

# Capitolo 1

## Introduzione al cloud computing

### L'annuncio

Nel mese di ottobre del 2008, a Los Angeles, durante la prima giornata della Microsoft Professional Developer Conference (PDC) 2008, è stato ufficializzato l'annuncio trapelato già qualche mese prima in vari blog e sulla stampa: anche Microsoft ha in cantiere una piattaforma per il *cloud computing*. Il nome definitivo sarà Azure Services Platform e consta di una serie di servizi per ospitare applicazioni di qualunque livello: da semplici siti web ospitati sui loro server (vedremo più avanti che il termine server è riduttivo), fino ad applicazioni enterprise ad alta affidabilità e scalabilità, che possono sfruttare meccanismi di queueing per disaccoppiare il front-end dal back-end. Il tutto è condito da un concetto di sistema operativo "in the cloud" che racchiude le funzionalità di base (Windows Azure) e da una serie di servizi che vanno da SQL Azure a .NET Workflow Services. Chiariremo meglio questi componenti nel Capitolo 2 dedicato alla panoramica sull'intera piattaforma.

La definizione di *cloud computing* riportata da Wikipedia è la seguente:

*"In informatica, con il termine **cloud computing** si intende un insieme di tecnologie informatiche che permettono l'utilizzo di risorse distribuite (storage, CPU).*

*La caratteristica principale di tale approccio è rendere disponibili all'utilizzatore tali risorse come se fossero implementate da sistemi (server o periferiche personali) "standard". L'implementazione effettiva delle risorse non è definita in modo dettagliato; anzi l'idea è proprio che sia un insieme eterogeneo e distribuito – the cloud, in inglese "nuvola" – di risorse le cui caratteristiche non sono note all'utilizzatore".*

La definizione, anche se lontana dalla perfezione, mette in luce due aspetti importanti: l'utilizzo di risorse distribuite e risorse le cui caratteristiche non sono note all'utilizzatore.

Il primo aspetto si riferisce alle possibilità offerte dalle piattaforme di *cloud computing* di distribuire le risorse su vari server virtuali, al fine di garantire alta affidabilità e alta scalabilità.

Il secondo aspetto, sicuramente da chiarire meglio, fa riferimento a caratteristiche fisiche che non solo non sono note all'utilizzatore, (che in fondo è ignaro anche delle caratteristiche fisiche dei tradizionali server su cui è ospitata l'applicazione che usa), ma non sono note neanche allo sviluppatore della soluzione stessa! Con un po' di pazienza continuando a leggere, scoprirete che l'aspetto più interessante del *cloud computing* è proprio l'indipendenza dalle risorse fisiche.

## Visione a lungo termine

Immaginando il futuro a lungo termine, proiettandoci in avanti di qualche decennio, possiamo ipotizzare uno scenario in cui la "fisicità" dei dati e dei programmi sia stata in qualche modo eliminata del tutto, almeno dal punto di vista dei fruitori. Questi ultimi, poi, non necessariamente sono persone fisiche, ma possono essere altre applicazioni.

La terminologia che abbiamo adottato è quella dei servizi. L'acronimo oggi in voga è SOA (Service Oriented Architecture), che già definisce un ecosistema di servizi interconnessi e fruibili *anche* da applicazioni che interagiscono con un utente finale. Questo tipo di architetture definisce un concetto di interoperabilità anche tra sistemi diversi. Ciascun servizio, però, al suo interno è implementato su un sistema specifico, in un certo linguaggio, con un particolare hardware e un particolare Sistema Operativo. I componenti al suo interno comunicano tra loro utilizzando, il più delle volte, uno standard che non è lo stesso usato per l'infrastruttura "SOA". Per fare un esempio, il modo in cui i componenti .NET dialogano tra loro all'interno di un'applicazione è diverso da ciò che avviene tra componenti scritti in Java o che utilizzano COM. Tutti, però, possono aderire a dei contratti di comunicazione che garantiscono la reciproca interoperabilità.

L'evoluzione dei linguaggi, dei sistemi operativi e dei framework ha già portato a un'evoluzione, che nel tempo ha reso sempre meno importanti molti dettagli della piattaforma necessaria all'esecuzione di un programma: penso ai compilatori di linguaggi evoluti per evitare la specificità del codice macchina di ciascun processore, ai sistemi operativi per generalizzare, condividere e semplificare l'accesso alle risorse logiche e fisiche, o ai runtime (come Java Virtual Machine e Common Language Runtime) per generalizzare i sistemi operativi sottostanti. Il salto prevedibile, ancora lontano da raggiungere, ma ormai perfettamente ipotizzabile, è l'indipendenza dalla ubicazione fisica delle risorse.

A cosa porta tutto questo? All'idea di avere un sistema operativo *distribuito* su cui eseguire le proprie applicazioni. Con il termine *distribuito* non si intende solo l'idea dell'esecuzione del codice suddivisa tra più processori, magari su server diversi. Il concetto di sistema operativo distribuito è applicabile a tutte le risorse tipicamente controllate dal sistema operativo (CPU, memoria, I/O e quindi storage e comunicazione con altri processi). Attenzione, perché non è così importante che dal punto di vista implementativo si scelga la strada di avere un unico sistema monolitico (che utilizza contemporaneamente tante CPU e tanti dischi su tanti server fisicamente dislocati in posti diversi) piuttosto che quella di creare un sistema distribuito più in una logica "peer-to-peer", dove ogni nodo partecipa al sistema complessivo ma senza una struttura gerarchica predefinita. Queste scelte sono dettagli implementativi. Dal punto di vista del fruitore di un sistema simile (il nostro programma o servizio) l'importante è poter richiedere l'esecuzione di una certa operazione richiedendo determinati livelli di servizio (utenti concorrenti, tempi di risposta, spazio di memorizzazione, ecc.). Se queste richieste sono soddisfatte, il fatto che siano ottenute con l'uso di un unico enorme mainframe o con una rete di piccoli server può essere irrilevante.

In un mondo completamente connesso, con velocità di trasferimento infinita e latenza quasi nulla, qualsiasi applicazione può essere eseguita ovunque senza che ciò ne pregiudichi i risultati. In questo mondo, però, resterebbe necessario definire come andrebbero scritti i programmi perché possano interagire tra loro ed essere eseguiti in maniera "distribuita". Qui entra in gioco la necessità di un sistema operativo "in the cloud", cioè di qualcosa che offra quei servizi di base necessari a qualsiasi applicazione su un hardware tradizionale, riproducendo la stessa metafora di risorse in un ambiente di esecuzione realmente distribuito.

La realizzazione di un sistema simile presuppone che l'accesso a una qualsiasi risorsa sia, in effetti, la richiesta di accesso a un servizio. In pratica, è come applicare SOA fino alle estreme conseguenze, rendendo fruibile come servizio fino all'ultima API di un sistema operativo tradizionale. Benché questo non sia applicabile realmente per qualsiasi funzionalità, la visione dei sistemi "in the cloud" va esattamente in questa direzione. Un sistema operativo creato "ad hoc" per questo scopo consente di semplificare la scrittura di applicazioni che utilizzino questo tipo di architettura, mantenendo un paradigma di programmazione del tutto simile a quello attuale (dove si presuppone di avere un accesso quasi diretto alle risorse fisiche) e ottenendo un'implementazione molto più SOA-oriented anche da applicazioni "monolitiche" tradizionali.

In questo modello, l'idea di fondo è scrivere un programma, mandarlo in esecuzione, e da quel momento potersi disinteressare di dove è in esecuzione e di dove sono i dati. Dal momento dell'attivazione, il programma resta "in vita" senza che l'autore si debba più preoccupare della sua manutenzione. Bug a parte, ovviamente.

## Cosa abbiamo oggi

Nel modello futuristico che abbiamo descritto, non c'è più ragione di avere dei server in un'azienda. Il loro ruolo è svolto dai servizi "in the cloud". L'elemento critico diventa la connettività con il "cloud".

La realtà attuale, però, è molto diversa. Vale la pena descrivere alcuni scenari comuni in cui oggi un'applicazione viene eseguita, che è quindi anche quello per cui comunemente viene progettata.

Le grandi aziende dispongono di almeno un reparto dedicato ai sistemi informativi, che copre tutte le aree del ciclo di vita di un'applicazione (pianificazione, progettazione, sviluppo e manutenzione). Le aziende medie hanno di solito almeno un piccolo reparto che cura la manutenzione (qualche sistemista), acquisendo dall'esterno procedure standard o personalizzate. Le aziende più piccole richiedono servizi di manutenzione per il mantenimento della loro piccola infrastruttura, che usa software standard. Tutte queste aziende, però, di solito hanno una rete interna (locale e/o geografica) all'interno della quale sono collocati sia i client sia i server che ospitano i servizi applicativi (o almeno i dati) usati dagli utenti.

L'esecuzione di un servizio su un server che è fisicamente all'interno dell'azienda, o è comunque direttamente controllato dalla stessa, viene detto "*on premise*". Con questo termine si vuole creare una distinzione tra l'accesso a un servizio ospitato su macchine dedicate e completamente controllate (*on premise*) e lo stesso servizio ospitato su uno o più server di un provider esterno (*in the cloud*).

Allo stato attuale, la maggior parte dei servizi usati da un'azienda è *on premise*. File server, software gestionale, database, intranet, posta elettronica e sistemi di collaborazione sono il più delle volte ospitati su server aziendali. Di conseguenza, anche le applicazioni aziendali più personalizzate sono costruite con la stessa logica, già per il solo fatto di dover accedere a dati che risiedono su server aziendali. Questa strategia presenta i suoi limiti quando è necessario interagire al di fuori del perimetro aziendale. Accessi VPN e il concetto di *extranet* sono soluzioni che aggirano il problema di partenza (la rete aziendale è chiusa), ma non sono sempre utilizzabili. Quando le applicazioni devono interagire tra loro (si pensi a un sistema di e-procurement, a un sito che pubblica un listino prezzi, all'acquisizione di ordini via web, allo scambio di informazioni con clienti e fornitori, ecc.), il problema di come far comunicare i sistemi interni (*on premise*) con quelli di terze parti genera il problema di dove e come sviluppare i servizi necessari. L'infrastruttura esistente, spesso, diventa un vincolo forte sulle scelte architetturali e tecnologiche per lo sviluppo di tali servizi.

## Cosa succede quando qualcosa è già “on the cloud”

In realtà, esistono già delle situazioni ibride in cui la maggioranza dei sistemi *on premise* deve interagire con dei servizi esterni, che potremmo già definire *on the cloud*, in quanto non ne conosciamo l'esatto dimensionamento e la reale ubicazione. È il caso dell'accesso a servizi remoti divenuti famosi con il termine di Web Service, termine che ne caratterizza principalmente l'accessibilità attraverso il protocollo HTTP. Ma a prescindere dallo standard, se si pensa alle interazioni che avvengono in una transazione di e-commerce (pagamento, eventuali ordini inviati ai fornitori e ai servizi di logistica, ecc.) è evidente come l'interoperabilità tra servizi di enti diversi sia una realtà già affrontata e risolta.

In tutti questi scenari, però, il livello d'interoperabilità è quello dei servizi esplicitamente esposti attraverso specifici contratti. Esistono degli standard di comunicazione che semplificano la definizione di questi contratti (per es. SOAP) ed esistono dei framework applicativi e delle librerie che semplificano l'implementazione di tali contratti (per es. Windows Communication Foundation in .NET) almeno per quanto riguarda l'aspetto relativo alla comunicazione.

L'implementazione di un servizio, oggi, è sempre fatta presupponendo che sia poi eseguito *on premise*. Gli strumenti, i framework di sviluppo, e la nostra stessa mentalità, ci portano a ragionare esclusivamente in questi termini. Diventa quindi un nostro problema, sistemistico se non addirittura applicativo, dotare il servizio di tutte quelle caratteristiche (scalabilità, affidabilità, prestazioni) necessarie a renderlo fruibile rispettando i livelli di servizio attesi. Per esempio, oggi siamo abituati a investire in hardware scegliendo accuratamente il dimensionamento delle varie risorse partendo da CPU, RAM, disco fisso per arrivare alla tipologia di RAID da utilizzare sui dischi, valutando l'impiego di SAN e dettagliando le specifiche della fibra ottica da utilizzare in un cluster. Tutte queste attività (pianificazione, acquisto, manutenzione) hanno un costo significativo e richiedono una competenza che il più delle volte va al di là delle conoscenze di chi progetta e sviluppa il servizio stesso.

## Come funziona Windows Azure

Windows Azure è la promessa di un sistema operativo *in the cloud*, che rende la scelta delle caratteristiche necessarie a garantire un determinato livello di servizio qualcosa di molto più semplice. Rispetto all'esempio precedente, per un servizio è necessario scegliere il numero di istanze parallele di cui si ha bisogno: il numero delle istanze è configurabile nel tempo, adattando le prestazioni all'incremento o alla contrazione delle richieste o dei dati.

I costi sono proporzionali al modello di configurazione scelto, che può essere adattato nel tempo. L'onere della gestione fisica delle macchine e degli strumenti di amministrazione è compito di chi offre la piattaforma, nel nostro caso Microsoft che offre Windows Azure.

Questo coinvolge anche l'aspetto della manutenzione. Sicuramente nessuno meglio di Microsoft conosce il .NET Framework, il kernel del sistema operativo e servizi come Internet Information Server, SQL Server e così via. Non occorre installare patch o configurare fisicamente le macchine e i servizi; per configurare le applicazioni è sufficiente descrivere "i nostri bisogni" nel modello. Per esempio, potremmo indicare che abbiamo bisogno di 100 GB di storage per memorizzare delle immagini: non occorre specificare il tipo di disco, la macchina fisica su cui è installato il disco o condividere una directory per l'accesso via rete.

Rispetto alla visione a lungo termine descritta prima, Windows Azure definisce implicitamente l'accesso ad alcune risorse come *accesso a servizi*, rendendo la cosa così trasparente al programmatore da renderla del tutto simile all'accesso a risorse locali. Le risorse fisiche sono nascoste dall'infrastruttura che, esponendo semplici metodi, consente al nostro codice di non dover conoscere:

- l'ubicazione fisica delle risorse
- le directory su disco
- le Virtual Directory
- i percorsi di rete
- i nomi dei server

Per rispondere subito alle domande che sorgono spontaneamente dopo queste affermazioni, è sufficiente una sola risposta: non è necessario conoscere *dove* e neanche *come* sono memorizzate le informazioni, o come sono gestite le risorse.

Facciamo un parallelo con qualcosa che conosciamo tutti: in .NET non è così importante sapere come lavora internamente il Garbage Collector (GC). L'importante è rilasciare correttamente gli oggetti in modo che il GC possa fare egregiamente il suo lavoro; non è necessario suggerire al GC quando intervenire e soprattutto è controproducente farlo. *In the cloud* non è necessario sapere il percorso fisico per raggiungere il file system, ma è importante aprire e chiudere nel modo corretto i file che utilizziamo dall'applicazione.

Raggiungere una risorsa implica la chiamata a un servizio che espone la risorsa stessa: tecnicamente, faremo sempre riferimento a un servizio tramite un URI per inserire, modificare, cancellare e leggere le informazioni che il servizio espone.

Questo vuol dire non poter usare il codice attuale? La risposta, come sempre, è: dipende. Dipende da come abbiamo scritto il codice; se un'applicazione usa dei servizi per eseguire delle operazioni il codice resta immutato: sarà necessario solo aggiornare i file di configurazione per puntare a nuovi indirizzi esposti *in the cloud*. Se, al contrario, l'applicazione è stata scritta seguendo le regole di astrazione, sarà sufficiente adattare i livelli più bassi alle nuove metodologie di accesso. Se invece abbiamo un'applicazione monolitica, con il codice di accesso alle risorse fisiche sparso in vari moduli, sarà sicuramente più complesso il porting di tale applicazione al modello *in the cloud*.

Per fare un esempio concreto: se la nostra applicazione utilizza uno strato separato per l'accesso ai dati, possiamo semplicemente rimpiazzare questo componente con le classi adatte all'accesso *in the cloud*. Se la nostra applicazione non espone metodi centralizzati per accedere alle informazioni, è molto probabile che occorra una rivisitazione globale.

Nel corso dei vari capitoli vedremo all'opera l'accesso al file system e soprattutto il ruolo del file system, l'accesso alle risorse di tipo *Blob*, *Table*, *Queue* e l'accesso a SQL Azure. Nel Capitolo 2 daremo uno sguardo all'intera piattaforma e chiuderemo la trattazione con un pratico esempio architetturale basato sulla nostra applicazione "DevCon 2009", composta da servizi WCF, pagine ASP.NET e client Silverlight. L'applicazione è stata portata su Windows Azure e memorizza le informazioni sul Cloud Storage.