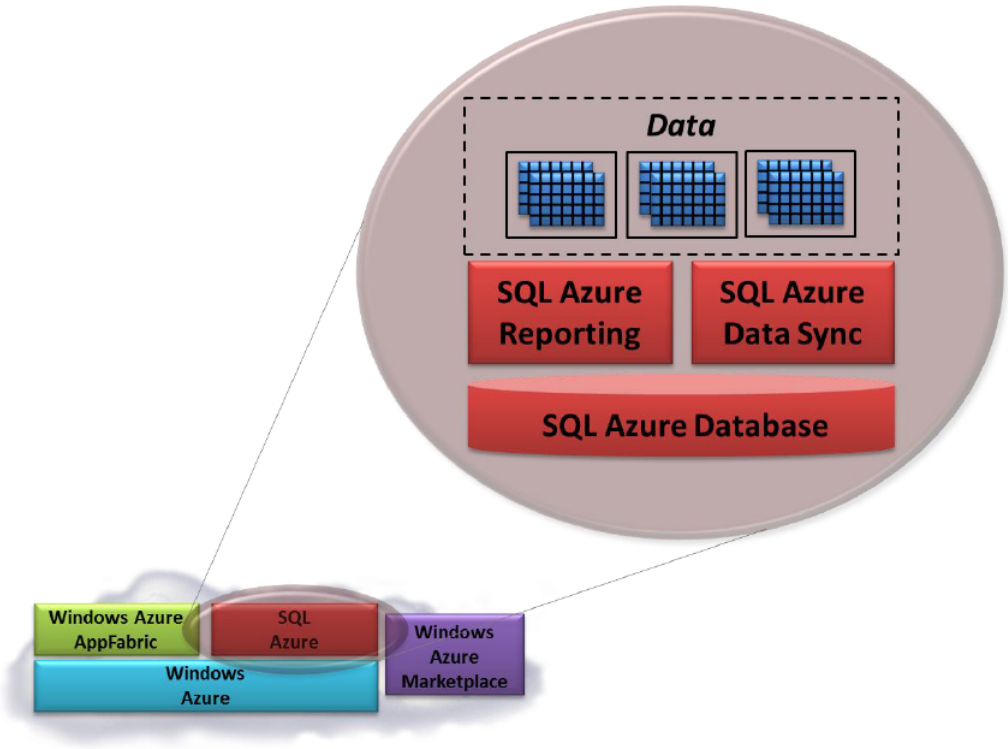



Figura 3.3: Schema riassuntivo delle componenti di SQL Azure.

si pensi ad un'applicazione che viene eseguita sulla cloud, ma deve accedere ai dati memorizzati su un database SQL Server, esterno alla cloud. Tramite il servizio Windows Azure Connect è possibile effettuare una connessione a livello IP, di fatto rendendo l'applicazione e il database appartenenti alla stessa rete IP.

### 3.1.2 SQL Azure

Come visto in sezione 3.1.1, l'offerta di Windows Azure per la memorizzazione dei dati non contempla un database relazionale, ma solo tabelle prive dei vincoli e delle capacità di vero database. Pertanto, per facilitare la migrazione sulla cloud di applicazioni esistenti, nonché per rispondere a tutti quei problemi che necessitano di un database relazionale, è nato il servizio SQL Azure, che offre un database SQL Server operante nella cloud. Esso è, a

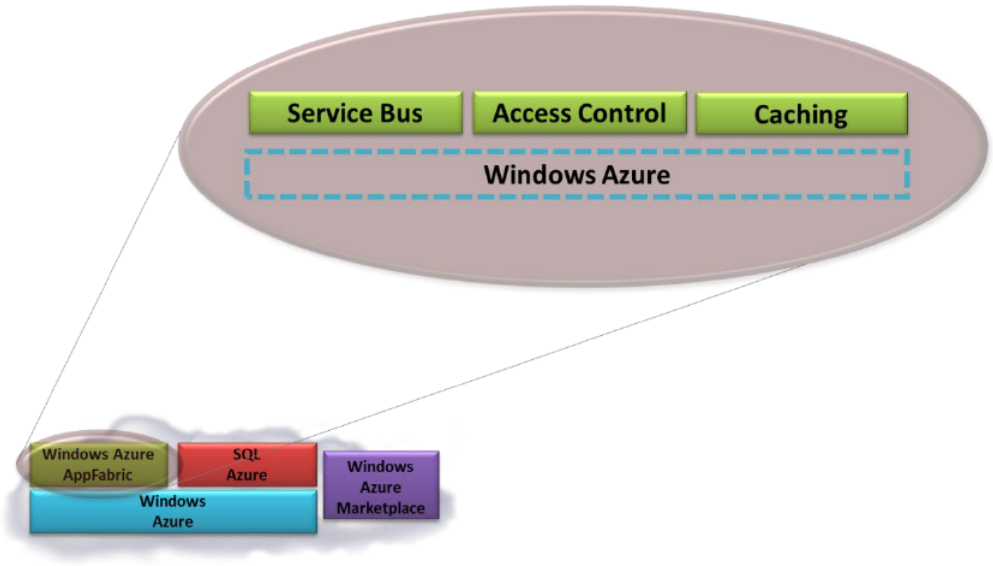



Figura 3.4: Windows Azure AppFabric pone un ulteriore livello di astrazione su Windows Azure, facilitando la vita agli sviluppatori di applicazioni Web.

parte alcune differenze, un vero e proprio database SQL Server, il che rende molto semplice la migrazione e rende allo stesso tempo molto piatta la curva di apprendimento per chi già conosce SQL Server; ulteriori informazioni possono essere recuperate in sezione 7.1.

### 3.1.3 Windows Azure AppFabric

Per quanto Windows Azure ponga delle astrazioni verso gli sviluppatori, tuttavia esse possono risultare non sufficienti; o meglio, è possibile astrarre ancora i servizi, semplificando ancora di più la vita a buona parte degli sviluppatori. Windows Azure AppFabric ha esattamente questo ruolo, porre un ulteriore livello di astrazione, offrendo utilità aggiuntive per la comunicazione (Service Bus), il controllo dell'accesso (Access Control) e la disponibilità dei dati usati più frequentemente (Caching). In figura 3.4 è presente una visione schematica del servizio.

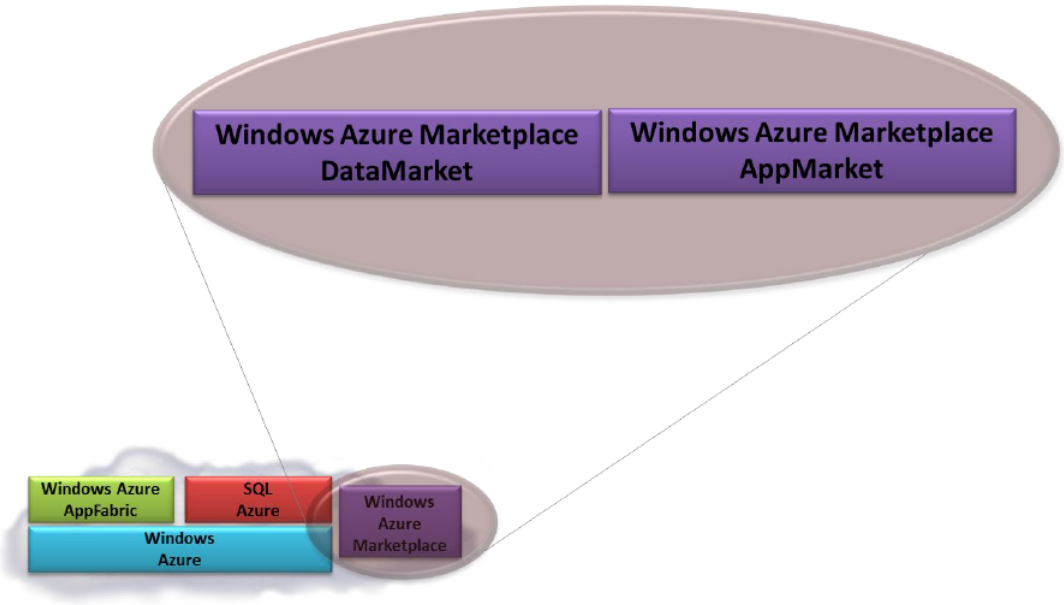



Figura 3.5: Il servizio Marketplace è composto da due "mercati": DataMarket ed AppMarket.

### 3.1.4 Windows Azure Marketplace

Trattato più ampiamente in sezione 7.2, Windows Azure Marketplace offre, come si può notare in figura 3.5, due tipi di *mercati*: uno orientato ai dati (DataMarket) e uno orientato alle applicazioni (AppMarket). In particolare, il DataMarket ha come obiettivo la compravendita di insiemi di dati, ad esempio per fini statistici, mentre l'AppMarket consente di vendere applicazioni Web memorizzate nella cloud.

## 3.2 Windows Azure SDK

Sia per ovvi motivi economici, sia per altrettanto ovvi motivi pratici, non è assolutamente consigliabile sviluppare *direttamente* sulla cloud. Porre sulla piattaforma software che consuma molte più risorse del dovuto a causa di errori di programmazione può risultare in perdite economiche più o meno consistenti; inoltre, non è per nulla pratico eseguire il deployment a ogni test.

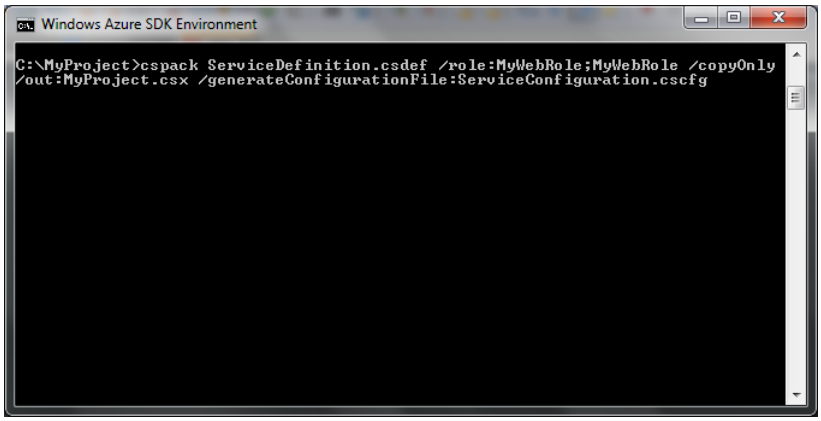



Figura 3.6: Come si presenta il Command Prompt.

Pertanto, l'SDK per Windows Azure consente di far partire in locale alcuni servizi offerti dalla piattaforma; tuttavia, si faccia attenzione al fatto che tale SDK non contiene al suo interno gli strumenti per avviare una cloud locale, ma esso offre semplicemente un'implementazione delle specifiche in modo tale da consentire il testing locale delle applicazioni. A titolo d'esempio, si vedrà tra poco che buona parte dei servizi di storage viene emulata attraverso un database SQL Server locale, cosa che ovviamente non succede sulla cloud. Infine, si noti bene che non per tutti i servizi viene fornito un emulatore, ma ciò viene fatto solo per Windows Azure Compute (sezione 3.2.2) e Storage (3.2.3).

SQL Azure non necessita di avere un emulatore in locale perché ciò non è necessario; infatti, come si vedrà in sezione 7.1, esso è, a parte piccole differenze, totalmente analogo a SQL Server. Pertanto, applicazioni sviluppate usando astrazioni come ADO.NET o ODBC possono essere tranquillamente essere provate con un database locale e poi, con un semplice cambio della stringa di connessione, possono cominciare a comunicare con SQL Azure.

### 3.2.1 Command Prompt

Per svariati motivi, potrebbe essere necessario dovere preparare il pacchetto per il deployment da riga di comando, piuttosto che utilizzando le comodità offerte da software come Visual Studio, sia per motivi di prestazioni, sia

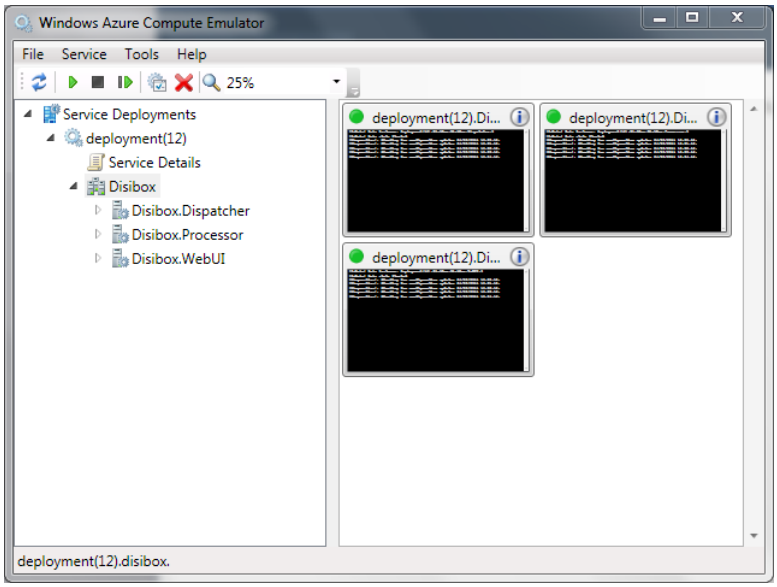



Figura 3.7: Interfaccia grafica del Compute Emulator.

per mancanza di una vera e propria interfaccia grafica. Il programma che si occupa dell'impacchettamento è *cspack*, ma una guida dettagliata sul suo uso esula dallo scopo di questa relazione; se si ha interesse a scoprire di più sull'argomento, si veda [11].

### 3.2.2 Compute Emulator

Il Compute Emulator ha lo scopo di mettere in esecuzione i ruoli contenuti in un certo deployment; una volta eseguiti, l'interfaccia permette di interagire con essi, dando la possibilità di verificarne lo stato e di visualizzarne l'output in finestrelle simili a console. Inoltre, l'interfaccia permette di riavviare (o fermare) tutte le istanze dei ruoli.

### 3.2.3 Storage Emulator

Esso si occupa di far girare in locale i tre servizi legati alla memorizzazione, cioè, i blob, le tabelle e le code. L'interfaccia grafica per controllare l'emulatore, come si può vedere in figura 3.8, è molto scarna e non molto ricca di

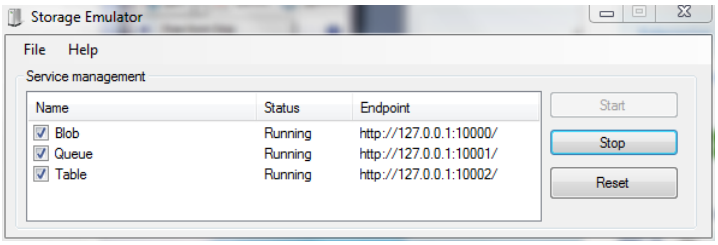



Figura 3.8: Interfaccia grafica dello Storage Emulator.

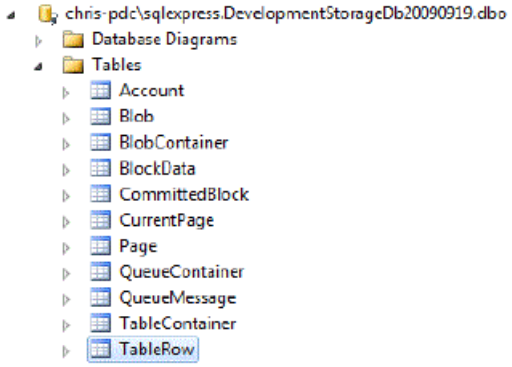



Figura 3.9: Tabelle contenute nel database SQL Server usato per emulare Windows Azure Storage.

funzionalità; infatti, per ciascun servizio, sono solamente rese disponibili le seguenti operazioni:

- Verifica dello stato del servizio (fermato, in avvio, attivo);
- Avvio/arresto del servizio;
- Cancellazione di tutti i dati memorizzati.

Quindi, l'interfaccia offerta lascia fuori operazioni importanti come la verifica del *contenuto* dei servizi di memorizzazione e la possibilità di manipolare i dati aggiungendo, togliendo o modificando elementi. Uno sviluppatore non può accontentarsi di uno strumento così poco potente e potrebbe trovare più utile e produttivo lo strumento presentato nella sezione successiva, Azure Storage Explorer.

Come mostrato in figura 3.9, il servizio di memorizzazione viene emulato in locale con delle tabelle su un database SQL Server locale; ciò, almeno a

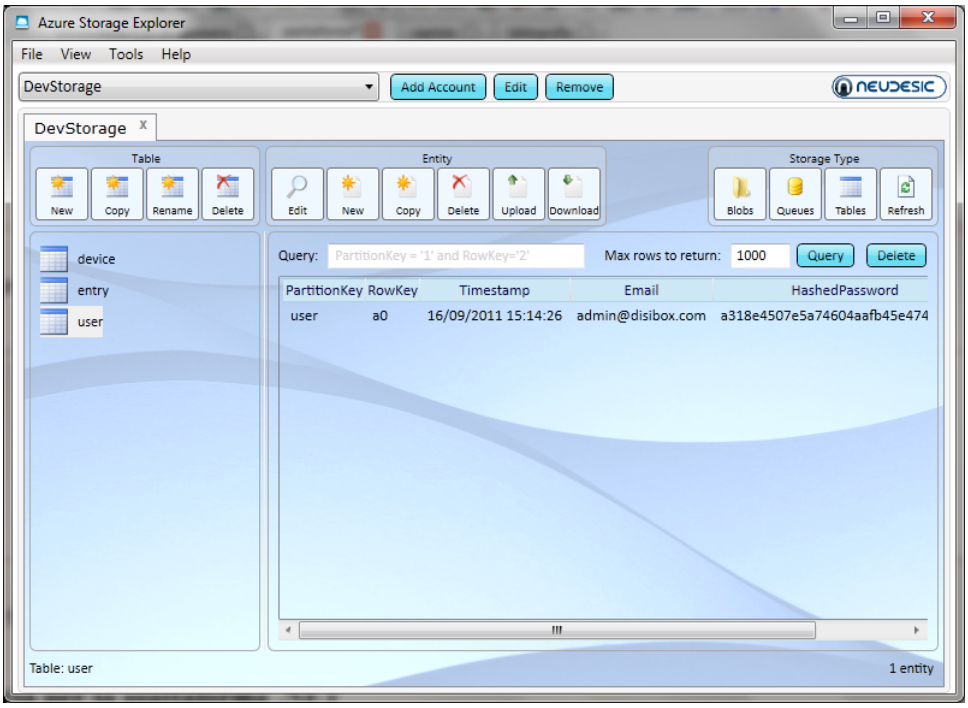



Figura 3.10: Interfaccia grafica offerta da Azure Storage Explorer per la gestione delle tabelle.

livello teorico, consente allo sviluppatore di controllare che effettivamente il salvataggio dei dati avvenga come da lui programmato. Tuttavia, come si può facilmente intuire, l'operazione non è molto pratica a lungo termine e si preferiscono altre vie per quel tipo di verifica.

### 3.2.4 Azure Storage Explorer

Come affermato nella sezione precedente, lo Storage Emulator fornito con l'SDK ha un numero troppo piccolo di funzionalità per poter essere soddisfacente per uno sviluppatore software. Pertanto, è conveniente (se non necessario) appoggiarsi allo strumento Azure Storage Explorer, software open source e scritto da Neudesic[1]; esso fornisce tutte le caratteristiche mancanti all'esploratore di default, cioè, la manipolazione e la visualizzazione dei con-

[1]Neudesic è una società partner tecnologico di Microsoft, specializzata nella gestione di soluzioni cloud.[13]

tenuti. Volendo riassumere con poche parole ciò quali sono le funzionalità di Azure Storage Explorer, si può dire che esso offra un corrispondente *grafico* a (quasi) ogni metodo nelle interfacce dei servizi.

Ad esempio, preso un blob, è possibile visualizzarne il contenuto, con più o meno dettaglio a seconda del tipo del contenuto stesso: se esso è un file di testo, lo si potrà editare, se è un'immagine, la si potrà visualizzare, mentre, se è un video, lo si potrà riprodurre. Indipendentemente dal tipo del contenuto, l'interfaccia consente, per ogni blob, di modificarne le proprietà e i metadati, oltre a dare la possibilità di eseguire operazioni CRUD (Create, Read, Update, Delete) su di esso.

Passando alle tabelle, lo strumento in questione permette di modificare i contenuti delle tabelle stesse ed eseguire query su di esse. Analogamente, sono consentite le operazioni di manipolazione anche sulle code, come l'inserimento e la cancellazione dei messaggi, la manipolazione del loro contenuto, nonché delle loro proprietà (come il tempo di fine dei messaggi).

Inoltre, essendo uno strumento open source, Azure Storage Explorer risulta molto utile come punto di riferimento *pratico* per chi si accinge ad usare le librerie .NET per interfacciarsi con Azure. Il codice sorgente, l'installatore e altre informazioni possono essere ottenuti su [12].

## 3.3 Libreria per la piattaforma .NET

Come detto in sezione 3.1.1, si può accedere ai servizi di memorizzazione utilizzando un'interfaccia REST; pertanto, lo scopo della libreria è farsi carico di costruire tali chiamate via HTTP, mostrando agli sviluppatori una comoda e abituale interfaccia object oriented. In pratica, la libreria offre un sistema di oggetti che mappano esattamente i servizi offerti da Windows Azure Storage, ma, nell'implementazione di tali oggetti, si nascondono tante chiamate all'interfaccia REST, usando il protocollo OData se necessario per realizzare le interrogazioni.

A livello qualitativo, tuttavia, la libreria sembra, agli occhi di chi sta scrivendo questa relazione, immatura, e, per dirla in modo molto ironico, leggermente "schizofrenica". Come si vedrà andando avanti con la lettura, la

**Algoritmo 3.1** Segnature di metodo poco convincenti presenti nella libreria .NET di Azure.

```
// Metodi della classe TableServiceContext, usata per interagire
// con le tabelle;  essi richiedono una stringa per identificare
// una tabella, e non pongono vincoli sull'oggetto da manipolare
// (quando esso, come si vedra', ha dei vincoli da rispettare).
public void AddObject(string entitySetName, Object entity) { ... }
public void DeleteObject(Object entity) { ... }
public void UpdateObject(Object entity) { ... }
// Costruttore di CloudQueueMessage, messaggio da porre nelle code;
// la stringa non puo' superare, per contratto, gli 8KB,
// ma il costruttore, in base alla documentazione,
// non si occupa di controllare che cosi' davvero sia.
public CloudQueueMessage(string content) { ... }
```

**Algoritmo 3.2** Molteplici modi per ottenere un riferimento allo stesso blob.

```
// Primo metodo: usare il client per il servizio blob.
var blob = blobClient.GetBlobReference("container/blob");
// Secondo metodo: usare il contenitore del blob.
blob = blobContainer.GetBlobReference("blob");
// Terzo metodo: usare il costruttore del blob.
blob = new CloudBlob(blobEndpointUri + "/container/blob", credentials);
// NOTA: Tutti i metodi sopra esposti fanno ottenere la stessa cosa.
```

libreria offre talvolta chiamate di funzione non fortemente tipate, che molto spesso si basano sull'uso di semplici stringhe per identificare oggetti e punti di memorizzazione; inoltre, spesso la libreria delega l'intercettazione degli errori alla cloud stessa, quando si potrebbero intercettare prima, nella libreria stessa. A titolo d'esempio, si osservino in listato 3.1 le segnature di alcuni metodi estratti dalla libreria.

Si è poi detto, in modo un po' azzardato, che la libreria soffre di "schizofrenia", perché esistono fin troppi modi per eseguire la medesima operazione; a titolo d'esempio, si veda listato 3.2, dove si mostrano tre modi per ottenere un riferimento allo stesso blob.

Tutto sommato, le critiche sopra esposte non devono in nessun modo *in-*

*disporre* un futuro utente della libreria .NET. L'immaturità di tale libreria può essere semplicemente dovuta alla giovane età di Azure stesso; inoltre, alcuni problemi di inconsistenza o poca praticità possono essere semplicemente risolti con codice wrapper, come si esporrà dove opportuno nei capitoli successivi.

In generale, si ricorda che è opportuno, nei casi di servizi aventi interfacce proprietarie come Windows Azure, porre un ulteriore livello di astrazione a separare la propria applicazione dall'interfaccia proprietaria, in modo tale che, dovendo cambiare servizio, si debba cambiare solo il livello di astrazione.

#### Note per l'uso

Per poter provare gli esempi d'uso della libreria proposti nei capitoli successivi, è necessario aggiungere da Visual Studio la referenza alle seguenti librerie:

- `Microsoft.WindowsAzure.StorageClient`
- `System.Data.Service.Client`

In caso un esempio richieda librerie aggiuntive, sarà puntualmente annotato nella descrizione dell'esempio stesso. Infine, la stringa di connessione utilizzata nel corso degli esempi (e da utilizzare in generale per accedere allo storage emulato) è "`UseDelevelopmentStorage=true`".

## 3.4 Libreria per la piattaforma Java

WindowsAzure4j, sviluppata da Soyatec[2] in collaborazione con Microsoft, è la libreria che consente ad applicazioni Java di interagire con Windows Azure Storage; tale libreria offre le medesime funzionalità della controparte .NET, integrandosi bene all'interno dell'IDE Eclipse.

Per quanto la mancanza di LINQ, in un ambiente altamente orientato alla gestione dei dati, la faccia sembrare meno potente, in realtà la libreria

[2]Come Neudesic, Soyatec è una società parter tecnologico di Microsoft; in particolare, essa offre prodotti basati su Java e orientati al business come eUML2 ed eBPMN.

**Algoritmo 3.3** La libreria Java offre (giustamente) un solo modo per creare un blob generico.

```
1 // Di seguito, si ha l'unico modo per creare un blob generico.
2 IBlobContainer container = blobClient.createContainer("container");
3 IBlob blob = container.getBlobReference("blob");
```

Java conserva lo stesso potere funzionale, risultando, dal punto di vista dello scrittore di questa relazione, di *qualità* superiore rispetto a quella per la piattaforma .NET. Infatti, si nota una certa coerenza nelle segnature dei metodi e, in particolare, una corretta limitazione dei modi d'uso della libreria. Ora, con alcuni brevi esempi, si cercherà di spiegare ciò che si è appena affermato.

Per cominciare, si prenda listato 3.2 e lo si confronti con listato 3.3, si noterà subito come da ben tre modi per ottenere lo stesso blob generico, si passi (giustamente) ad un solo modo. Inoltre, a parte la trascurabile piccolezza che le interfacce della libreria hanno la *I* iniziale, come da tradizione .NET ma contro gli "usi e costumi" di Java, la libreria presenta giustamente delle interfacce verso il cliente, e non delle classi pure come la libreria .NET; in pratica, la libreria Java è costruita secondo un modello per cui si hanno le implementazioni delle *factory* come dipendenza forte, mentre non vi è, grazie alle interfacce, alcuna dipendenza nei confronti delle implementazioni di elementi come i blob.

### Note per l'uso

Istruzioni dettagliate su come installare il supporto per Eclipse e come aggiungere la libreria a un dato progetto possono essere trovate su [14]: nella sezione "Learn" si cerchi il "Lab 0", laboratorio dedicato per l'appunto all'impostazione di un progetto Eclipse. In particolare, si faccia molta attenzione al fatto che la libreria ha un gran numero di dipendenze (la cui mancanza, tristemente, viene segnalata solo a tempo di esecuzione), e che talvolta i collegamenti indicati sul sito di WindowsAzure4j non funzionano o puntano a librerie datate.

**Algoritmo 3.4** Classe contenente le "informazioni" necessarie per accedere all'emulatore dello storage attraverso la libreria Java.

```
1 final class Settings {
2     public static final URI BLOB_ENDPOINT;
3     public static final URI QUEUE_ENDPOINT;
4     public static final URI TABLE_ENDPOINT;
5
6     public static final String DEV_ACCOUNT_NAME;
7     public static final String DEV_ACCOUNT_KEY;
8
9     // Blocco usato per questioni di formattazione nella relazione,
10    // non e' realmente necessario inizializzare i campi qui.
11    static {
12        BLOB_ENDPOINT  = URI.create("http://127.0.0.1:10000");
13        QUEUE_ENDPOINT = URI.create("http://127.0.0.1:10001");
14        TABLE_ENDPOINT = URI.create("http://127.0.0.1:10002");
15
16        DEV_ACCOUNT_NAME = "devstoreaccount1";
17        DEV_ACCOUNT_KEY = "Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsu" +
18                          "Fq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==";
19    }
20 }
```

Per accedere allo storage emulato localmente, non si può utilizzare la stessa stringa di connessione presentata in sezione 3.3 per la libreria .NET; infatti, sono necessarie delle particolari credenziali per il login e occorre scrivere "a mano" gli indirizzi di ciascun tipo di storage. Pertanto, in listato 3.4 si presenta una classe che raccoglie tutte le informazioni necessarie per avviare una connessione con l'emulatore dello storage.

# Capitolo 4

# Azure Storage - Blob

Il blob è l'unità di storage concettualmente più *semplice* offerta dalla piattaforma; esso consente di memorizzare una grande quantità di dati binari non strutturati, come tracce audio e video, immagini, documenti di testo. L'account di storage può contenere diversi *contenitori*, all'interno dei quali possono essere inseriti più blob. Ciascun blob può avere dimensioni notevoli (sino a un terabyte) ed avere associati dei metadati, come l'autore di un video o la macchina con cui una certa foto è stata scattata; infine, è possibile salvare un'istantanea di un dato blob, allo scopo di effettuare operazioni di backup.

Sono disponibili due tipi di blob, il block blob ed il page blob; ciascuno offre caratteristiche particolari, complementari alle funzionalità dell'altro tipo. In particolare, come si vedrà meglio più avanti, un page blob può essere utilizzato per offrire un file system NTFS alle applicazioni operanti all'interno di Windows Azure.

Al fine di garantire un rapido accesso degli utenti ai contenuti dei blob, la piattaforma offre il servizio Content Delivery Network (CDN, si veda sezione 4.4), il quale consente di mettere i blob nella cache di un centro dati in prossimità degli utenti stessi.

## 4.1 Perché usare i blob?

Blob è l'acronimo di "Binary Large OBject" ed è un termine preso in prestito dal vocabolario dei database relazionali, dove descrive il tipo di una colonna atta a memorizzare dati binari (come un'immagine o un file MP3). Un cliente potrebbe trovare interessante l'uso del servizio dei blob perché esso offre una via per salvare file in modo durabile e scalabile; in particolare, un amministratore di sistema potrebbe porsi le seguenti domande prima di scegliere un servizio di storage dei file:

- Si ha abbastanza spazio per salvare tutti i file di cui si necessita?
- Se lo spazio finisse, si potrebbe incrementare la dimensione della memoria?
- I dati sono (almeno) duplicati da qualche parte?
- In caso di crash, i dati vengono preservati?
- Il carico di lavoro di un'unità di memoria è bilanciato?

Nel caso del servizio dei blob, la risposta alle domande appena scritte è positiva. L'*elasticità* della cloud offre uno spazio di memorizzazione pressoché illimitato, e le politiche di gestione dei dati della piattaforma di Windows Azure garantiscono gli altri punti. Infatti, come per tutti gli altri tipi di storage offerti, anche per i blob vale il principio che, per ciascun blob, vi sono almeno tre copie di esso, sparse se possibile su più centri dati.

Il servizio dei blob offre delle interfacce RESTful, consentendo l'accesso ai blob anche da browser. A livello di performance, si ha che la velocità di accesso fuori dai centri dati non è molto alta, mentre è, come ci si può aspettare, incredibilmente alta all'interno di tali centri; è tanto veloce che il trasferimento di gigabyte di dati può avvenire in pochi secondi.

## 4.2 Elementi costitutivi del servizio

Si vedrà ora una sintetica carrellata degli elementi base del servizio, cioè, i contenitori e i diversi tipi di blob.

### 4.2.1 Blob container

Per quanto sia possibile memorizzare i blob direttamente alla radice del servizio, è decisamente conveniente, sia per questioni di modularità logica, sia per questioni pratiche, inserire i blob all'interno di contenitori. Essi possono (e devono) avere un nome, e non hanno vincoli sulla dimensione massima raggiungibile; a livello pratico, è utile sapere che è possibile impostare il livello di visibilità di un contenitore: essi possono privati, completamente pubblici, pubblici soltanto il lettura.

Si vedrà che vi sono diversi tipi di blob, ma un contenitore ne può memorizzare di diversi tipi, contemporaneamente; salvarne uno di un certo tipo non vincola i blob inseribili in futuro.

### 4.2.2 Blob "generici"

Il termine "blob generico" è un termine coniato dall'autore di questa relazione, e indica le caratteristiche condivise dai block blob e dai page blob (presentati nelle sezioni successive). Ciascun blob ha associate una serie di proprietà (come la dimensione, il tipo MIME dei contenuti) e una serie di metadati, proprio come i file residenti sui comuni file system. Utilizzando i metadati è possibile preservare informazioni *accessorie* come le informazioni contestuali di un file MP3, o le caratteristiche di una macchina fotografica che ha scattato una certa foto.

### 4.2.3 Block blob

I block blob sono ottimizzati per lo streaming (upload e download) in quanto, come potrebbe suggerire il nome, sono composti da *blocchi* aventi dimensione non superiore ai 4MB; un *blocco* rappresenta la quantità massima trasferibile in una singola operazione ed è identificato da codice unico. I block blob si caricano sulla cloud blocco per blocco; in particolare, prima si caricano i blob, e poi se ne effettua il commit. Essendo suddiviso in blocchi, tale tipo di blob consente di riprendere un download fallito o interrotto dall'ultimo blocco

inviato, senza inviare di nuovo il blob intero. Attualmente, la dimensione massima di block blob è di 200GB, cioè, 50000 blocchi da 4MB.

### 4.2.4 Page blob

A differenza dei block blob, i page blob sono ottimizzati per l'accesso casuale; anche in questo caso, il nome suggerisce la loro struttura: infatti, essi sono composti da *pagine*, ciascuna pagina riferita in base allo scostamento dall'inizio del blob. I page blob possono raggiungere dimensioni superiori ai block blob, in quanto è consentito loro *pesare* sino ad un 1TB; non vi sono vincoli sulle dimensioni delle singole pagine, purché non siano più grandi del più grande page blob creabile (cioè, 1TB).

## 4.3 Azure Drive

Come già accennato, i page blob presentati nella sezione precedente possono essere usati per fornire un file system NTFS alle applicazioni operanti sulla piattaforma. In particolare, tutto ciò viene realizzato inserendo un disco rigido virtuale (VHD, Virtual Hard Drive), avente NTFS come file system, all'interno di un page blob. Quindi, il VHD può essere montato e vi si può accedere come se esso fosse un disco locale; questo servizio può avere una importanza determinante nel facilitare il trasferimento di applicazioni esistenti verso la cloud, in quanto esse non hanno bisogno di essere riprogettate per usare i nuovi tipi di storage ma possono continuare, con pochi cambiamenti al codice, a utilizzare il vecchio approccio basato sull'uso dei file e del file system.

Tuttavia, non è tutto così perfetto, dato che questo servizio ha tre grandi limitazioni:

- Il VHD può essere montato solo dalle applicazioni ospitate all'interno di Windows Azure;
- Solo i ruoli possono montare un VHD;
- Soltanto un ruolo alla volta può accedere a un dato VHD.

Partendo dalla prima limitazione, si cercheranno di giustificare le altre. La possibilità di montare un page blob come se fosse un disco NTFS è data da un particolare driver, scritto apposta per il sistema operativo che sta dietro a ciascun ruolo (evidentemente gli ingegneri Microsoft non hanno molta fantasia, in quanto tale sistema operativo è denominato... Windows Azure); pertanto, già da qui si capisce che un'applicazione operante fuori dalla cloud (su Windows 7, ad esempio) non potrà montare un drive simile proprio per la mancanza del driver apposito. Solo i ruoli, che "girano" sul sistema operativo Windows Azure possono montare quel drive, anche perché è richiesto l'accesso allo storage locale (a disposizione soltanto dei ruoli) per poter garantire una cache al drive. Infine, secondo quanto esposto in [18], si ha che solo un ruolo alla volta può accedere a tale servizio.

## 4.4 Content Delivery Network

Se gli utenti di un certo servizio sono geograficamente sparsi, può avere senso usare una Content Delivery Network (CDN) per migliorare l'esperienza dell'utente, e non solo. Si vedrà ora una visione generale del servizio, ed alcuni cenni su come integrarlo con il servizio dei blob.

### 4.4.1 Cos'è una CDN?

Una CDN è un grande insieme di Web server distribuiti in tutto il mondo. Quando un utente effettua una richiesta di un file alla CDN, la rete stessa si occupa di individuare il centro dati più vicino a quell'utente; servendo la richiesta da quel centro, le performance saranno necessariamente migliori. In generale, la CDN può servire tutti quei file *statici*, il cui contenuto non cambia rapidamente, come i file PDF e le immagini.

### 4.4.2 Vantaggi alle prestazioni

Usare una CDN può migliorare le prestazioni del proprio servizio, senza la necessità di radicali cambiamenti alla sua architettura. In particolare, i benefici dell'uso di tale rete sono due:

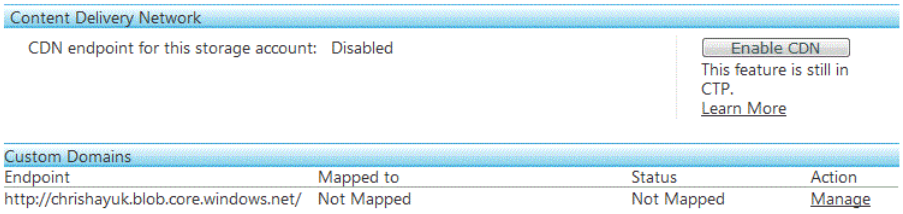



Figura 4.1: Abilitazione della CDN per un servizio dei blob.

- Riduzione del carico di lavoro sui server dell'applicazione Web;
- Aumento delle performance per l'utente.

Il primo *effetto* è dovuto al fatto che, memorizzando i contenuti statici su una CDN, la loro gestione viene delegata alla rete stessa e il server dell'applicazione non se ne deve più occupare. Questo fatto comporta una riduzione delle connessioni attive, dell'uso della CPU e del traffico di rete sul server; in pratica, tutto questo si traduce in minori costi di gestione del server.

Invece, il secondo *effetto*, le prestazioni migliori per l'utente, sono causate dal modo in cui i browser gestiscono il download delle pagine: essi non scaricano più di due contenuti contemporaneamente dalla stessa fonte; pertanto, spostare certi contenuti su una CDN (avente un dominio differente dall'applicazione) consente loro di scaricare le pagine più rapidamente.

### 4.4.3 I blob e la rete CDN

Windows Azure offre una rete CDN, ma va ricordato che ve ne sono altre sul mercato, come Akamai ed Amazon CloudFront. Come ci si può aspettare, però, la CDN di Windows Azure è la più semplice da integrare con il servizio dei blob.[18]

Infatti, si può rendere un servizio dei blob *coperto* dalla CDN semplicemente abilitando il servizio dal portale di Windows Azure, come mostrato in figura 4.1; una volta abilitato, i blob contenuti nei container pubblici saranno sulla CDN nel giro di un'ora.[18]

| Costi relativi ai dati memorizzati |
|---|
| 0.15$ per gigabyte salvato ogni mese |
| 0.01$ ogni 10000 transazioni con lo storage |

Tabella 4.1: Costi pertinenti la quantità di dati memorizzati.

## 4.5 Costo del servizio

In questa sezione si esporranno come vengono stabiliti i costi del servizio di storage in generale: la logica è la medesima per i blob, le tabelle e le code. Nel servizio, si paga secondo due fattori:

- La memoria occupata;
- La quantità di trasferita.

Per ciascuno dei due fattori, si vedrà ora sinteticamente come si gestiscano i costi. Tutti i dati sono stati ricavati da [17].

### 4.5.1 Quantità di dati memorizzati

Si paga, come si può immaginare, in base allo spazio che si occupa, ma non solo. Infatti, per ottenere il prezzo finale si calcola quanto spazio (in gigabyte) si è occupato nel corso di un mese. Ad esempio, se si sono memorizzati 10GB per quindici giorni (metà mese), ma poi sono stati rimossi, si pagherà soltanto per 5GB mensili; al contrario, se quei 10GB sono rimasti memorizzati per tutti i trenta giorni, allora si pagherà per 10GB mensili. Inoltre, vi sono dei costi anche per le transazioni, cioè, per le varie operazioni di manipolazione dello storage (aggiunta, modifica, rimozione). Tutti i costi relativi a questa parte sono riassunti in tabella 4.1.

### 4.5.2 Quantità di dati trasferiti

Quando si dice ”dati trasferiti”, si sottintende all’esterno dei centri dati; infatti, i trasferimenti all’interno di un singolo centro non devono essere pagati. Quindi, in tabella 4.2 si riportano i costi relativi al trasferimento dei dati esternamente ai centri dati.

| Costi relativi ai dati trasferiti |
|---|
| 0.15$ ogni gigabyte, per il Nord America e l'Europa |
| 0.20$ ogni gigabyte, per l'Asia |

Tabella 4.2: Costi pertinenti la quantità di dati trasferiti.

**Algoritmo 4.1** Creazione di un blob container.

```
1 var connectionString = Settings.Default.DataConnectionString;
2 var storageAccount = CloudStorageAccount.Parse(connectionString);
3
4 var blobEndpointUri = storageAccount.BlobEndpoint.ToString();
5 var containerUri = blobEndpointUri + "/" + containerName;
6 var credentials = storageAccount.Credentials;
7 var container = new CloudBlobContainer(containerUri, credentials);
8
9 container.CreateIfNotExist(); // Effettiva creazione del container.
```

## 4.6 Esempi d'uso

Seguono ora alcuni esempi di come si può usare la libreria .NET per effettuare alcuni compiti molto semplici come la creazione di un container e la manipolazione dei vari tipi di blob, nonché alcuni cenni su Azure Drive.

### 4.6.1 Creazione di un container

I passi necessari per la creazione di un container, esposti in listato 4.1, sono pochi e semplici. In particolare, sono necessari l'URI del container (una stringa avente il seguente formato: `$URI_Servizio_Blob/$Nome_Container`) e le credenziali di accesso (ricavabili da un oggetto di tipo `CloudStorageAccount`).

### 4.6.2 Blob "generico"

In listato 4.2 si mostra l'uso delle proprietà e dei metadati associati a ciascun blob; in particolare, vengono usati la lunghezza (o dimensione) del blob (riga 4), insieme ad alcuni metadati (da riga 6 in poi). I metadati, a livel-

**Algoritmo 4.2** Esempio d'uso di un blob "generico".

```
1  var blob = container.GetBlobReference(container.Uri + "/blob");
2  var streams = CreateSimpleStreams(128);
3  blob.UploadFromStream(streams[0]);
4  Debug.Assert(blob.Properties.Length == streams[0].Length);
5
6  blob.Metadata.Add("Author", "Pino");
7  blob.Metadata.Add("Date", "Duemilamai");
8  Debug.Assert(blob.Metadata.Count == 2);
9  Debug.Assert(blob.Metadata.Get("Author") == "Pino");
10 Debug.Assert(blob.Metadata.Get("Date") == "Duemilamai");
```

lo di programmazione, possono essere considerati come una sorta di mappa chiave/valore.

### 4.6.3 Block blob

L'uso dei block blob non è propriamente intuitivo, e in listato 4.3 se ne fa un semplice esempio. La difficoltà consiste nel fatto che occorre caricare i vari blocchi che comporranno il blob uno alla volta, ricordandosi per ciascuno l'id assegnatogli; in seguito al caricamento, il blob non sarà pronto all'uso finché non si sarà fatto il commit di tutti i blocchi, come effettuato in riga 14.

### 4.6.4 Page blob

Analogamente ai block blob, anche per i page blob occorre caricare i contenuti pagina per pagina, ma non se ne deve effettuare il commit. Infatti, come si fa vedere in listato 4.4, il page blob viene prima allocato, ed esso risulta vuoto; in seguito, si potranno caricare i dati, suddividendoli per "pagine" di memoria.

### 4.6.5 Azure Drive

Listato 4.5 mostra come si può creare e montare un disco virtuale basato su NTFS, usando come *dispositivo di memorizzazione* un page blob. Per

**Algoritmo 4.3** Esempio d'uso di un block blob.

```
1  var blockBlob = container.GetBlockBlobReference(container.Uri + "/blockblob");
2  var streams = CreateSimpleStreams(256);
3  // Lista con gli id dei blocchi, necessaria per il commit.
4  var blockIds = new List<string>();
5  for (var i = 0; i < streams.Count; ++i)
6  {
7      // L'id di ciascun blocco deve essere codificato in base 64.
8      var blockId = EncodeTo64("b" + i);
9      blockBlob.PutBlock(blockId, streams[i], null);
10     blockIds.Add(blockId);
11 }
12 // Esegue il commit della lista di blocchi; la sua invocazione
13 // e' necessaria al fine del completamento dell'operazione.
14 blockBlob.PutBlockList(blockIds);
15 // Tramite il metodo qui sotto si "aggiornano" gli attributi del blob.
16 blockBlob.FetchAttributes();
17 Debug.Assert(blockBlob.Properties.Length == streams.Count*streams[0].Length);
18 // Verifica che i blocchi siano coerenti con quanto fatto finora.
19 var blocks = blockBlob.DownloadBlockList();
20 Debug.Assert(blocks.Count() == streams.Count);
21 Debug.Assert(blocks.Count(b => !b.Committed) == 0);
```

**Algoritmo 4.4** Esempio d'uso di un page blob.

```
1  var pageBlob = container.GetPageBlobReference(container.Uri + "/pageblob");
2  var streams = CreateSimpleStreams(1024 * 1024); // 1MB
3  // Si crea un page blob adatto a contenere tutti gli stream.
4  pageBlob.Create(streams.Count * streams[0].Length);
5  for (var i = 0; i < streams.Count; ++i)
6      // Il secondo parametro, l'offset in byte, deve essere multiplo di 512.
7      pageBlob.WritePages(streams[i], i * streams[0].Length);
8  // Tramite il metodo qui sotto si "aggiornano" gli attributi del blob.
9  pageBlob.FetchAttributes();
10 Debug.Assert(pageBlob.Properties.Length == streams.Count * streams[0].Length);
```

**Algoritmo 4.5** Esempio d'uso di Azure Drive.

```
1  // Creazione del page blob "contenitore".
2  var pageBlob = container.GetPageBlobReference(container.Uri + "/pageblob");
3  pageBlob.Create(128 * 1024 * 1024); // 128MB
4  // Creazione e montaggio del drive.
5  var drive = CreateCloudDrive(pageBlob.Uri);
6  drive.Create(64); // Va chiamato una volta sola nella vita di un drive.
7  var driveLetter = drive.Mount(32, DriveMountOptions.None);
8  // Prove di scrittura e lettura.
9  var testFilename = driveLetter + "\\test.txt";
10 var testContent = "We are testing Azure Drive!";
11 File.WriteAllText(testFilename, testContent);
12 var readContent = File.ReadAllText(testFilename);
13 Debug.Assert(readContent == testContent);
14 // Smontare il drive e' un'ottima abitudine.
15 drive.Unmount();
```

poter usare i metodi e gli oggetti presentati nell'esempio, occorre aggiungere un riferimento alla libreria `Microsoft.WindowsAzure.CloudDrive`, oltre ai riferimenti di cui si è parlato in sezione 3.3. Si noti come, una volta montato, Azure Drive possa essere usato nello stesso identico modo in cui si usano i drive normalmente; inoltre, si ricorda che per provare il seguente esempio si deve porre il codice all'interno di un *ruolo* di Windows Azure e si deve avere definito un *LocalStorage* nel file `ServiceDefinition.csdef`.

# Capitolo 5

# Azure Storage - Tabelle

Questo servizio porta un nome che tende, di per se, a dare un'idea sbagliata di quale sia la reale offerta. Infatti, spesso si associa il termine *tabella* alle tabelle dei database relazionali, mentre, in questo caso, le tabelle offerte dal servizio hanno una connotazione completamente diversa. Infatti, esse possono essere viste come una collezione di entità di tipo riga, dove ciascuna *riga* può essere composta da un massimo di 255 *colonne* (o, come vedremo meglio in seguito, proprietà); tuttavia, a differenza delle tabelle *classiche*, non vi è alcuno schema a imporre una struttura comune a tutte le righe di una data tabella.

Per farsi un'idea di come funzioni il tutto, si può pensare alle tabelle proposte dai fogli elettronici come Microsoft Excel e LibreOffice Calc, in quanto esse seguono un modello analogo: ciascuna riga può avere un numero arbitrario di colonne e celle appartenenti alla stessa colonna possono contenere diversi tipi di dato.

Pertanto, vista la struttura molto flessibile delle tabelle di Windows Azure, non è possibile effettuare operazioni molto usate sulle tabelle dei database relazionali, come impostare i vincoli di chiave esterna e i trigger, effettuare join, invocare le stored procedure, e, in generale, eseguire ogni tipo di processing lato storage. In ogni caso, il servizio supporta ovviamente le operazioni di base: inserimento, modifica, cancellazione e selezione. Se si ha necessità di usare le funzioni non offerte dal servizio, ci si può appoggiare a SQL Azure

(si veda sezione 7.1), il quale garantisce un completo database relazionale ospitato nella cloud.

## 5.1 Perché abbandonare l'approccio SQL?

Molti lettori si saranno probabilmente già chiesti perché, nonostante le notevoli comodità fornite da un server SQL, Microsoft abbia deciso di creare un servizio che *somiglia* soltanto a un database di quel tipo, gettando via caratteristiche interessanti come la forzatura dello schema e meccanismi oramai di uso comune come le chiavi esterne e i trigger. La risposta breve è, semplicemente, per questioni di scalabilità. Infatti, a meno di usare complicati sistemi, non è facile rendere scalabile un database SQL; l'unica via *facile* è potenziare l'hardware, ma questo spesso non riesce a tenere testa alla crescita della domanda. In generale, questo andrebbe contro uno dei capisaldi del cloud computing: la potenza e la scalabilità, sia verso l'alto sia verso il basso, derivano dalla presenza di tante piccole istanze, non dalla presenza di una sola con grandi capacità.

## 5.2 Caratteristiche notevoli del servizio

Oltre ad essere un servizio di memorizzazione altamente scalabile, le tabelle possono vantare due caratteristiche di interesse: la replicazione dei dati e la lettura garantita dopo il commit di una transazione.

Per quanto riguarda la replicazione dei dati, si ha che il contenuto di ogni tabella è replicato almeno tre volte su server differenti, appartenenti a centri dati differenti. Così facendo, i dati sono disponibili sia in caso di fallimento di un server, sia in caso di fallimento dell'intero centro dati. Tuttavia, se il servizio si fermasse qui, la caratteristica di triplicare i dati potrebbe rivelarsi più svantaggiosa che vantaggiosa: se si effettua una scrittura su un server A, e si fa poco dopo la lettura su un server B, sapendo che A e B tengono copie della stessa tabella, B conterrà i cambiamenti effettuati su A?

A risolvere questo problema, interviene la seconda caratteristica, cioè, le letture garantite dopo il commit di una transazione; dietro questo lungo nome, si nasconde un semplice fatto: dopo che una transazione è stata conclusa, si ha la garanzia che tutte le copie dei dati sono sincronizzate. Questo, però, è vero solo in parte, si vedrà ora perché. Si è detto che vi sono almeno tre copie dei dati per ragioni di sicurezza, ma non si è detto che avere più copie agevola anche le prestazioni: infatti, così facendo, si può avere facilmente l'accesso in concorrenza. Un po' come succede nei kernel per i computer, l'aspetto della concorrenza viene gestito dal *load balancer*, un software che dirige le richieste in modo tale da garantire un carico mediamente basso su tutte le macchine. Ci si è un attimo allontanati dalla sincronizzazione dei dati per dire che, dopo una transazione avvenuta con successo, il load balancer farà arrivare le richieste (ad esempio, di lettura) verso le tabelle che sono sincronizzate, e non verso le altre (almeno finché non sono aggiornate). Pertanto, per sintetizzare, una lettura successiva al commit di una scrittura leggerà i contenuti scritti, come ci si può ragionevolmente aspettare che sia.

## 5.3 Dettagli tecnici

Ciascuna tabella può raggiungere dimensioni davvero notevoli (100TB) e ciascuna riga può *pesare* sino a 1MB. Come già accennato, ogni entità di tipo riga può avere un massimo di 255 proprietà, ed un minimo di 3; infatti, ciascuna riga deve almeno avere le seguenti proprietà: Timestamp, PartitionKey e RowKey. All'interno di una data tabella, la proprietà PartitionKey identifica univocamente una partizione, mentre RowKey identifica univocamente una riga; tali proprietà sono gli unici indici sui dati disponibili. Inoltre, la proprietà PartitionKey viene usata per ripartire le partizioni di ciascuna tabella, al fine di bilanciare il carico o scalare meglio: infatti, ciascuna partizione può essere allocata in un nodo differente all'interno della cloud. Infine, presa un'entità, si ha le sue proprietà devono avere un tipo presente in tabella 5.1.

Tra i tipi di dato comunemente usati ma assenti dalla tabella, si hanno le enumerazioni. Secondo lo scrittore di questa relazione, questa mancanza è grave, perché induce gli sviluppatori ad utilizzare interi o booleani per

| Tipo a runtime | Tipo del servizio | Descrizione |
|---|---|---|
| `byte[]` | `Edm.Binary` | Un array di byte avente dimensione massima di 64KB. |
| `bool` | `Edm.Bool` | Un valore booleano. |
| `DateTime` | `Edm.DateTime` | Un valore a 64 bit espresso secondo UTC. |
| `double` | `Edm.Double` | Un valore in virgola mobile a 64 bit. |
| `Guid` | `Edm.Guid` | Un identificatore globale a 128 bit. |
| `Int32, int` | `Edm.Int32` | Un intero a 32 bit. |
| `Int64, long` | `Edm.Int64` | Un intero a 64 bit. |
| `String, string` | `Edm.String` | Una stringa in UTF-16, con dimensione fino a 64KB. |

Tabella 5.1: Tipi di dato disponibili per le entità di una tabella.

rappresentare ciò che sarebbe elegante e naturale rappresentare con delle enumerazioni. Un'altra pecca, è che se si inserisce una proprietà avente un tipo non incluso nella tabella, solo a runtime ci si accorgerà dell'errore.

## 5.4 Costo del servizio

Attualmente, il criterio secondo cui si stabilisce quanto si debba pagare mensilmente è lo stesso usato per i blob e per le code; pertanto, se non lo si è già visto, si veda sezione 4.5 per una descrizione dettagliata.

## 5.5 Transazioni

In maniera simile a quanto si fa con i database relazionali, è possibile eseguire un gruppo di operazioni (inserimenti, aggiornamenti e cancellazioni) all'interno di una singola transazione; tale *raggruppamento* di operazioni viene detto "Entity Group Transaction" (EGT). Affinché sia effettuabile, un EGT deve rispettare i seguenti requisiti:

- Tutte le entità coinvolte nelle operazioni della transazione devono avere lo stesso valore per l'attributo PartitionKey;
- Una data entità può apparire solo una volta nella transazione, ed una sola operazione può essere eseguita su di essa;
- Una transazione può includere al massimo cento entità, e il suo *carico* totale non può superare i 4MB.

La semantica per le EGT è definita nelle specifiche di ADO.NET (per i dettagli, si veda [15]); esse introducono i seguenti per le richieste *batch*, cioè, di gruppo:

- Un *change set* è un gruppo di una o più operazioni di inserimento, aggiornamento e cancellazione;
- Un *batch* è un contenitore di operazioni, al cui interno possono essere sistemati molteplici change set e query.

Le tabelle di Azure supportano soltanto un sottoinsieme delle funzionalità definite dalle specifiche di ADO.NET; in particolare, si ha che:

- Un batch può contenere uno e un solo change set, il quale a sua volta può includere più operazioni di inserimento, aggiornamento e cancellazione. Se un batch contiene più di un change set, solo il primo sarà processato dal servizio, mentre gli altri saranno rigettati e verrà restituito un codice di errore.
- Non è consentito porre una query all'interno di un batch contenente operazioni di inserimento, aggiornamento, cancellazione; essa deve essere l'unica operazione nella batch. Attualmente il servizio non supporta più query all'interno dello stesso batch.
- Le operazioni all'interno di un change set sono processate atomicamente, cioè, o tutte le operazioni vengono eseguite con successo, o tutte falliscono. L'ordine di esecuzione delle operazioni è quello specificato dal change set.

In base a quanto detto finora sulle transazioni relative alle tabelle di Azure, si può affermare che il servizio le implementa soltanto per i gruppi di operazioni agenti su una singola partizione, mentre non implementa alcuno strumento per eseguirle su più partizioni.

## 5.6 Tabella fortemente tipata

Proprio all'inizio del capitolo, si è fatta l'analogia con i software simili a Microsoft Excel per descrivere la logica secondo cui le tabelle di Windows Azure sono state progettate. Brevemente, si era detto che, essendo prive di uno schema, in esse si possono inserire diversi tipi di tuple; ad esempio, in una tabella denominata "frutta", nessuno proibisce di inserire, oltre ai frutti, anche un motore a vapore. Questo fatto può essere utile per alcuni ambiti, dove non è facile trovare uno schema comune o dove un schema non può essere stabilito per via di requisiti molto volatili; tuttavia, in informatica di solito la possibilità di mescolare entità di domini diversi può portare, senza mezzi termini, a disastri spettacolari nel corso della vita del sistema che si sta sviluppando.

Pertanto, in questa parte si propone una semplice classe wrapper (per la piattaforma .NET) che rende le tabelle fortemente tipate. Si deve sottolineare il fatto che questa estensione non le rende esattamente *restrittive* nei confronti delle colonne di una riga come lo sono le tabelle relazionali, in quanto non si è riuscito a risolvere un problema (a seconda dei punti di vista, potrebbe non essere tale). Supponendo di avere una classe C, l'estensione garantisce che solo gli oggetti di classe C o sue derivate possano essere inserite nella tabella; in altre parole, non si è riusciti a porre un vincolo a livello statico che bloccasse l'aggiunta dei sottotipi. Ricapitolando, l'estensione proposta (visibile in versione riassuntiva in listato 5.1) risolve alcune "smerigliature" della libreria .NET e rende le tabelle fortemente tipate, offrendo maggiore garanzia sui tipi delle entità presenti in una tabella.

**Algoritmo 5.1** Classe ”wrapper” che consente di avere una tabella fortemente tipata.

```
1  public class AzureTable<TEntity> : IStorage where TEntity : TableServiceEntity
2  {
3      private static readonly string TableName = (typeof (TEntity)).Name.ToLower();
4
5      private readonly CloudTableClient _tableClient;
6      private readonly TableServiceContext _tableContext;
7
8      private AzureTable(CloudTableClient tableClient) { ... }
9
10     public IQueryable<TEntity> Entities // Da usare per le query.
11     {
12         get { return _tableContext.CreateQuery<TEntity>(TableName); }
13     }
14
15     public string Name { ... }
16     public Uri Uri     { ... }
17
18     public static AzureTable<TEntity> Connect(string tableEndpointUri,
19                                               StorageCredentials cred) { ... }
20     public static void Create(string tableEndpointUri,
21                               StorageCredentials cred) { ... }
22
23     // Chiamano i (quasi) omonimi metodi su _tableContext.
24     public void AddEntity(TEntity entity)    { ... }
25     public void DeleteEntity(TEntity entity) { ... }
26     public void UpdateEntity(TEntity entity) { ... }
27     public void SaveChanges(SaveChangesOptions options) { ... }
28
29     public void Clear()  { ... } // Pulisce il contenuto della tabella.
30     public void Delete() { ... } // Cancella la tabella dalla cloud.
31     public bool Exists() { ... } // Controlla se la tabella esiste o meno.
32 }
```

## 5.7 Esempi d'uso

In questa sezione, verranno esposti alcuni semplici esempi, in modo da offrire al lettore una base dalla quale avviare l'apprendimento delle tabelle di Windows Azure.

### 5.7.1 Classi di "contorno"

Nel corso degli esempi presentati in seguito, si farà uso di alcune entità, presentate in questa parte; in particolare, si coglierà l'occasione per descrivere come costruire correttamente un'entità, facendo attenzione ad alcune particolarità richieste dal servizio. Per cominciare, si veda la classe `Device` definita in listato 5.2, la quale rappresenta una componente di un personal computer; si vede immediatamente che essa eredita dalla classe astratta (fornita dalla libreria .NET) `TableServiceEntity`, in quanto questa implementa già le proprietà richieste dal servizio come RowKey e PartitionKey. In particolare, si noti come il costruttore con i parametri chiami quello base, passandogli rispettivamente il valore per PartitionKey e RowKey: la classe `TableServiceEntity` offre anche un costruttore senza parametri, ma esso andrà usato per altri scopi. Per evitare di lasciare dubbi, è necessario ripetere che è obbligatorio chiamare il costruttore di `TableServiceEntity`, al fine della corretta impostazione dell'entità.

Tali scopi sono la serializzazione, necessaria per poter trascrivere l'entità nella memoria persistente; pertanto, come si vede nella classe `Device`, è necessario definire un costruttore pubblico e senza parametri, che verrà usato solamente per la serializzazione (per questa ragione, esso è stato opzionalmente marcato come "Obsolete", al fine di non farlo usare nel codice "normale").

Nella classe `Device`, l'ultima cosa di interesse sono le proprietà Name, Role e Socket, in quanto esse rappresentano le colonne, insieme a Timestamp, RowKey e PartitionKey che l'entità di tipo riga corrispondente a `Device` avrà. I tipi degli attributi sono tutti string, in accordo con tabella 5.1.

Ora, in listato 5.3 sono elencate alcune classi che ereditano da `Device`, le quali per verranno utilizzate per mostrare il problema dell'ereditarietà

**Algoritmo 5.2** Classe che rappresenta una componente di un personal computer.

```
1 public class Device : TableServiceEntity
2 {
3     // Convenzione per il nome delle tabelle usata negli esempi proposti.
4     private static readonly string TableName = (typeof (Device)).Name.ToLower();
5
6     // La chiamata al costruttore base imposta PartitionKey e RowKey.
7     public Device(string deviceId, string deviceName = "Device",
8                   string deviceRole = "Role", string deviceSocket = "Socket")
9         : base(TableName, deviceId) { ... }
10
11    // Richiesto per la serializzazione, non usare altrimenti.
12    [Obsolete] public Device() { /* Vuoto */ }
13
14    // Nuove "colonne" aggiunte all'entita' base.
15    public string Name   { get; set; }
16    public string Role   { get; set; }
17    public string Socket { get; set; }
18 }
```

**Algoritmo 5.3** Classi che ereditano da Device.

```
1 public class Screen : Device
2 {
3     public Screen(string deviceId, string deviceName,
4                   int widthInPixel, int heightInPixel)
5         : base(deviceId, deviceName, "Screen", "DVI") { ... }
6
7     [Obsolete] public Screen() { /* Vuoto */ }
8
9     public int WidthInPixel  { get; set; }
10    public int HeightInPixel { get; set; }
11 }
12
13 public class UsbDrive : Device
14 {
15    public UsbDrive(string deviceId, string deviceName, int capacityInMb)
16        : base(deviceId, deviceName, "Usb drive", "Usb port") { ... }
17
18    [Obsolete] public UsbDrive() { /* Vuoto */ }
19
20    public int CapacityInMb { get; set; }
21 }
```

presentato in sezione 5.6; si faccia attenzione al fatto che anch'esse presentano un costruttore vuoto, utilizzato solamente per la serializzazione.

Infine, in listato 5.4 si è definita un'altra entità, `Fruit`, che verrà impiegata per mettere in luce la problematica della "promiscuità" delle tabelle fornite da Windows Azure.

Seguono ora alcuni esempi d'uso; per ciascun, se opportuno, si mostrerà come si potrebbe gestire la stessa situazione con la tabella definita in sezione 5.6.

**Algoritmo 5.4** Classe che rappresenta un frutto.

```
1 public class Fruit : TableServiceEntity
2 {
3     private static readonly string TableName = (typeof(Fruit)).Name.ToLower();
4
5     public Fruit(string fruitId, string fruitName)
6         : base(TableName, fruitId) { ... }
7
8     [Obsolete] public Fruit() { /* Vuoto */ }
9
10    public string Name { get; set; }
11 }
```

**Algoritmo 5.5** Creazione di una tabella.

```
1 var connectionString = Settings.Default.DataConnectionString;
2 var storageAccount = CloudStorageAccount.Parse(connectionString);
3
4 var tableEndpointUri = storageAccount.TableEndpoint.ToString();
5 var credentials = storageAccount.Credentials;
6 var tableClient = new CloudTableClient(tableEndpointUri, credentials);
7
8 tableClient.CreateTableIfNotExist(tableName);
```

### 5.7.2 Creazione di una tabella

Il codice necessario per creare una tabella, presentato in listato 5.5 è molto semplice e breve, ed assomiglia a quello necessario per creare un blob container od una coda. Le uniche informazioni richieste sono le credenziali d'accesso e l'URI del servizio delle tabelle (entrambi ricavabili da un oggetto di tipo CloudStorageAccount).

In modo analogo, e con le stesse informazioni, si crea la tabella usando la classe wrapper esposta in questo capitolo; in listato 5.6 se ne vede una dimostrazione.

**Algoritmo 5.6** Creazione di una tabella fortemente tipata.

```
1 var connectionString = Settings.Default.DataConnectionString;
2 var storageAccount = CloudStorageAccount.Parse(connectionString);
3
4 var tableEndpointUri = storageAccount.TableEndpoint.ToString();
5 var credentials = storageAccount.Credentials;
6 AzureTable<Device>.Create(tableEndpointUri, credentials);
```

### 5.7.3 Uso di una tabella

Chi ha già provato ADO.NET, troverà il codice in listato 5.7 vagamente familiare, in quanto il modello di programmazione di programmazione per le tabelle assomiglia molto all'Entity Data Model. Infatti, per eseguire un qualunque tipo di operazione, è necessario avere un oggetto di tipo TableServiceContext, ricavabile da un oggetto di classe CloudTableClient; un lettore molto acuto può avere già notato un problema della libreria nelle ultime parole della frase precedente.

Infatti, come si spiega il nome della tabella presente tra i parametri delle chiamate ad AddObject, nelle righe sette ed otto? Il contesto preso dal client del servizio non è collegato ad alcuna tabella; pertanto, è necessario specificare ogni volta quale sia la tabella obiettivo dell'aggiunta. Inoltre, si noti come il nome dei metodi di manipolazione contenga un Object nel nome, e non solo lì: la segnatura dei metodi prevede un Object, cosa che può creare i più svariati problemi a tempo di esecuzione.

I problemi sopra esposti non sussistono nella versione che usa la tabella fortemente tipata; in listato 5.8, essa è stata creata per ospitare entità di tipo Device; pertanto, nei metodi di manipolazione il parametro non avrà tipo Object ma Device, e la proprietà Entities restituirà un oggetto di tipo IQueryable<Device>.

### 5.7.4 Gestione del tipaggio forte

Come evidenziato nella parte precedente, ed in generale nel corso di tutto il capitolo, le tabelle fornite dal servizio consento l'inserimento di entità di vari

**Algoritmo 5.7** Uso di una tabella.

```
1  const string tableName = "devices";
2  var devices = SetupTable(tableName); // Torna un oggetto TableServiceContext.
3  var pinoMouse = new Device("m1", "PinoMouse");
4  var ginoCam = new Device("c7", "GinoCam");
5
6  // Notare che si richiede il nome della tabella ad ogni aggiunta.
7  devices.AddObject(tableName, pinoMouse);
8  devices.AddObject(tableName, ginoCam);
9  devices.SaveChanges();
10
11 pinoMouse.Name = "ExtremePinoMouse";
12 devices.UpdateObject(pinoMouse);
13 devices.SaveChanges();
14
15 var query = devices.CreateQuery<Device>(tableName).Where(d => d.RowKey == "m1");
16 Debug.Assert(query.First().Name == pinoMouse.Name);
```

**Algoritmo 5.8** Uso di una tabella fortemente tipata.

```
1  var pinoMouse = new Device("m1", "PinoMouse");
2  var ginoCam = new Device("c7", "GinoCam");
3
4  // "devices" ha tipo AzureTable<Device>
5  devices.AddEntity(pinoMouse);
6  devices.AddEntity(ginoCam);
7  devices.SaveChanges();
8
9  pinoMouse.Name = "ExtremePinoMouse";
10 devices.UpdateEntity(pinoMouse);
11 devices.SaveChanges();
12
13 var query = devices.Entities.Where(d => d.RowKey == "m1");
14 Debug.Assert(query.First().Name == pinoMouse.Name);
```

**Algoritmo 5.9** Gestione del tipaggio forte con una tabella "standard".

```
1  const string tableName = "devices";
2  var devices = SetupTable(tableName);
3  var pinoMouse = new Device("m1", "PinoMouse");
4  var ginoPeach = new Fruit("p6", "GinoPeach");
5
6  // Si mescolano componenti di pc e frutta; questo puo' essere utile
7  // per alcuni casi d'uso, ma una bomba ad orologeria per altri.
8  devices.AddObject(tableName, pinoMouse);
9  devices.AddObject(tableName, ginoPeach);
10 devices.SaveChanges();
11
12 var query = devices.CreateQuery<Fruit>(tableName).Where(d => d.RowKey == "p6");
13 Debug.Assert(query.First().Name == ginoPeach.Name);
```

tipi; questo, tuttavia, potrebbe essere qualcosa di non voluto, tutto dipende dall'ambito in cui si sta lavorando. Ad ogni modo, in listato 5.9 si fa vedere come, con la libreria standard, non ci siano blocchi verso questo tipo di problema, mentre in listato 5.10 si mette alla prova il tipaggio forte della tabella wrapper descritta in questo capitolo.

### 5.7.5 Gestione dell'ereditarietà

In precedenza, si è detto che la tabella fortemente tipata non è in grado di impedire l'aggiunta di un'entità discendente dall'entità per cui la tabella è stata creata; in altre parole, non si è riusciti ad ottenere lo stesso livello di forzatura dello schema offerto dalle tabelle relazionali. Questo fatto, messo in luce in listato 5.11, può o no rappresentare un problema; in ogni caso, finora non si è trovata soluzione ad esso. Ovviamente, questa problematica è presente anche nella libreria .NET, in quanto essa consente la presenza di entità completamente distinte nella stessa tabella.

**Algoritmo 5.10** Gestione del tipaggio forte con una tabella fortemente tipata.

```
1 var pinoMouse = new Device("m1", "PinoMouse");
2 var ginoPeach = new Fruit("p6", "GinoPeach");
3
4 devices.AddEntity(pinoMouse);
5 // devices.AddEntity(ginoPeach); -> Bloccato dal compilatore stesso.
6 devices.SaveChanges();
7
8 var query = devices.Entities.Where(d => d.RowKey == "m1");
9 Debug.Assert(query.First().Name == pinoMouse.Name);
```

**Algoritmo 5.11** Gestione dell'ereditarietà con una tabella fortemente tipata.

```
1 var pinoMouse = new Device("m1", "PinoMouse");
2 var ginoDrive = new UsbDrive("u5", "GinoDrive", 512);
3 var bobScreen = new Screen("s3", "BobScreen", 800, 600);
4
5 // Le classi "figlie" possono essere aggiunte alla tabella.
6 devices.AddEntity(pinoMouse);
7 devices.AddEntity(ginoDrive);
8 devices.AddEntity(bobScreen);
9 devices.SaveChanges();
10
11 var query = devices.Entities.Where(d => d.Name == "GinoDrive");
12 var entity = (UsbDrive)query.First();
13 Debug.Assert(entity.CapacityInMb == ginoDrive.CapacityInMb);
```

# Capitolo 6

# Azure Storage - Code

Le code forniscono un meccanismo per effettuare lo scambio di messaggi tra le applicazioni operanti su Windows Azure (e non solo), secondo i principi di persistenza e durabilità; i messaggi sono persistenti perché vengono appositamente memorizzati, come per tutti gli altri servizi di storage, almeno tre volte, e allo stesso tempo essi sono durabili perché riescono ad essere disponibili nonostante eventuali errori all'interno del sistema. Si vedranno ora alcuni usi *teorici* delle code fornite da Windows Azure, seguiti da brevi dettagli tecnici sul servizio. Infine, come nei precedenti capitoli, si mostreranno alcuni esempi d'utilizzo delle code attraverso la libreria .NET.

## 6.1 Scenari d'uso delle code

Le code, per loro natura, consentono di realizzare un sistema di messaggistica tra le applicazioni o, ancora meglio, tra parti di esse; cioè, tramite l'uso delle code, è possibile rendere le componenti di un sistema lascamente accoppiate, ma al tempo stesso altamente integrate. Infatti, supponendo di avere uno scenario dove un'applicazione si occupa di ricevere richieste di calcolo mentre un'altra si occupa del calcolo vero e proprio, la coda potrebbe essere usata dalla prima applicazione per mandare alla seconda una descrizione del lavoro da eseguire; in particolare, l'applicazione che invia le richieste (produttore) non sa nulla (e nulla deve sapere) su come il *calcolatore* (e consumatore)

gestisce le richieste, le basta sapere che le richieste vengono davvero gestite e che riceverà una risposta in un formato concordato. Si noti che le due componenti, pur essendo lascamente accoppiate (si può cambiare l'applicazione che esegue i calcoli senza che l'altra se ne accorga), sono strettamente collegate, perché nessuna delle due può girare indipendentemente dall'altra.

Come si è accennato all'inizio del capitolo, non solo le applicazioni che "girano" su Windows Azure possono accedere alle code; infatti, esse possono essere utilizzate anche da applicazioni in locale, nonché da applicazioni di tipo *mobile*. Pertanto, da questo punto di vista, le code sono una sorta di *interfaccia* tra diversi tipi di applicazioni e di piattaforme. Come esempio di questo concetto, si potrebbero pensare al fatto che i dispositivi mobile non hanno una grande capacità computazionale, e potrebbero incontrare difficoltà a caricare pesanti pagine web: usando le code, essi potrebbero comunicare il loro desiderio di mostrare una pagina; la richiesta verrebbe poi rapidamente processata da un cluster, il quale, sempre usando come mezzo la coda, potrebbe inviare la pagina compilata (o un riferimento ad esso) al dispositivo mobile[1].

## 6.2 Dettagli tecnici

Ciascun messaggio inserito nella coda può avere dimensione massima di 8KB e viene memorizzato/recuperato in genere secondo una politica FIFO; si è detto "in genere" perché l'attuazione di tale politica non è garantita dal servizio. In particolare, la gestione dei messaggi della coda è un processo articolato in due fasi: il prelievo del messaggio e la cancellazione dalla coda del messaggio stesso a gestione compiuta. Questo metodo consente una consegna garantita dei messaggi, in quanto il messaggio rimane nella coda finché non è stato completamente gestito. Prelevare un messaggio lo rende invisibile, in modo tale che più applicazioni non possano gestire contemporaneamente lo stesso messaggio; il messaggio rimane invisibile finché non viene cancellato

[1] Un sistema simile è stato realizzato per il dispositivo Kindle Fire, di Amazon; ovviamente, in quel sistema viene usata la cloud di Amazon stessa come *backend* per i calcoli.

**Algoritmo 6.1** Creazione di una coda.

```
1 var connectionString = Settings.Default.DataConnectionString;
2 var storageAccount = CloudStorageAccount.Parse(connectionString);
3 var queueEndpointUri = storageAccount.QueueEndpoint.ToString();
4 var credentials = storageAccount.Credentials;
5 var queue = new CloudQueue(queueEndpointUri + "/" + queueName, credentials);
6 queue.CreateIfNotExist();
```

oppure è trascorsa una certa quantità di tempo. Inoltre, se si effettua solo un'operazione di "peek", cioè, si *sbircia* soltanto il contenuto di un messaggio, non viene effettuata la marcatura a invisibile di tale messaggio.

## 6.3 Costo del servizio

Attualmente, il criterio secondo cui si stabilisce quanto si debba pagare mensilmente è lo stesso usato per i blob e per le tabelle; pertanto, se non lo si è già visto, si veda sezione 4.5 per una descrizione dettagliata.

## 6.4 Esempi d'uso

Si propongono ora alcuni esempi molto semplici su come usare le code; in particolare, si vedrà sinteticamente un sistema produttore/consumatore per l'ordinamento di un insieme di numeri interi e si mostrerà una classe wrapper per facilitare (e potenziare) l'uso delle code.

### 6.4.1 Creazione di una coda

Come si può notare in listato 6.1, la creazione di una coda è un processo del tutto simile alla creazione di un blob o una tabella. In particolare, al fine del completamento dell'operazione, è necessario avere l'URI della coda (cioè, una stringa del tipo `$URI_Servizio_Code/$Nome_Coda`) e le credenziali di accesso (ricavabili dall'oggetto di tipo CloudStorageAccount).

**Algoritmo 6.2** Uso di una coda.

```
1 // "Integers" e' una lista di interi che deve essere ordinata;
2 // "inputs" e' una coda in cui si inseriscono gli insiemi da ordinare,
3 // mentre "outputs" e' sempre una coda dove si inseriscono gli insiemi ordinati.
4 // Inserimento dell'insieme da ordinare:
5 inputs.AddMessage(new CloudQueueMessage(ListToString(Integers)));
6 // Estrazione dell'insieme e suo ordinamento, con conseguente
7 // inserimento nella coda degli output:
8 var integersToSort = StringToList(inputs.GetMessage().AsString);
9 integersToSort.Sort();
10 outputs.AddMessage(new CloudQueueMessage(ListToString(integersToSort)));
11 // Controllo che l'insieme sia stato davvero usato:
12 var sortedOutput = StringToList(outputs.GetMessage().AsString);
13 Debug.Assert(sortedOutput.Count == integersToSort.Count);
14 for (var i = 0; i < integersToSort.Count; ++i)
15     Debug.Assert(sortedOutput[i] == integersToSort[i]);
```

### 6.4.2 Uso di una coda

Come accennato prima, per esemplificare l'uso della coda si realizzerà un semplicissimo sistema produttore/consumatore, il cui scopo è ordinare insiemi di interi. Come mostrato in listato 6.2, il produttore inserisce l'insieme da ordinare (sotto forma di stringa) nella coda per gli input; da lì, il consumatore estrae la stringa contenente i numeri e, dopo averla trasformata in un insieme, la ordina e la riporta sotto forma di stringa, pronta per essere aggiunta alla coda degli output.

Già in questo esempio davvero sintetico un lettore acuto potrebbe aver notato alcune *problematiche* relative all'uso delle code. Infatti, considerato che i messaggi possono essere soltanto stringhe, occorre servirsi di sistemi particolari per propagare oggetti interi sulla coda; i sistemi più comuni, come in tutto il resto del Web, sono XML e JSON, mentre, nei casi semplici come l'esempio qui esposto, è possibile ricorrere a sistemi di serializzazione "fatti in casa". Inoltre, la libreria per accedere ai servizi di Azure non offre metodi *bloccanti* per eseguire l'estrazione di un messaggio dalla coda; se non è presente alcun messaggio, una chiamata a GetMessage ritorna immediatamente.

### 6.4.3 Coda ”personalizzata”

Risolvere le problematiche esposte nella sezione precedente non è complicato, o meglio, è facile rendere meno *dolorosa* la loro presenza con del codice wrapper. In listato 6.3 si espongono i metodi di una classe wrapper che aggira tali problemi. Si noti che, per il problema della lettura bloccante/non bloccante, si è introdotto un parametro indicante il tipo di lettura al metodo che si occupa di estrarre il messaggio dalla coda; all'interno, tale metodo effettua un polling della coda ad intervalli costanti in quanto, nonostante la cosa non sia per nulla elegante, le API del servizio non consentono altre vie. Infine, si noti come si richieda che i messaggi da inserire nella coda debbano rispettare una data interfaccia; essa impone la presenza dei metodi FromString e ToString, per consentire una elegante trasformazione della stringa ad oggetto e viceversa.

### 6.4.4 Uso della coda personalizzata

Nel codice presente in listato 6.4 poco cambia da quello mostrato in listato 6.2, se non l'uso della coda personalizzata al posto di quella di default. Questo fatto, unito all'aggiunta della classe `IntSortMessage` e dei suoi metodi, rende il codice più snello e leggibile, pur mantenendo lo stesso grado di funzionalità.

**Algoritmo 6.3** Classe che implementa una coda ”personalizzata”.

```
1 public class AzureQueue<TMsg> : IStorage where TMsg : IMessage, new()
2 {
3     private readonly CloudQueue _queue;
4
5     private AzureQueue(CloudQueue queue) { ... }
6
7     public string Name { ... }
8     public Uri Uri     { ... }
9
10    public static AzureQueue<TMsg> Connect(string queueName, string queueEndpointUri,
11                                           StorageCredentials credentials) { ... }
12    public static void Create(string queueName, string queueEndpointUri,
13                              StorageCredentials credentials) { ... }
14
15    public TMsg DequeueMessage(ReadMode readMode = ReadMode.Blocking) { ... }
16    public void EnqueueMessage(TMsg msg) { ... }
17    public IList<TMsg> PeekMessages()    { ... }
18
19    public void Clear()  { ... } // Pulisce il contenuto della coda.
20    public void Delete() { ... } // Cancella la coda dalla cloud.
21    public bool Exists() { ... } // Controlla se la coda esiste o meno.
22 }
23
24 public enum ReadMode { Blocking, NotBlocking }
```

**Algoritmo 6.4** Uso di una coda personalizzata.

```
1  // Le variabili hanno lo stesso ruolo che avevano nell'esempio analogo.
2  // Inserimento dell'insieme da ordinare (notare come, ereditando da
3  // CloudQueueMessage, si possano effettuare chiamate piu' eleganti):
4  inputs.EnqueueMessage(IntSortMessage.FromList(Integers));
5  // Estrazione dell'insieme e suo ordinamento, con conseguente
6  // inserimento nella coda degli output:
7  var integersToSort = inputs.DequeueMessage().Integers;
8  integersToSort.Sort();
9  outputs.EnqueueMessage(IntSortMessage.FromList(integersToSort));
10 // Controllo che l'insieme sia stato davvero usato:
11 var sortedMsg = outputs.DequeueMessage();
12 Debug.Assert(sortedMsg.Integers.Count == integersToSort.Count);
13 for (var i = 0; i < integersToSort.Count; ++i)
14     Debug.Assert(sortedMsg.Integers[i] == integersToSort[i]);
```

# Capitolo 7

# Altri servizi della piattaforma

In questo capitolo, si presentano altri due servizi della piattaforma, entrambi a modo loro legati al concetto di storage. Il primo, SQL Azure, ha già nel proprio nome un evidente legame con la memorizzazione, in quanto offre un database SQL Server residente nel cloud; il secondo, Windows Azure Marketplace, invece, ha un legame più fine con tale idea: infatti, una parte di esso, come si vedrà, è dedicata alla compravendita di dati.

## 7.1 SQL Azure

SQL Azure rappresenta l'offerta Microsoft di un database relazionale ospitato nella cloud; in quanto tale, esso si contrappone (o meglio, complementa) le funzionalità delle tabelle offerte da Windows Azure (si veda capitolo 5). A differenza degli altri servizi di memorizzazione offerti dalla piattaforma, SQL Azure fornisce, oltre alla capacità di memorizzare dei dati, anche la capacità di processarli, sempre lato server. Ciò consente di processare i dati in modo complesso, senza dover caricare l'intero insieme dei dati all'interno delle applicazioni.

Un database di tale tipo è, dal punto di vista applicativo, non molto diverso da un database SQL Server; rispetto ad esso, si hanno alcune funzionalità mancanti (come SQL Common Language Runtime), a fronte di una maggiore sicurezza e ridondanza dei dati; infatti, tutti i dati presenti su un database

SQL Azure sono replicati almeno tre volte, e i meccanismi di scrittura sono studiati in modo tale da essere consistenti: se una operazione di scrittura conclude il proprio compito, si ha la garanzia che la scrittura sia stata fatta per davvero. Vista la "parentela" con SQL Server, tutte le applicazioni correnti che usano tecnologie come ADO.NET o ODBC per comunicare con SQL Server richiedono pochi cambiamenti al codice al fine di poter comunicare con SQL Azure. In generale, l'unico cambiamento che sarà richiesto a tali applicazioni consisterà nella modifica della stringa di connessione, in modo che punti al nuovo database SQL Azure.

SQL Azure ricade nella categoria PaaS (Platform as a Service); pertanto, quasi per definizione, non è possibile accedere all'hardware sottostante. Ciò impedisce di eseguire quei compiti di amministrazione che richiedono l'accesso all'hardware, come stabilire la locazione del database. L'amministrazione *fisica* della piattaforma viene gestita in automatico, mentre l'amministrazione *logica* (compiti quali creazione di utenti e ruoli) è ancora a carico dell'utente.

Infine, in figura 7.1 si possono osservare due cose: la prima, è che i dati memorizzati su un database SQL Azure possono anche essere acceduti utilizzando il protocollo OData; la seconda, è che SQL Azure fornisce due servizi aggiuntivi, SQL Azure Reporting e Data Sync, di cui si parlerà ora.

### 7.1.1 Reporting

Lo scopo di SQL Azure Reporting è generare report sui dati salvati su database SQL Azure; il tutto è basato su SSRS (SQL Server Reporting Services), strumento che può essere familiare agli utenti esperti di SQL Server. In particolare, considerato che un database SQL Azure risulta alle applicazioni come un database SQL Server, è possibile usare direttamente SSRS per generare i report; tuttavia, SQL Azure Reporting, oltre ad essere eseguito nella cloud, presenta caratteristiche utili per due tipi di scenari:

- I report creati usando SQL Azure Reporting possono essere pubblicati in un portale specifico, dando la possibilità agli utenti di usufruirne direttamente lì, o rendendoli disponibili attraverso un URL.

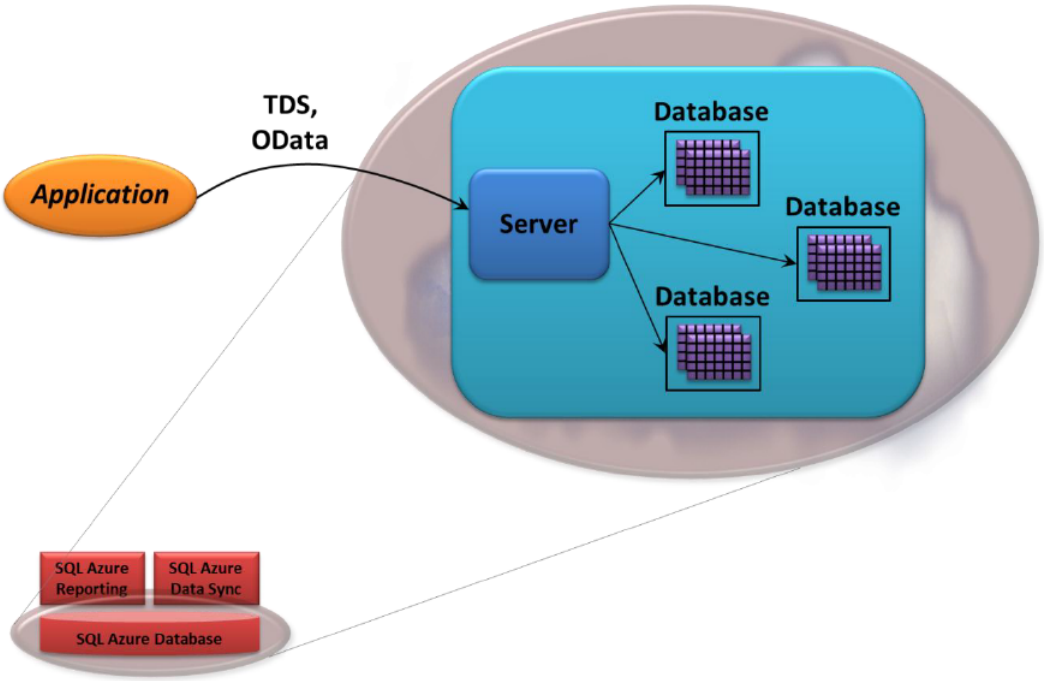



Figura 7.1: Visione dettagliata di SQL Azure.

- Un ISV (Independent Software Vendor) può includere in un'applicazione i report pubblicati sul portale di SQL Azure Reporting; va notato il fatto importante che anche le applicazioni per Windows Azure possono includerli. Includere i report all'interno di un'applicazione rappresenta una comodità per gli utenti, i quali possono vedere i report senza abbandonare l'applicazione stessa; per ottenere ciò, si possono utilizzare i controlli ReportViewer offerti da Visual Studio, in modo analogo a quanto si fa per i report generati in locale.

Come altre parti di SQL Azure, SQL Azure Reporting non fornisce tutte le funzionalità di SSRS. Attualmente, non vi è il supporto per la programmazione dei report e la possibilità di sottoscriversi a essi; pertanto, ad esempio, non è possibile generare e consegnare un report a intervalli regolari.

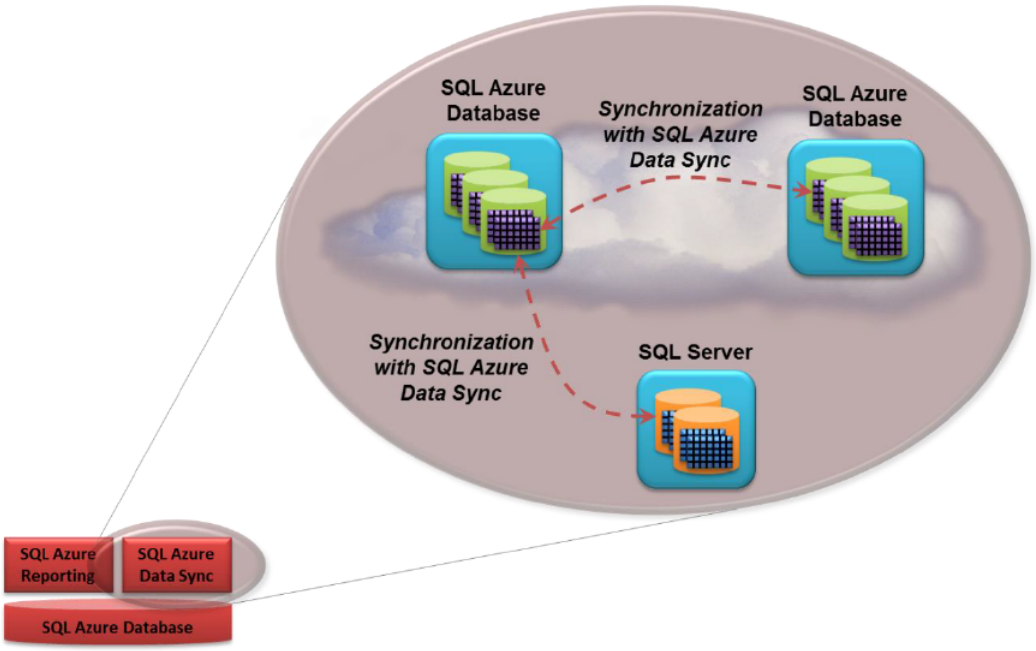



Figura 7.2: Descrizione schematica del funzionamento di SQL Azure Data Sync.

### 7.1.2 Data Sync

Nonostante i dati salvati su un database SQL Azure possano essere acceduti da una qualunque applicazione provvista di una connessione ad Internet, vi sono tuttavia dei casi in cui può avere senso mantenere una copia dei dati in locale o in centri dati di terze parti. Ad esempio, un'organizzazione potrebbe avere la necessità di mantenere una copia locale delle stesse informazioni presenti su un proprio database SQL Azure, per i più svariati motivi: prestazioni, ridondanza in caso di fallimento della rete, regolamentazioni locali.

In casi simili a quelli appena esposti, potrebbe essere la possibilità di sincronizzare i dati memorizzati su un database SQL Azure in un altro dispositivo di memorizzazione; pertanto, vista la concreta necessità, Microsoft offre lo strumento di SQL Azure Data Sync, il quale consente di eseguire tale compito, senza scrivere alcuna riga di codice, ma solo compilando alcuni file di configurazione.

Attualmente, il servizio offre due modalità differenti di sincronizzazione:

- Sincronizzazione dei dati tra un database SQL Azure e un database SQL Server locale. Come si è detto prima, le ragioni per una scelta simile sono molteplici; ad esempio, alcune leggi potrebbero richiedere che almeno una copia dei dati sia sempre entro i confini del paese, oppure, si potrebbe volere una copia locale per gestire eventuali errori amministrativi come un'accidentale cancellazione di alcune tabelle.

- Sincronizzazione dei dati tra più database SQL Azure collocati in differenti centri dati. Questa opzione può tornare utile per le organizzazioni che supportano un'applicazione usata a livello globale; infatti, al fine di mantenere alto il livello prestazionale per gli utenti sparsi per il mondo, avrebbe senso eseguire tale applicazione in tre centri dati distinti (per esempio, in Nord America, in Europa ed in Asia). Se quell'applicazione usasse SQL Azure, potrebbe appoggiarsi ad SQL Azure Data Sync per tenere le informazioni sincronizzate nei tre centri dati, in modo che ciascuna applicazione possa accedere ad un database (aggiornato) nello stesso centro dati in cui opera, andando senza dubbio a vantaggio delle prestazioni.

| Capacità | Costo mensile |
|---|---|
| 1GB | 9.99$ |
| 5GB | 49.95$ |

Tabella 7.1: Costi dei database SQL Azure "Web Edition".

| Capacità | Costo mensile |
|---|---|
| 10GB | 99.99$ |
| 20GB | 199.98$ |
| 30GB | 299.97$ |
| 40GB | 399.96$ |
| 50GB | 499.95$ |

Tabella 7.2: Costi dei database SQL Azure "Business Edition".

SQL Azure Data Sync usa un modello particolare per decidere come propagare le modifiche. Tutti i cambiamenti sono prima copiati nel database SQL Azure *sorgente*, poi essi vengono copiati nei database *obiettivo*; i database obiettivo possono essere altri database SQL Azure o database SQL Server locali. In entrambi i casi, lo strumento offre la possibilità di sincronizzare un intero database o solo alcune tabelle, con la garanzia che i cambiamenti fatti ad ogni copia siano propagati alle altre.

Infine, con SQL Azure Data Sync è possibile sia avviare manualmente la sincronizzazione, sia programmarla in base alle specifiche esigenze.

### 7.1.3 Costo del servizio

Come indicato su [17], attualmente si hanno due tipi di offerte: una versione *Web*, per siti Web e piccole applicazioni, e una versione *Business*, per grandi applicazioni SaaS; in tabella 7.1 sono indicati i costi dei database versione Web, mentre in tabella 7.2 sono indicati quelli dell'edizione Business. Osservando i prezzi, si noterà che, in generale, il rapporto costo mensile/capacità equivale a 9.99$ al mese per ogni gigabyte.

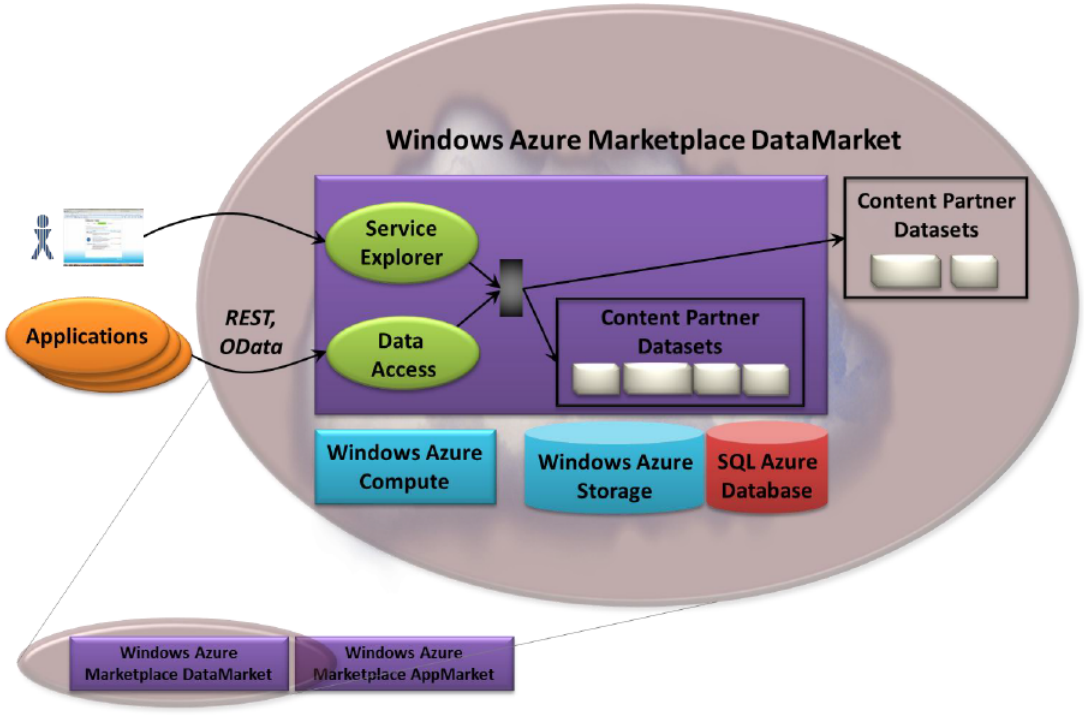



Figura 7.3: Visione dettagliata di Windows Azure Marketplace.

## 7.2 Windows Azure Marketplace

Windows Azure Marketplace è stato progettato per soddisfare due differenti necessità: creare un *luogo* centralizzato dove poter recuperare dei *dati* (in particolare, "insiemi di dati", come le informazioni raccolte da un censimento) e dove poter trovare (cioè, comprare e vendere) le applicazioni ospitate su Windows Azure. Avendo due problemi da risolvere, il Marketplace è suddiviso in due parti, come mostrato in figura 7.3: il DataMarket, dedicato alla compravendita dei dati, e l'AppMarket, avente come scopo la compravendita delle applicazioni.

### 7.2.1 DataMarket

Il DataMarket rappresenta un singolo luogo dove possono essere trovati, comprati ed acceduti una grande quantità di insiemi di dati commerciali. Sia le

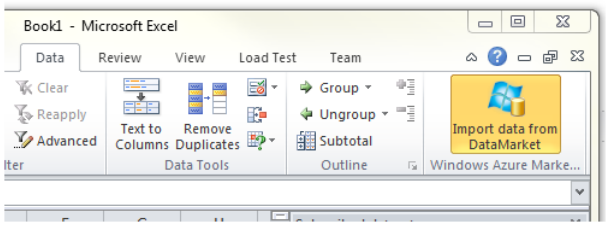



Figura 7.4: Plugin di Microsoft Excel per accedere al DataMarket.

*persone* sia le *applicazioni* possono accedere alle informazioni offerte dal DataMarket, in svariati modi. Ad esempio, le persone possono visualizzare gli insiemi di dati in vendita attraverso un qualunque browser, utilizzando Service Explorer, un'applicazione basata su Windows Azure, per costruire query e osservarne i risultati. Una volta che un insieme di dati è stato acquistato, le applicazioni potranno accedere ad esso tramite richieste RESTful e il protocollo OData; più semplicemente, si potranno usare plugin per software esistenti (come Microsoft Excel, si veda figura ) per manipolare e visualizzare tali dati.

Inoltre, cosa molto importante, i dati resi disponibili sul DataMarket non devono necessariamente essere conservati su Windows Azure Storage o su uno o più database SQL Azure, ma possono anche essere conservati esternamente, come in centri dati gestiti dal fornitore di tali dati.

Ulteriori informazioni su come accedere al DataMarket da vari tipi di software (browser, Visual Studio, PowerPivot) sono presenti su [16].

### Vantaggi per gli sviluppatori

Coerentemente con gli altri servizi di storage offerti dalla piattaforma, e da come si può osservare in figura 7.3, il Marketplace (in particolare, la parte relativa ai dati) offre consistenti interfacce basate su REST e sul protocollo OData per accedere a tutti gli insiemi di dati; ciò, oltre a rappresentare un approccio "familiare", consente lo sviluppo su differenti piattaforme.

Come si sarebbe potuto facilmente immaginare, gli utenti di Visual Studio possono usufruire di facilitazioni aggiuntive, come la possibilità di aggiungere l'insieme di dati come *Service Reference* e ottenere di conseguenza un *object*

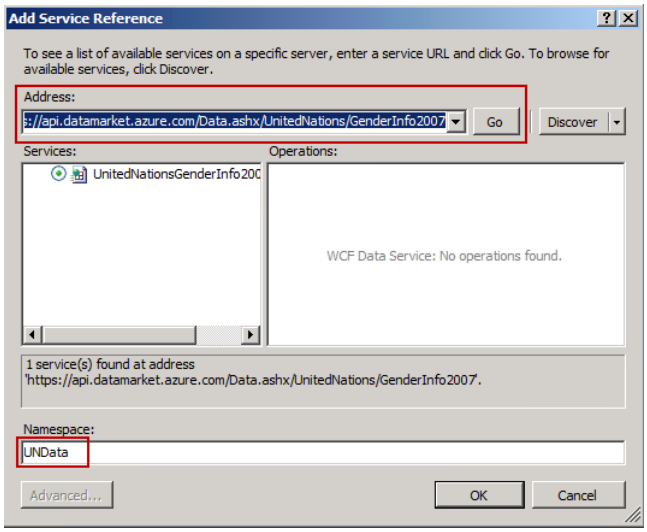



Figura 7.5: Aggiunta di un insieme di dati come Service Reference.

*model*, il quale elimina la necessità di scrivere "a mano" codice per creare i dovuti livelli di astrazione (come mostrato in figura 7.5).

### Vantaggi per le startup

Per una piccola società, uno dei problemi primari, insieme all'impostazione ed alla gestione di un'infrastruttura efficiente e scalabile, è ottenere rapidamente visibilità sul mercato. Pertanto, una startup il cui scopo è produrre insiemi di dati troverà il DataMarket molto utile, perché offre sia l'infrastruttura, sia la visibilità richiesta; infatti, nel momento in cui una società pone i propri dati sul mercato, può raggiungere tutta la gamma di clienti Microsoft.

## 7.2.2 AppMarket

Come si può vedere in figura 7.6, l'AppMarket consente alle applicazioni operanti su Windows Azure di essere vendute e comprate; contrariamente al DataMarket, dove non si pongono vincoli sul luogo dove i dati vengono conservati, l'AppMarket impone che l'applicazione si conservata ed eseguita su Windows Azure. Anche questo tipo di mercato, per gli stessi motivi del DataMarket, può essere molto vantaggioso per le piccole e nascenti società,

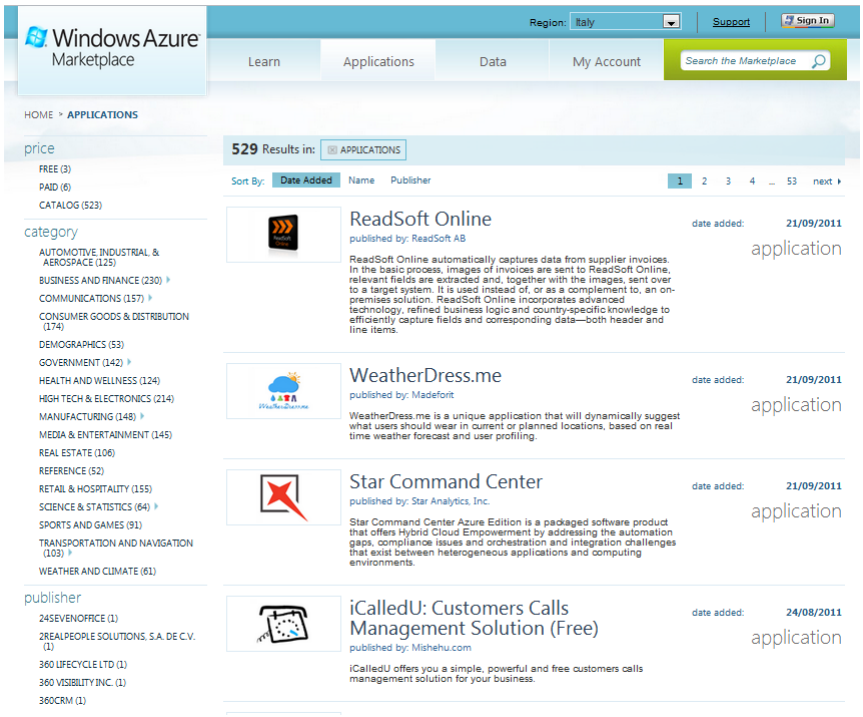



Figura 7.6: Esempio di come appare e cosa offre l'AppMarket.

offrendo una completa infrastruttura per la compravendita (memorizzazione, unita alla gestione dei pagamenti) e un buon grado di visibilità.

Sfortunatamente, non si sono trovate molte altre informazioni; oltretutto, *provare* il funzionamento servizio non è stato possibile sia per questioni pratiche, sia per questioni tecniche.

# Capitolo 8

# Servizi concorrenti

La piattaforma Windows Azure non è l'unica offerta di servizi di tipo cloud; ovviamente, ve ne sono altre e, sempre abbastanza ovviamente, esse differiscono per tipi di servizi offerti e/o per livello di configurabilità. In questo capitolo, si analizzeranno sinteticamente le offerte di Amazon (Amazon Web Services, AWS) e di Google (Google App Engine, GAE), focalizzandosi sul lato storage di tali offerte.

## 8.1 Amazon Web Services

AWS è una collezione di servizi web, i quali, messi insieme, costituiscono una piattaforma per il cloud computing. Tra i vari servizi offerti, i più noti e famosi sono Amazon EC2 (Elastic Compute Cloud) e Amazon S3 (Simple Storage Service); inoltre, tutti i servizi forniti possono essere acceduti usando HTTP e protocolli basati su REST o su SOAP.[1]

### 8.1.1 AWS e Windows Azure

Dovendo rapportare Windows Azure con AWS, si potrebbe semplicemente riassumere il confronto dicendo che Windows Azure è un "sottoinsieme" di AWS: tranne alcuni dettagli trascurabili e non (si vedranno in seguito quelli

[1] SOAP (Simple Object Access Protocol) è la specifica per protocolli atti allo scambio di informazioni strutturate.[19]


Figura 8.1: Logo di Amazon Web Services.

non trascurabili), AWS può tranquillamente replicare i servizi di Windows Azure, e offrirne altri più complessi. In altre parole, una persona ben pagata o con una grande volontà potrebbe realizzare una componente avente interfaccia compatibile con Windows Azure, ma usando AWS come *backend* per le operazioni.

AWS, a differenza di Windows Azure, non si pone tanto come servizio PaaS, ma piuttosto come servizio IaaS, per via della grande varietà e configurabilità dei servizi. Tuttavia occorre fermarsi qui, perché, in generale, non si può fare un confronto puntuale tra le due piattaforme, in quanto ad esso andrebbe dedicata una relazione a parte.

### 8.1.2 Differenze notevoli

La prima differenza di cui si deve tenere conto è il fatto che la consistenza di una lettura subito dopo una scrittura non è garantita, mentre su Windows Azure lo è; più precisamente, lo è solo per alcune operazioni in alcuni centri dati: sinteticamente, tuttavia, la si deve considerare come non garantita.

Vi sono poi molte altre differenze, meno eclatanti di quella di cui si è appena parlato. Ad esempio, il servizio Amazon SimpleDB offre un qualcosa di molto simile alle tabelle di Windows Azure; tuttavia, esso fornisce un'indicizzazione più accurata (le tabelle di Windows Azure indicizzano solo PartitionKey e RowKey) e la capacità di eseguire delle operazioni server side, come il conteggio e l'ordinamento.

Va fatto notare, infine, che AWS ha subito diversi fallimenti, l'ultimo risale all'aprile dell'anno corrente (2011), in base a quanto riportato su [20];


Figura 8.2: Logo di Google App Engine.

in generale, AWS ha subito più fallimenti di Windows Azure. Questo dato va tuttavia preso con le pinze, perché si deve considerare anche che AWS è attivo dal 2006, mentre Windows Azure lo è dal 2009.

## 8.2 Google App Engine

GAE ha come obiettivo principale separare il più possibile la gestione delle applicazioni dalla gestione dell'infrastruttura e delle piattaforma sottostanti; si vuole fare in modo che l'utente debba pensare soltanto alla logica della propria applicazione, avendo la garanzia che la propria applicazione "girerà" sulle stesse basi che sorreggono i servizi di Google. Inoltre, GAE offre il supporto per i linguaggi che si possono compilate in ByteCode (come Java e Scala), Python e Go[2].

Tra i servizi offerti, si hanno lo scaling automatico, uno storage persistente, integrazione con Google Account, una piattaforma che emula localmente Google App Engine (dedicata agli sviluppatori) e altre caratteristiche interessanti, come la possibilità di scatenare eventi a intervalli prestabiliti.

Ora, si vedrà, come ci si sarebbe potuto aspettare, la parte relativa allo storage dei dati.

[2]Go è un linguaggio staticamente tipato e compilato, definito e sviluppato da Google. Nonostante implementi alcune caratteristiche utili per realizzare programmi concorrenti (come le "goroutine" e i canali), non sta avendo molto successo tra gli sviluppatori. Piccola curiosità: il progetto è gestito da Rob Pike, creatore del sistema operativo Inferno e del linguaggio di programmazione Limbo.

### 8.2.1 Storage su GAE

GAE fornisce un servizio di storage distribuito, il quale consente sia operazioni di interrogazione, sia operazione di manipolazione dei dati. Allo stesso modo con cui, grazie allo scaling automatico, i web server aumentano all'aumentare del traffico di rete, così il *datastore* (nome dell'offerta storage di GAE) cresce al crescere dei dati. Si vedrà che GAE fornisce un solo servizio di storage, anteponendosi sia a Windows Azure, sia ad AWS, i quali offrono molteplici e differenti servizi per la memorizzazione dei dati.

### 8.2.2 Modelli di storage

Il servizio di storage di GAE può funzionare in due differenti modalità:

- Master/Slave
- High Replication

Nel primo, solo un centro dati può scrivere sui dati (Master), e le nuove informazioni vengono replicate in maniera asincrona su altri dispositivi (Slave). Questo modello offre una forte consistenza, col rischio che i dati siano non disponibili nel caso di un fallimento del centro dati Master o di un downtime programmato. Questo metodo è molto simile a quello proposto da Windows Azure.

Nel secondo invece, i dati sono replicati su vari centri dati, garantendo una grande disponibilità per le operazioni di lettura e di scrittura, ma non è garantita la consistenza immediata (prima o poi, il sistema tornerà consistente).

### 8.2.3 Datastore e tabelle di Windows Azure

Proprio come le tabelle di Windows Azure, allo stesso modo il datastore non è un database relazionale. Esso può memorizzare entità, aventi un certo tipo ed un certo insieme di proprietà; le query recuperano le entità di un certo tipo, filtrandole ed ordinandole secondo i valori delle loro proprietà.

| Risorsa | Unità di misura | Costi unitari |
|---|---|---|
| Tempo di CPU | Ore CPU | 0.10$ |
| Dati memorizzati | Gigabyte al mese | 0.15$ |
| Storage High Replication | Gigabyte al mese | 0.45$ |

Tabella 8.1: Costi del datastore di GAE.

Sempre come nelle tabelle, anche nel datastore le entità non hanno uno schema; la struttura delle varie entità deve essere fissata e verificata dall'applicazione.

### 8.2.4 Transazioni

Le transazioni offerte da GAE sono molto più flessibili e vicine all'abituale transazione relazionale di quanto non lo siano quelle fornite da Windows Azure. Infatti, non sussistono alcune delle limitazioni imposte da Windows Azure; in particolare, è possibile avere una transazione che include sia operazioni di manipolazione, sia operazioni di interrogazione.

### 8.2.5 Costo del servizio

Per un elenco dei costi dello storage di GAE si veda tabella 8.1, mentre si ricorda che in sezione 4.5 sono esposti i costi dello storage di Windows Azure.

# Capitolo 9

# Progetto dimostrativo

Nel campo dell'Informatica, come nel resto delle cose, i fatti (o, per meglio dire, il codice) tendono ad avere più importanza delle parole. Pertanto, allegato a questa relazione (decisamente ricca di parole, come si sarà oramai accorto il lettore) si trova un piccolo progetto esplicativo scritto in C# usando la libreria .NET per la piattaforma Windows Azure; in particolare, esso è stato pensato in modo tale da "sfruttare" i servizi per lo storage di Windows Azure: i blob (capitolo 4), le tabelle (capitolo 5) e le code (capitolo 6).

## 9.1 Visione generale

Il progetto esplicativo, denominato *Disibox*, consiste in una semplice implementazione di Dropbox[1], la quale trascura la parte di sincronizzazione dei file. In particolare, si è implementata, per rimanere in tema con la relazione, la parte "storage" di tale progetto, cioè, si sono create una serie di librerie per gestire i dati memorizzati da un servizio simile.

Infatti, il progetto, dovendo imitare Dropbox, ha bisogno di memorizzare almeno gli utenti (in particolare, le loro credenziali di accesso) e i loro file. Tuttavia, un lettore acuto si può già essere accorto che, se i dati da memorizzare fossero soltanto gli utenti e i file, si lascerebbe fuori dall'esempio

[1] Dropbox è un noto servizio per la sincronizzazione dei file tra più personal computer; un po' meno noto è il fatto che tale servizio si appoggia ad Amazon Web Services per le proprie necessità di memorizzazione.

Figura 9.1: Logo di Dropbox, servizio che il progetto esplicativo cerca di imitare.

un servizio di memorizzazione, le code. Gli utenti sono comodamente rappresentabili con una tabella, mentre i file trovano un naturale *alloggio* tra i blob.

Al fine di poter utilizzare anche le code, si è deciso di aggiungere un'ulteriore funzionalità a Disibox che non è presente su Dropbox: il *processing* dei file memorizzati. In pratica, preso un file e una libreria contenente degli strumenti, è possibile far processare tale file agli strumenti contenuti nella libreria, purché gli strumenti siano in grado di *elaborare* file come quello in questione. Ad esempio, un caso d'uso è l'inversione dei colori: presa un'immagine, tramite il processing è possibile invertirle i colori, cosa che non sarebbe possibile se il file non fosse un'immagine ma un file di testo.

In questo problema strettamente di *computazione*, la libreria dati interviene su due fronti: per prima cosa, offre la possibilità di memorizzare le DLL contenenti gli strumenti e gli output del processing, mentre, per seconda cosa, fornisce dei canali di comunicazione (basati sulle code) che i *ruoli* dedicati al processing possono usare per comunicare.

Ricapitolando, le librerie implementate offrono le seguenti funzionalità:

- Manipolazione e lettura dei dati relativi agli utenti;
- Manipolazione e lettura dei file salvati dagli utenti, delle DLL contenenti gli strumenti e degli output generati dal processing;

- Due canali per lo scambio di messaggi, sempre nell'ambito del processing.

Nella sezione seguente si scenderà un pochino più nei dettagli relativi alle librerie; per ciascuna, si vedrà quali sono i suoi usi e gli eventuali dettagli implementativi, se di interesse.

## 9.2 Librerie implementate

Si è deciso, per questioni di modularità logica ma anche di *sicurezza*, di dividere l'implementazione in tante piccole librerie, i cui scopi sono ben contenuti. In totale, sono presenti cinque librerie:

- `Disibox.Data`
- `Disibox.Data.Client`
- `Disibox.Data.Server`
- `Disibox.Data.Setup`
- `Disibox.Data.Tests`

Nelle parti seguenti, ciascuna di loro verrà descritta e commentata.

### 9.2.1 Disibox.Data

Utilizzata da tutte le altre libreria, questa rappresenta la base dell'implementazione, in quanto contiene tutte le classi wrapper verso la libreria .NET, insieme ad alcune funzioni per la verifica dei prerequisiti. Tra i wrapper presenti, si trova la tabella fortemente tipata, usata per memorizzare gli utenti ed una sorta di *mappa* chiave-valore; tale mappa, attualmente, conserva solo una coppia, dove si trova l'ultimo id utente generato.

### 9.2.2 Disibox.Data.Client

Come suggerisce il nome, questa libreria presenta le funzionalità necessarie (ma soprattutto, sufficienti) ai client, cioè, eventuali interfacce grafiche o Web. Come mostrato in listato 9.1, essa consente pieno controllo sugli utenti, i file e le DLL di strumenti, mentre solo la lettura e la cancellazione sono consentite per gli output. Infine, considerato che tutte le operazioni sopra esposte richiedono un utente in possesso delle credenziali per poter essere eseguite, sono offerti dalla libreria anche i metodi per effettuare il login ed il logout.

### 9.2.3 Disibox.Data.Server

Analogamente alla libreria precedente, anche in questo caso il nome è un indizio del suo uso. Infatti, questa libreria è stata pensata per essere impiegata dai *ruoli*, come quello che si occuperà del processing. Quindi, essa contiene le chiamate per sfruttare le code di messaggi, oltre ad alcune chiamate necessarie per il processing, come la lettura di un dato file, l'aggiunta di un output e la lettura di tutte le DLL di strumenti disponibili. Listato 9.2 presenta un sunto dei metodi offerti.

### 9.2.4 Disibox.Data.Setup

Le librerie appena descritte (Client e Source) suppongono che sulla cloud siano già presenti tutti gli elementi necessari alla memorizzazione, come, ad esempio, la tabella per gli utenti. Pertanto, è necessario "tirare su" tali elementi, almeno una volta nella vita del servizio. Di questo, si occupa la libreria Setup, che è di tipo *ibrido*, nel senso che si può sia lanciare come eseguibile (si veda figura 9.2), sia usare all'interno di altro codice (come fanno i test per avere ogni volta *tabula rasa*).

La libreria crea i seguenti elementi, in modo *non distruttivo* (cioè, se un elemento esiste già non lo si cancella e lo si ricrea):

- Un blob container per i file, uno per gli output ed uno per le DLL degli strumenti;

**Algoritmo 9.1** ClientDataSource, la classe principale della libreria Client.

```
1  public class ClientDataSource
2  {
3      public ClientDataSource() { ... }
4
5      public string AddFile(string fileName, Stream fileContent,
6                            bool overwrite) { ... }
7      public bool DeleteFile(string fileUri) { ... }
8      public Stream GetFile(string fileUri) { ... }
9      public IList<FileMetadata> GetFileMetadata() { ... }
10
11     public void DeleteOutput(string outputUri) { ... }
12     public Stream GetOutput(string outputUri) { ... }
13
14     public void AddProcessingDll(string dllName, Stream dllContent,
15                                  bool overwrite) { ... }
16     public void DeleteProcessingDll(string dllName) { ... }
17     public Stream GetProcessingDll(string dllName) { ... }
18     public IList<string> GetProcessingDllNames() { ... }
19
20     public void AddUser(string userEmail, string userPwd,
21                         UserType userType) { ... }
22     public void DeleteUser(string userEmail) { ... }
23     public IList<string> GetAdminUsersEmails() { ... }
24     public IList<string> GetCommonUsersEmails() { ... }
25
26     public void Login(string userEmail, string userPwd) { ... }
27     public void Logout() { ... }
28 }
```

**Algoritmo 9.2** ServerDataSource, la classe principale della libreria Server.

```
1  public class ServerDataSource
2  {
3      public ServerDataSource() { ... }
4
5      public bool UserExists(string userEmail, string userPwd) { ... }
6
7      public ProcessingMessage DequeueProcessingRequest() { ... }
8      public void EnqueueProcessingRequest(ProcessingMessage procReq) { ... }
9      public IList<ProcessingMessage> PeekProcessingRequests() { ... }
10     public void ClearProcessingRequests() { ... }
11
12     public ProcessingMessage DequeueProcessingCompletion() { ... }
13     public void EnqueueProcessingCompletion(ProcessingMessage procCompl) { ... }
14     public IList<ProcessingMessage> PeekProcessingCompletions() { ... }
15     public void ClearProcessingCompletions() { ... }
16
17     public Stream GetFileFromUri(string fileUri) { ... }
18
19     public string AddOutput(string toolName, string outputContentType,
20                             Stream outputContent) { ... }
21
22     public Stream GetProcessingDll(string dllName) { ... }
23     public IEnumerable<string> GetProcessingDllNames() { ... }
24 }
```

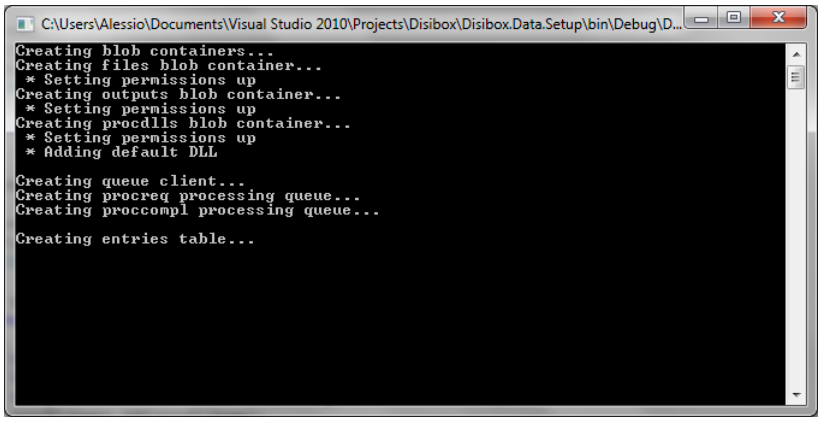



Figura 9.2: Istantanea durante l'esecuzione di Setup.

- Una tabella per gli utenti, includente l'admin di default, e la tabella *registro*, includente la riga corrispondente all'ultimo id utente generato;
- Le due code usate per lo scambio di messaggi durante il processing.

### 9.2.5 Disibox.Data.Tests

Questa libreria contiene alcuni test usati per provare i metodi offerti dalle librerie Client e Server. Come già detto, questa libreria usa Setup per ricreare, ad ogni test, una cloud vuota, ma contenente almeno le strutture necessarie al servizio Disibox.

# Appendice A

# Repository codice sorgente

Nel corso della relazione, si è fatto riferimento ad un piccolo progetto e ad alcuni brevi listati di codice, entrambi sviluppati per fini esplicativi. Tutto il codice ad essi relativo può essere trovato al seguente indirizzo:

http://disibox.googlecode.com/svn/

Tale indirizzo punta ad un repository svn, gratuitamente fornito dal servizio Google Code. Il repository è strutturato in diverse cartelle, ma il lettore troverà di interesse prevalentemente le seguenti:

- *trunk*, dove si trova la versione più recente del progetto esplicativo;
- *examples*, dove sono raccolti tutti gli esempi presentati nella relazione (insieme a del codice di "contorno" per poterli eseguire).

# Ringraziamenti

L'autore vorrebbe ringraziare:

- Google, per aver cercato di rispondere a ogni suo quesito sugli argomenti qui trattati;
- Il barattolo dei biscotti, per avergli fornito le energie necessarie a portare a termine la relazione;
- Artur, compagno di corso ma soprattutto amico, per averlo accompagnato e supportato nel corso di questa "avventura";
- La sua famiglia (mamma, papà e Matteo), per averlo sopportato nei (frequenti) momenti di delirio; in particolare, l'esempio del panificio è nato dal mestiere portato avanti da suo padre;
- Marta, ragazza davvero speciale e cara, per avergli riportato il sorriso nei momenti difficili.

# Bibliografia


[1] http://www.cloudtweaks.com/2011/02/a-history-of-cloud-computing/

[2] http://en.wikipedia.org/wiki/Cloud_computing

[3] http://en.wikipedia.org/wiki/Hybrid_Cloud

[4] http://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)

[5] http://en.wikipedia.org/wiki/Grid_computing

[6] http://www.bus.emory.edu/ram/

[7] http://msdn.microsoft.com/en-us/data/ee844254

[8] http://en.wikipedia.org/wiki/Microsoft_Open_Specification_Promise

[9] http://aws.amazon.com/message/65648/

[10] David Chappell, "Introducing the Windows Azure Platform"

[11] http://blog.smarx.com/posts/building-running-and-packaging-windows-azure-applications-from-the-command-line

[12] http://azurestorageexplorer.codeplex.com/

[13] http://cloud.neudesic.com/

[14] http://www.windowsazure4j.org/

[15] http://msdn.microsoft.com/en-us/library/dd672588.aspx

[16] http://msdn.microsoft.com/en-us/hh184848

[17] http://www.microsoft.com/windowsazure/pricing/

[18] Chris Hay, Brian H. Prince, "Azure in Action"

[19] http://en.wikipedia.org/wiki/SOAP_(protocol)

[20] http://en.wikipedia.org/wiki/Amazon_Web_Services