Cloud and Green Computing

Alessio Delgadillo

Anno accademico 2020-2021

Indice

Capitolo 1

Introduzione

Negli ultimi anni l'economia non è più incentrata sui beni, ma è incentrata sui servizi. È nata una tendenza a vedere "tutto come un servizio": bike sharing, car sharing, streaming musicale, cloud storage, ...

La **Quality of Service** (QoS) è fondamentale per qualsiasi servizio che utilizziamo, non basta che sia conveniente economicamente ma deve anche essere affidabile. Le informazioni sull'affidabilità vengono scritte nel contratto che i clienti spesso (quasi sempre) ignorano e accettano senza leggere.

I clienti non sanno (e spesso non vogliono sapere) come è implementato il servizio che usano, scelgono (o dovrebbero scegliere) se usarlo o meno sulla base del contratto che è (o dovrebbe essere) esposto da chi fornisce il servizio. I termini di un contratto vengono definiti nel **Service Level Agreement** che è formulato da esperti. Un esempio di Service Level Agreement può essere quello di Facebook:

"*... quando l'utente condivide, pubblica o carica un contenuto protetto da diritti di proprietà intellettuale in relazione o in connessione con i Prodotti di Facebook, concede una licenza non esclusiva, trasferibile, sub-licenziabile, non soggetta a royalty e valida in tutto il mondo per la trasmissione, l'uso, la distribuzione, la modifica, l'esecuzione, la copia, la pubblica esecuzione o la visualizzazione, la traduzione e la creazione di opere derivate dei propri contenuti...*"

Perché sono così importanti i dati che vengono messi sui Social Network?

Nei colloqui di assunzione, alcuni specialisti di risorse umane cercano di valutare com'è una persona inizialmente dai Social Network. Una volta si facevano delle domande per capire il tipo di persona che avevamo davanti, ma oggi è tutto trasparente sui Social.

1.1 Cloud Computing

La richiesta di un servizio può variare nel tempo, **nessuno** sa come. Quindi solitamente ci troviamo davanti due grandi problemi:

- **Overprovisioning**: cercare di assicurarsi di soddisfare i picchi attesi di richieste (dovuti a pattern giornalieri, stagionali o picchi inattesi) porta a sprecare risorse (se la stima è corretta – ancora peggio se la richiesta è sovrastimata).
- **Underprovisioning**: se la richiesta è sottostimata allora underprovisioning può accidentalmente **respingere utenti**. Costo di underprovisioning più difficile da misurare, ma serio come quello di overprovisioning; non solo gli **utenti respinti** non generano entrate, **potrebbero non tornare mai più**.

Questi problemi erano molto più frequenti finché le risorse non erano virtualizzate, ma tutto è cambiato quando si è evoluto il Cloud. Il Cloud Computing offre una quantità di risorse, apparentemente infinite, e disponibili su richiesta. Il NIST (National Institute of Standards and Technology) l'ha definito come:

"*Un modello per permettere un accesso via rete diffuso, conveniente e su richiesta a un insieme condiviso di risorse di calcolo configurabili. Queste risorse possono essere rapidamente fornite e rilasciate con un minimo costo di gestione o di interazione col fornitore del servizio.*"

Idee chiave

Le idee chiave del Cloud Computing:

- raggruppare in modo efficiente e on-demand infrastrutture virtuali, offerte come servizi;
- fornire risorse dinamicamente scalabili, virtualizzate a molti clienti attraverso Internet;
- separare la fornitura dei servizi di calcolo dalla tecnologia sottostante (gli utenti non vedono l'infrastruttura).

Attrattiva economica

Il Cloud Computing attrae gli utenti per due motivi principali

- Eliminazione di impegni "in anticipo" da parte degli utenti (elimina le spese di capitale CapEx e vengono convertite in spese di operazione OpEx).

- **Pay-per-use**: i clienti lo amano e anche se è più costoso, il costo è compensato dai benefici economici in termini di **elasticità** e **trasferimento dei rischi**. Il rischio è spostato sul Service Level Agreement.

Modelli di Servizio

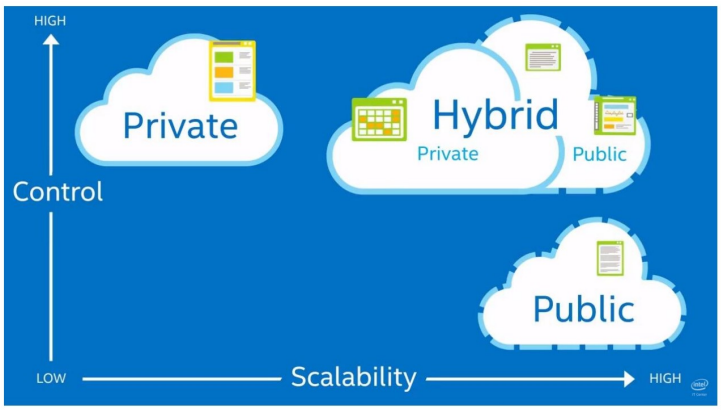
Esistono vari modelli di servizio.

ToP (Tradition on Premise): faccio tutto "in casa" e non prendo servizi esterni.

SaaS (Software as a Service) fornisce software on-demand accessibile mediante client thin o API. Il fornitore SaaS gestisce infrastruttura, sistema operativo e applicazione, mentre il cliente non è responsabile di niente.

PaaS (Platform as a Service) fornisce un'intera piattaforma come un servizio (machine virtuali, sistema operativo, servizi, ambiente di sviluppo). Il fornitore PaaS gestisce infrastruttura, sistema operativo e enabling software, mentre il cliente è responsabile di installare e gestire l'applicazione.

IaaS (Internet as a Service) fornisce server, memoria, rete (virtualizzati). Il fornitore di servizi IaaS gestisce tutta l'infrastruttura, mentre il cliente è responsabile di tutti gli altri aspetti del deployment (p.e. sistema operativo, applicazione).



Model Deployment

Esistono modelli di deployment privati e pubblici, ma oggi sta avendo molto successo il Cloud ibrido: i dati sensibili sono mantenuti in privato, per esempio dentro l'azienda, e dove è necessaria più elasticità utilizzo un cloud pubblico.

Perché si chiama Cloud?

Perché quando si disegnava lo schema client-server tutto ciò che stava fra i due veniva rappresentato come una nuvoletta per semplicità.

Ostacoli al Cloud

Confidenzialità dei dati: Dove verranno memorizzati i nostri dati, concretamente? Privacy e integrità dei dati saranno garantiti? Come? Come sapremo se si è verificato un problema?
Disponibilità dei servizi: Cosa succede se un cloud provider fallisce? Mantra dell'informatica: "no single point of failure..."
Vendor lock-in: rimanere "bloccati" con un fornitore di un servizio, per esempio dover pagare una penale per il recesso del contratto.

1.2 Nuovi Modelli di Business

Sono state innovative le idee di Dropbox, Spotify, Google. Dropbox ha deciso di offrire memoria gratuita a tutti, Spotify musica gratuita a tutti pagando i diritti ogni volta che viene riprodotta una canzone, mentre Google un motore di ricerca gratuito finanziandosi grazie al **customized advertising**.

1.3 Green Computing

Per costruire un computer ci vogliono due tonnellate di materiale grezzo, l'ICT in generale produce più emissioni di CO_2 degli aerei. Tra i fattori che contribuiscono all'inquinamento ci sono l'obsolescenza programmata e l'**obsolescenza percepita**: il venditore cerca di spingere il cliente ad acquistare un nuovo prodotto che avrà nuove funzionalità anche se al cliente queste non servono, ma il venditore farà in modo di fargli sentire questa "mancanza". Quindi c'è un gran numero di rifiuti tecnologici.

In Arizona ci sono più di quaranta Data Center perché l'elettricità costa pochissimo, chiaramente le temperature del luogo richiedono grossi sistemi di raffreddamento che saranno sicuramente economici ma per niente ecologici.

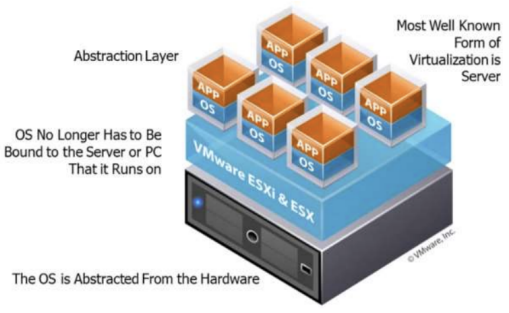
Capitolo 2

Internet as a Service

2.1 Virtualizzazione

IaaS significa creare macchine virtualizzate per i nostri servizi. La virtualizzazione è un "livello di astrazione": il sistema operativo non deve più essere associato al server/PC su cui viene eseguito, viene quindi astratto dall'hardware, cioè non è installato direttamente sull'hardware.

- **Server virtualization**: livello di virtualizzazione tra il server fisico e il sistema operativo normalmente installato.
- **Macchine virtuali**: dove vengono effettivamente installati i sistemi operativi e le applicazioni.



Modello di virtualizzazione

Un **Hypervisor** crea il livello di virtualizzazione e contiene il Virtual Machine Manager (VMM). Si possono distinguere due tipi di Hypervisor, il type 1 viene caricato direttamente sull'HW, il type 2 viene caricato su un sistema

operativo in esecuzione sull'HW. Il type 2 ha un rapporto di consolidamento maggiore/minore rispetto al type 1. Type 1 per data center, type 2 per desktop/laptop.

2.1.1 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Cloud Computing mette a disposizione server virtuali (istanze) in modo semplice, veloce e economico. La scelta del tipo d'istanza e template da utilizzare (Windows/Linux) e il numero d'istanze avviene in modo semplice con AWS management console (o interfaccia grafica). Ci sono vari tipo di istanze a seconda delle proprie esegenze e diverse modalità di pagamento

- **On demand**: ci sono molte opzioni che possiamo scegliere e ognuna ha un costo orario.
- **Reserved Instances**: significa di riservare delle instanze.
- **Spot Instances**: ci sono diversi tipo di offerte perché ci sono delle istanze che non vengono utilizzate nei datacenter.
- **Dedicated hosts**: esiste la possibilità anche di affittare un intero server per il nostro uso.

Tra le altre proprietà abbiamo lo storage persistente, Amazon Elastic Block Store (EBS), e l'autoscaling.

Company	2019 Revenue	2019 Market Share (%)	2018 Revenue	2018 Market Share (%)	2018-2019 Growth (%)
Amazon	19,990.4	45.0	15,495.0	47.9	29.0
Microsoft	7,949.6	17.9	5,037.8	15.6	57.8
Alibaba	4,060.0	9.1	2,499.3	7.7	62.4
Google	2,365.5	5.3	1,313.8	4.1	80.1
Tencent	1,232.9	2.8	611.8	1.9	101.5
Others	8,858	19.9	7,425	22.9	19.3
Total	**44,456.6**	**100.0**	**32,382.2**	**100.0**	**37.3**

Source: Gartner (August 2020)

IaaS Market share

2.1.2 Amazon Simple Storage Service (S3)

"*Everything must be made as simple as possible. But not simpler.*"

L'obiettivo di Amazon Simple Storage Service è quello di offrire un servizio virtualizzato per l'archiviazione dei dati più semplice possibile.

Amazon presenta l'idea di avere dei "*bucket*" (secchi) dove uno può trascinare i suoi dati. Mette a disposizione un'interfaccia grafica e permette di utilizzare il drag-and-drop per spostare i dati.

Amazon S3 effettua backup dei dati per evitare la perdita e mantiene una copia delle versioni dei dati per permettere il loro recupero. Abbiamo diversi piani:

- S3 Standard: si accede abbastanza frequentemente ai dati.
- S3 Infrequent Access: si accede poco frequentemente.
- Amazon Glacier: non si accede quasi mai, per esempio backup di database.

Per quanto riguarda la sicurezza offre una connessione SSL per la trasmissione dei dati e quindi per evitare che utenti malevoli catturino le nostre informazioni.

2.1.3 Amazon Elastic Block Store

Gli **Amazon Elastic Block Store** sono volumi di storage a blocchi persistenti da utilizzare con le istanze Amazon EC2 nel cloud AWS. Ogni volume Amazon EBS viene replicato automaticamente all'interno della sua zona di disponibilità per proteggerlo dal guasto dei componenti, offrendo alta disponibilità e durata.

Progettato per carichi di lavoro delle applicazioni che traggono vantaggio dalla messa a punto di prestazioni, costi e capacità (ad esempio, analisi dei big data, elaborazione di flussi, data warehousing).

2.1.4 Dropbox

È un servizio di file hosting che offre spazio di archiviazione gratuito, sincronizzazione dei file e strumenti di collaborazione (Dropbox Paper) a tutti. Inizialmente si trattava di un servizio che si interponeva tra Amazon e l'utente: Dropbox si occupava di prendere i file caricati dagli utenti e li inseriva in un bucket di S3.

L'esodo di Dropbox

Dropbox è stato fondato nel 2007 da due studenti del MIT, avevano pensato di offrire gratuitamente spazio di archiviazione con limitazioni, e chi voleva poteva acquistare l'account premium per avere più capacità di memorizzazione. A Marzo 2016 Dropbox aveva 500 milioni di utenti.

Per i primi otto anni della sua vita, Dropbox ha archiviato miliardi e miliardi di file su Amazons S3 (file su S3, metadati sulle proprie macchine).

Tra il 2014 e il 2016, Dropbox ha costruito la propria vasta rete di computer e ha spostato il proprio servizio su una nuova generazione di macchine progettate dai propri ingegneri. La migrazione è stata una sfida molto grande dal punto di vista

- **tecnico**: è stato progettato dell'hardware ad-hoc ("Diskotech") per memorizzare quantità grande di dati con un nuovo codice chiamato "Magic Pocket";
- **logistico**: è stato necessario trasferire molti dati da Amazon S3 alla nuova infrastruttura. Inoltre andavano costruiti i datacenter;
- **contrattuale**: le deadline dei contratti dovevano essere rispettate per non rinnovare il contratto.

Vantaggi:

- Autonomia (non dipendere da altro fornitore)
- Costi meno elevati per fornire il servizio (a regime)
- Possibilità offrire in modo più flessibile ed efficiente il servizio
- Supporto tecnico più veloce (in casa)

Svantaggi:

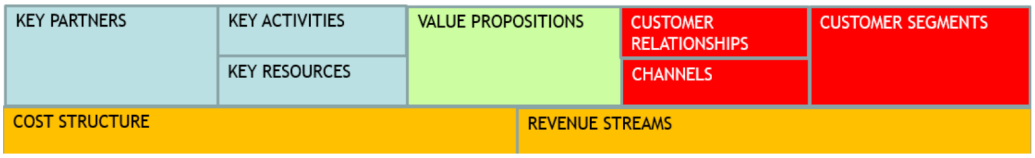
- Costi elevati sia per realizzazione DC e migrazione
- Rischio elevato per competenze da acquisire
- Rischio elevato per perdita utenti durante migrazione o anche dopo

Capitolo 3

Business Model

Cos'è un Business Model?

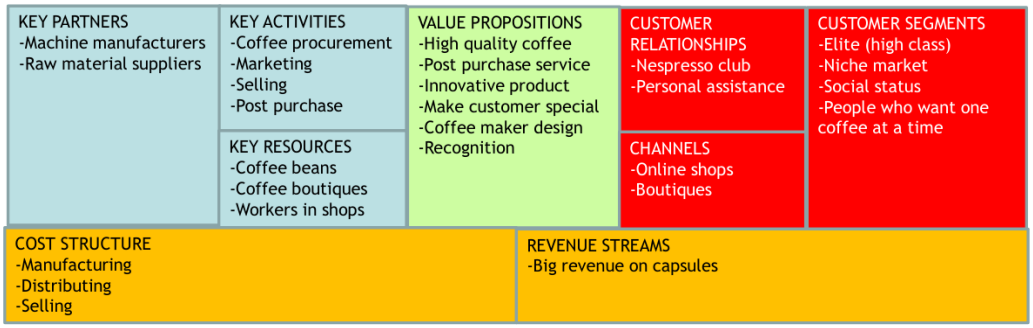
Descrive come un'azienda crea, consegna e cattura valore. Ma come si fa a dire queste cose di un'azienda? Attraverso i **Business Model Canvas**, possiamo utilizzarli in molti modi, possiamo prendere un canvas: vuoto e un'azienda esistente e provare e riempirlo oppure costruirne uno per una startup.



- **Customer Segment**: Il target di mercato, quali tipi di clienti vogliamo attirare.
- **Value Proposition**: Qual è il valore del nostro servizio? Cosa offriamo al cliente? Per cosa dovrebbero venire da noi?
- **Channels**: I canali con cui il servizio arriva ai clienti (shop online, negozio fisico...)
- **Customer Relationship**: Sembra opzionale ma in realtà il rapporto col cliente è molto importante, ci sono aziende che si focalizzano molto su questo aspetto sfruttando la "fidelizzazione" attraverso dei punti per il cliente.
- **Revenue Streams**: I flussi da cui arrivano le entrate, ne descrive anche il tipo.

- **Key Resources**: Risorse chiave, le risorse che eroghiamo e che sono fondamentali.
- **Key Activites**: L'attività principale della nostra azienda.
- **Key Partnership**: È bene cercare dei partner, come Dropbox aveva Amazon, però se il partner fallisce rischia di cadere tutto.
- **Cost structures**: Indica quali sono le uscite.

Esempio di Business Model: Nespresso



Nel 1976 Nestlé dominava il market con Nescafè il caffè solubile più venduto. Le macchinette Nespresso furono ideate nel 1976 quando nessun altro le faceva, erano pronte ma non riuscivano ad entrare nel mercato.

Nel 1988 il nuovo CEO cambiò il business model, cambiò il **Customer Segment** poiché le macchinette non dovevano avere lo stesso mercato del Nescafé ma dovevano attrarre impiegati di alto livello e in generale famiglie benestanti.

- Pubblicità: l'idea è che sei fig* se bevi Nescafé (vedi George Clooney).
- Canali di acquisto: online shop, Club Nespresso, oppure vai nelle boutique solo di capsule della Nespresso, solitamente nel centro della città vicino a Armani, Gucci, ...
- Club Nespresso: una delle cose più importanti per un'azienda è conoscere il comportamento dei propri clienti per riuscire a mantenerli. Analisi dei dati, Nespresso traccia il 100% delle attività dei clienti poiché ci sono solo due canali. Ogni volta che fai l'ordine col club nespresso vedono tutto quello che hai comprato in modo da poterti offrire tramite newsletter e news i prodotti più adatti.

Dal punto di vista della sostenibilità ambientale Nespresso produce una grandissima quantità di materiale difficilmente riciclabile che attualmente viene smaltito male. Come la commercializzazione del latte in polvere, grande successo di Nestlé, ma ha avuto effetti secondari: nei paesi in via di sviluppo l'acqua con cui veniva sciolto il latte era contaminata e molti bambini sono morti. Hanno conquistato comunque una fetta di mercato importante.

Esempio di Business Model: Zara

Zara ha deciso di produrre abiti in base a cosa la gente acquista, non è il cliente a seguire la moda ma è Zara a seguire il cliente. Rinnova continuamente. **No warehousing: no magazzino, viene spedito quello che serve ad ogni negozio**.

Esempio di Business Model: TicketResturant

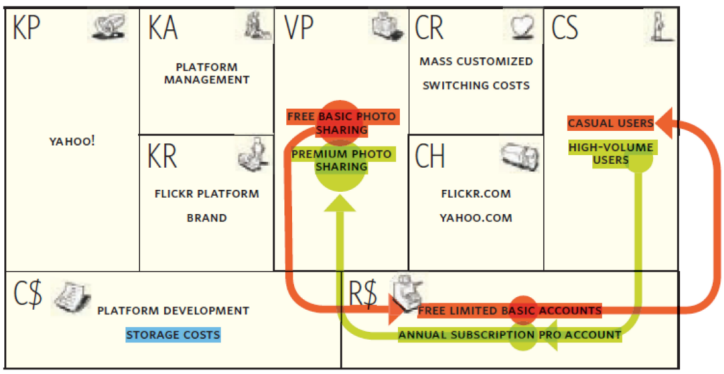
Idea Le piccole imprese acquistano buoni e li distribuiscono ai dipendenti; i dipendenti riscattano i voucher presso i ristoranti locali

Guadagna da entrambe le parte: il datore di lavoro paga un premio sull'importo del buono, mentre il commerciante incassa con uno sconto sull'importo del buono. Rottura (buoni emessi ma non utilizzati). Reddito fluttuante (intervallo tra l'acquisto del voucher e il rimborso del voucher).

3.1 Freemium come Business Model

Consiste in una suddivisione dei servizi: una parte **Free** che comprende cosiddetti i servizi "di base" e una parte **Premium** a pagamento che invece offre servizi aggiuntivi e/o migliorati. Chi adotta il modello Freemium deve porre molta attenzione al costo che comportano gli utenti "free" e alla percentuale di conversione degli utenti che decidono di passare a Premium. I prossimi esempi metteranno in luce quali sono i modi in cui guadagnano le aziende che seguono questo modello.

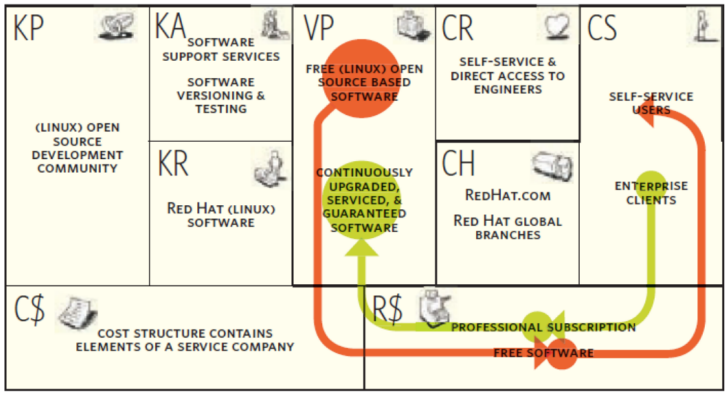
Flickr



Se paghi puoi avere premium photo sharing: oltre a poter organizzare foto in cartelle, si ottengono dei servizi addizionali.

È durato 3 anni, dal 2002 al 2005, acquistato da Yahoo per 25 milioni di dollari che a sua volta è stato acquistato da Verizon per 4 miliardi nel 2017 dopo aver rifiutato 44 miliardi da Microsoft nel 2009.

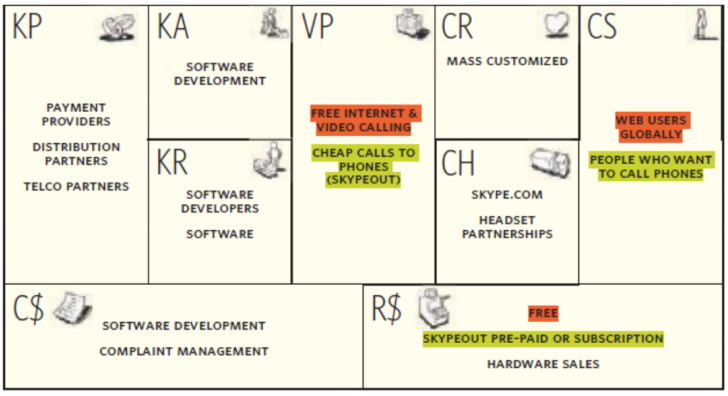
RedHat



Chi usava un open source e offriva un servizio aveva paura che l'open source fallisse, RedHat si metteva nel mezzo e garantiva stabilità, veniva pagato e manteneva l'open source sempre aggiornato: open source non significa che il software sia gratuito ma che è possibile modificarne il sorgente.

Venduto per 34 miliardi di dollari a IBM.

Skype



Tutti gli utenti web possono fare chiamate gratuite e anche videochiamate. Pagando si ha la possibilità di chiamare anche i cellulari a prezzi competitivi.

Nato nel 2003, comprato da Ebay nel 2005 e poi nel 2011 da Microsoft.

Dropbox

Dropbox ha 500 milioni di utenti di cui 12 milioni paganti.

Privati	
Free:	2 GB
Plus:	1 TB per €9,99 al mese
Professional:	2 TB per €19,99 al mese
Aziende	
Standard:	3TB per €10,00 al mese
Adavanced:	no limit per €15,00 al mese
Enterprise:	“contact us”

Neflix

Netflix usa attualmente il **15% del traffico internet** in download nel mondo. Paga i diritti dei film e guadagna per il “noleggio” verso gli utenti. Come già detto, Dropbox ha iniziato appoggiandosi ad Amazon AWS e poi si è distaccato; Netflix invece ha iniziato la partnership con Amazon, nonostante avesse costruito già i data center. L’arrivo dei Container ha creato dei problemi al reparto di sviluppo di Netflix e alla fine hanno dovuto mettere da parte l’idea di usare i propri data center chiudendoli e continuando ad appoggiarsi ad Amazon.

3.2 Business Model Generation

Esistono diversi tipi di strategie per i Business Model. Al giorno d'oggi la cosa più importante è mettere al centro il cliente e il rapporto che si crea con esso: bisogna chiedersi di che cosa ha bisogno, come preferisce essere contattato, a quale tipo di valore sono interessati i clienti e per cosa sono diposti a pagare. Per innovare il business model si può...

- Partire dalle risorse che ha già l'azienda (**Resource Driven**), come ad esempio **Amazon Fulfillment** che consiste nell'offrire a terze parti la possibilità di vendere e far arrivare i propri beni ai clienti. Prima di iniziare con questa pratica Amazon aveva già tutto pronto (shop online, catena organizzativa).

- Basarsi sull'offerta (**Offer Driven**), ad esempio: **Cemex** che in un momento in cui il cemento era garantito in 48h di tempo è riuscita a farlo in 4h.

- Basarsi sui clienti (Customer Driven): **23andMe** offriva test per dna ai singoli clienti privati.

- Basarsi sull'aspetto finanziario (**Finance Driven**), per esempio **Xerox** che introdusse un modello in cui il costo delle fotocopie veniva offerto in leasing e venivano inoltre garantite 2000 copie gratuite.

- Epicentri multipli (Multiple Epicenter Driven): sono tutti modi in cui si può ridefinire un Business Model innovativo partendo da angolature distinte. Le idee delle aziende che hanno fatto più successo (Google, Spotify, Dropbox, ...) si sono basate su domande del tipo "Cosa succederebbe se offrissi musica gratis a tutti?" e sono riuscite a trovare una strategia vincente.

3.2.1 Statistiche startup

Airbnb: valore di 35 miliardi di dollari. 2 Milioni di persone a notte dormono in una struttura prenotata su Airbnb.
Epic Games: Fortnite ha 250 Milioni di giocatori e guadagna centinaia di milioni di dollari al mese grazie alla vendita delle skin.
Facebook: 2.45 miliardi di persone ogni mese.
Instagram: acquistato da Facebook dopo 2 anni per un miliardo di dollari.

Il 90% delle startup fallisce per 4 motivi principali: incompetenza, incapacità di gestire il budget, mancanza di esperienza, problemi personali.

Spotify

Il nome è nato da un misunderstanding. Gli stakeholder di spotify sono:

- Freeusers
- Royalty holder: quelli che vengono pagati ogni volta che viene riprodotto un brano (SIAE in Italia).
- Subscriber: utenti premium.
- Advertiser: coloro che pagano Spotify per venire sponsorizzati.

Gli utenti free e gli utenti abbonati devono bilanciarsi, ma quando iniziamo con un modello Freemium non abbiamo idea di quale sarà la percentuale tra utenti free e premium, è difficile da stabilire.

Spotify per pagare meno di royalty ha ideato una strategia che permette di scegliere dove reperire il brano in base alla localizzazione dell'utente poiché i costi variano in base al paese da dove l'utente richiede l'ascolto.

Se gli utenti Premium ascoltano troppo rischiamo di andare in perdita poiché è più quello che paghiamo per le royalties che quello che riceviamo dagli utenti.

Statistiche:
217 milioni di utenti attivi 100 milioni di questi sono membri Premium

3.2.2 Casi di studio

Le entrate di Amazon (Amazon revenue)

Le entrate di Amazon nel 2019 sono state pari a **280 miliardi di dollari**.

1. Store Online
2. Venditore per altri (Amazon Fulfillment)
3. Amazon Web Services (Cloud)

Le entrate di Google

Google è un servizio gratuito, come fa a guadagnare? Grazie al Customized Advertising, la nostra "attenzione" verso le pubblicità permette a Google di farsi pagare.
Cinque Riflessioni sui motori di ricerca

- Per la prima volta nella storia tutti posiamo facilmente generare informazioni accessibili a tutti. Ci sono più di **4 miliardi di utenti su Internet e quasi 2 miliardi di siti Web**. Quasi tutte le ricerche di informazioni passano attraverso un unico punto di accesso: Google. Azienda che domina il mondo dei motori di ricerca con più di **3.5 miliardi di ricerche** giornaliere. Durante una ricerca **i primi 3 risultati ottengono il 75% dei click** e solitamente se non troviamo niente nei primi 3 risultati cambiamo query di ricerca. I siti Web cercano di comparire nei primi 3 risultati perché altrimenti non vengono cliccati. Dobbiamo sempre tenere a mente che la prima pagina dei risultati di Google è una parte molto piccola di tutte le informazioni esistenti.

- **Google non controlla le fonti, né se un'informazione è vera o no**. Se facciamo una ricerca oggi nei primi 3 risultati otteniamo determinati siti web, se riprovassimo a cercare la stessa cosa dopo un mese potremmo avere dei risultati diversi.

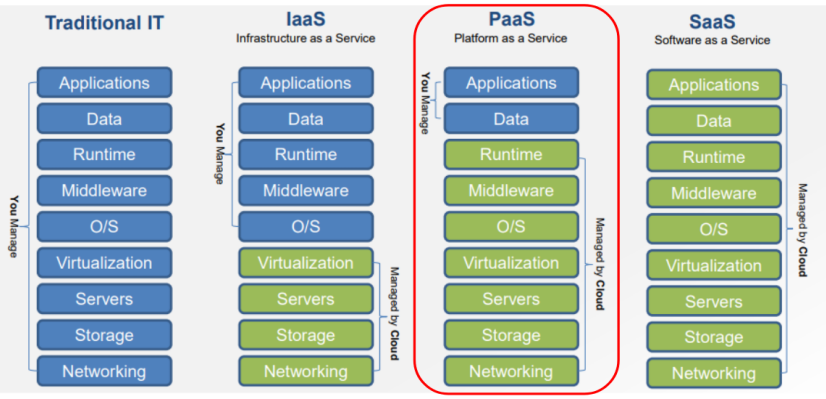
- Google può **tenere traccia** delle nostre attività poiché possiede molti servizi diversi: Youtube, Google Maps, Google Chrome, Google Shopping, Google Calendar, Waze, Google Photos, ...

- Cosa succederebbe se Google manipolasse i risultati delle ricerche? È stato fatto un esperimento ed è stato osservato che è possibile spostare le preferenze di voto degli elettori indecisi del 20% mostrando degli opportuni risultati di ricerca (in modo trasparente) agli utenti.

- Quali sono gli effetti di Google per la nostra memoria? Se sappiamo che possiamo avere accesso a certe informazioni in qualsiasi momento non le ricordiamo, ma ci limitiamo a ricordare dove possiamo cercarle o trovarle. Questa cosa si chiama *memoria transattiva* ed è sempre esistita: sapere che ci sono cose che non si ricordano, ma sapere che altre persone invece le sanno. **Internet diventa una *memoria transattiva***.

Capitolo 4

Platform as a Service (PaaS)



I PaaS aumentano ancora di più i servizi offerti dallo IaaS offrendoci un intero ambiente di sviluppo e gestione dell'applicazione. Vantaggi principali di PaaS per DevOps: non è necessario concentrarsi su provisioning, gestione o monitoraggio di elaborazione, archiviazione, rete e software.

- Può creare prototipi in **pochi minuti**.
- Può creare nuove versioni o deploy del codice più **rapidamente**.
- **Self-assemble service**, quindi le varie parti dell'applicazione si assemblano da sole.
- **Scala automaticamente**.
- Non bisogna preoccuparci della sicurezza.
- Il PaaS si occupa delle strategia di Backup e Recovery.

Il PaaS guadagna meno rispetto a IaaS, al secondo posto per guadagno, e SaaS, al primo posto, ma vale comunque molto: circa 19 miliardi di dollari.

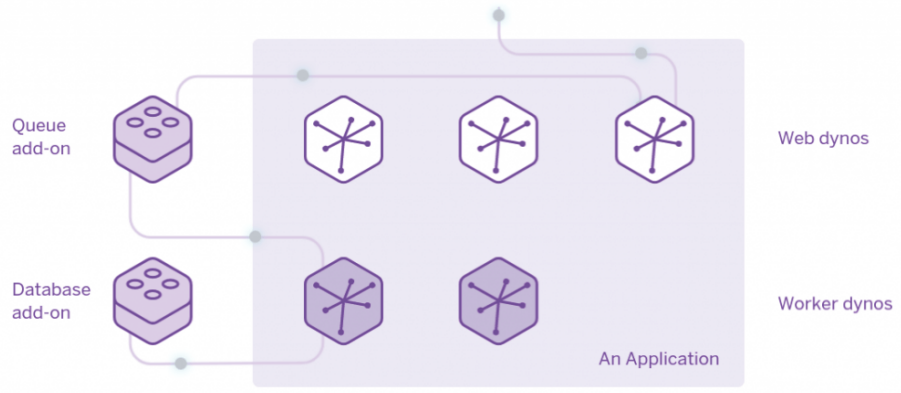
4.1 Heroku (Paas Freemium)

È una piattaforma cloud che fornisce una serie di servizi integrati e un intero sistema che ci permette **non solo di fare il deployment delle applicazioni ma anche di completarle, mandarle in esecuzione e gestirle**. Una delle caratteristiche di Heroku è di essere basata su un sistema di **Container**.

È nato nel 2007, poi nel 2010 è cresciuto ed è stato acquistato da Salesforce (che si occupa di Saas soprattutto). Comprende diversi linguaggi per lo sviluppo del codice (Node, Java, Php, Python, Go, Scala).

I *Container* di Heroku si chiamano **Heroku Dynos**. Gli Heroku Dynos sono una virtualizzazione, ambienti Linux abbastanza leggeri, isolati che permettono di "incapsulare" un'applicazione e tutte le sue dipendenze, librerie... in un grande "container". Questo ci permette di spostare la nostra applicazione senza dover dividere le varie parti: una volta giunto a destinazione viene prelevata l'immagine del container e ricostruita l'applicazione. I Dynos sono utilizzati anche per lo scaling: potrebbe volerci un solo Dynos inizialmente, ma poi l'applicazione ingrandendosi avrà bisogno di più spazio e vengono semplicemente aggiunti più container la cui gestione è molto semplice; non dobbiamo più preoccuparci dell'aspetto scalabilità.

4.1.1 Web Dynos



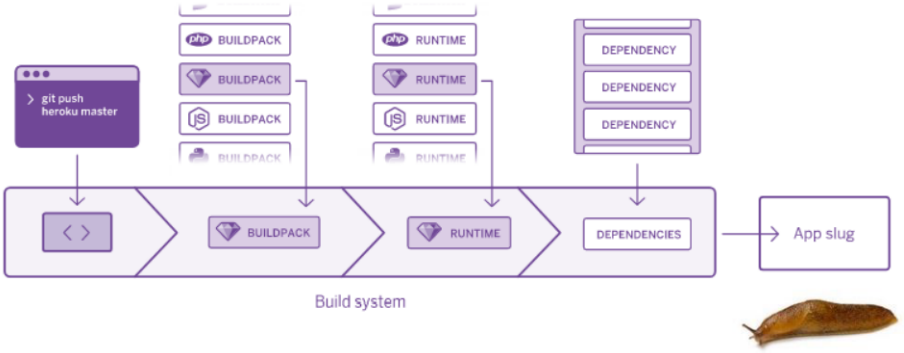
L'applicazione riceve la richiesta, la richiesta viene inviata a uno degli **Web Dynos**, viene analizzata e messa in una coda asincrona (che è ottima per la scalabilità orizzontale), poi il **Worker Dyno** prende la richiesta e la gestisce.

Se si vuole si può far persistere i risultati **in un database che è sempre parte di Heroku**. In questo caso **la scalabilità permette di aumentare il numero di Web Dynos** in modo da poter gestire un elevato numero di richieste ricevute contemporaneamente.

Buildtime

Heroku ha bisogno di tre cose per costruire un'applicazione:

- Codice sorgente
- Lista di dipendenze (Quali sono le cose di cui ha bisogno per funzionare)
- Un '`Procfile`' cioè un file di testo che indica qual è il comando per far partire il codice



Queste tre cose fanno parte del Buildtime, cioè la fase di costruzione; una volta date in input parte il **sistema automatico di built**: dopo aver ricevuto il codice scarica il **Build Packet** (linguaggio, dipendenze, librerie...), produce uno **Slug** e lo mette in esecuzione in un Dyno. Il componente finale per eseguire l'applicazione è il sistema operativo (Ubuntu) che è aggiunto da Heroku e che si chiama "**stack**".

Runtime

Una volta preparato tutto Heroku crea uno o più Dynos con lo stack e lo slug e fa partire l'applicazione, a quel punto ci abilita a modificare l'app con i vari tipi di Dyno (free, standard, performance, web, workers).

4.1.2 Add-ons

Heroku ha più di **150 Add-ons** (servizi esterni) da aggiungere alla propria applicazione per estenderne le funzionalità come servizio di autenticazione, log, monitorning, data stores... Tutto questo attrae l'utente perché offre la possibilità di integrare nuove funzionalità in un tempo molto breve e ne facilita lo sviluppo non dovendo costruire da zero ciò che è richiesto.

L'aspetto negativo è che questo lega l'utente all'utilizzo della piattaforma e instaura una forma di **vendor lock-in** rendendo l'applicazione **difficilmente portabile**. Infatti se volessimo spostare la nostra applicazione su un nuovo servizio cloud saremo costretti a rivedere tutto il codice perché gli Add-ons non funzionerebbero.

4.2 Microsoft Azure (PaaS e anche IaaS)

Possiede caratteristiche e capacità che consentono a un'organizzazione di creare o distribuire applicazioni in modo rapido e semplice senza doversi preoccupare dell'OS e dell'infrastruttura sottostanti:

- Scalabilità **verticale**
- Supporta **diversi linguaggi** e ha un'ampia varietà di strumenti per sviluppatori
- SQL database
- Machine Learning per analisi predittive
- Servizi media per supportare video live e on demand
- Meccanismi di sicurezza
- Data storage (IaaS: ai servizi PaaS è spesso richiesto di offrire IaaS)

4.3 Open Shift (PaaS)

Il video mostra come si possono "assemblare" varie parti per costruire una piccola applicazione che simuli un sistema di voto. Il video commerciale sembra molto più semplice di questo, in realtà è abbastanza macchinoso collegare i vari pezzi di funzionalità e linguaggi diversi tra loro.

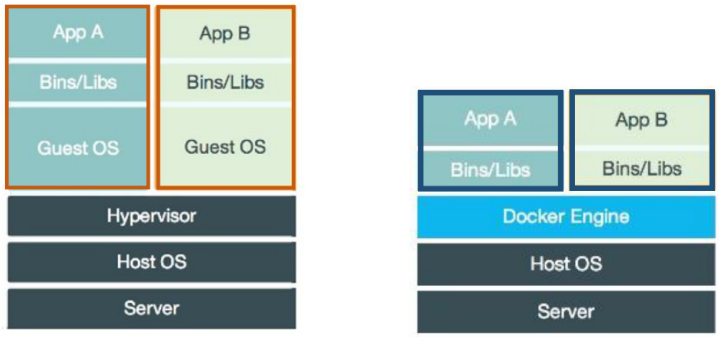
4.4 Quale PaaS usare? Find your PaaS

In un mercato così variegato è difficile fare la scelta giusta, fortunatamente ci sono dei siti a disposizione che ci permettono di trovare il PaaS adatto alle nostre esigenze semplicemente facendoci scegliere le funzionalità di cui abbiamo bisogno.

4.5 Container

Docker è una piattaforma che ci permette di mandare in esecuzione delle applicazioni software in un ambiente isolato tramite il meccanismo di virtualizzazione dei *containers*.

Nelle macchine virtuali varie applicazioni possono essere eseguite sullo stesso server e ogni VM richiede l'allocazione di risorse (CPU, memoria e storage) e l'installazione del sistema operativo ospite. I containers sfruttano la possibilità offerta dal kernel del sistema operativo di eseguire più istanze isolate.



Al posto dell'Hypervisor viene montato il Docker Engine. In confronto alle VM, i containers sono più leggeri (richiedono meno risorse), più veloci ad avviarsi e più semplici da "buildare"; tuttavia sono meno sicuri poiché meno isolati.

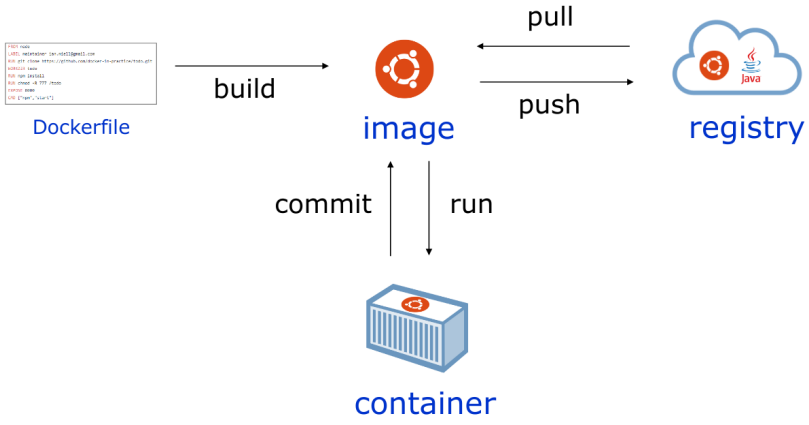
4.5.1 Docker

Docker sfrutta la virtualizzazione basata su containers per eseguire più istanze guest isolate sullo (stesso) sistema operativo. I componenti software sono confezionati in ***immagini***, che vengono sfruttate come modelli di sola lettura per creare ed eseguire ***containers***; inoltre, è possibile montare ***volumi*** esterni per garantire la persistenza dei dati.

"Build, ship and run any app, anywhere."

Docker images

Sono template in sola lettura usati per creare i containers che vengono memorizzati in un Docker **registry** (pubblico o privato). Ogni registry è strutturato in repositories e ogni repository contiene un insieme di immagini per versioni diverse di un software. Queste immagini sono strutturate in livelli, ogni livello è a sua volta un'immagine e il livello più basso si chiama *base image*.

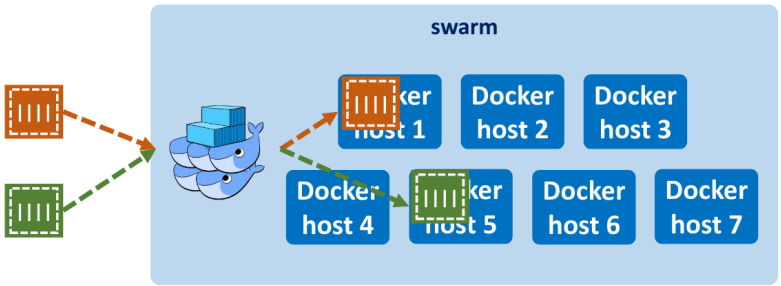


Docker compose

Docker compose permette sfruttare più container per costruire applicazioni che consistono di più servizi.

4.5.2 Swarm mode

Docker include la ***swarm mode*** per la gestione di un cluster di host Docker chiamato *swarm*.



I nodi Swarm possono agire come manager, delegando i compiti ai worker, o come worker, eseguendo i compiti loro assegnati. È possibile definire lo stato

desiderato dei vari servizi nello stack dell'applicazione, incluso il numero di attività da eseguire in ogni servizio. I nodi manager assegnano a ogni servizio nello swarm un nome DNS univoco e bilanciano il carico dei container in esecuzione. I nodi manager monitorano costantemente lo stato del cluster e riconciliano eventuali differenze tra lo stato effettivo e lo stato desiderato espresso: ad esempio, se si configura un servizio per eseguire 10 repliche di un contenitore e una macchina worker che ospita due di queste repliche si arresta in modo anomalo, vengono create due nuove repliche e assegnate ai worker in esecuzione e disponibili.

4.5.3 Kubernetes

Kubernetes è un sistema di orchestrazione per i containers Docker, più esteso di Docker Swarm e che ha lo scopo di coordinare i cluster di nodi su larga scala nella produzione in modo efficiente.

Un *pod* è un gruppo di uno o più containers (ad esempio, contenitori Docker), con archiviazione/rete condivisa e una specifica su come eseguire i containers. I contenuti di un pod sono sempre co-localizzati e co-pianificati ed eseguiti in un contesto condiviso. Un pod modella un "host logico" specifico dell'applicazione: contiene uno o più containers dell'applicazione che sono collegati in modo relativamente stretto.

Il kubelet è l'"agente nodo" primario che viene eseguito su ogni nodo.

- Il kubelet funziona in termini di PodSpec (un oggetto `YAML` o `JSON` che descrive un pod).
- Il kubelet accetta una serie di PodSpec forniti tramite vari meccanismi (principalmente tramite il server API) e garantisce che i contenitori descritti in quei PodSpec siano in esecuzione e integri.
- Il kubelet non gestisce i contenitori che non sono stati creati da Kubernetes.

Docker Swarm	Kubernetes
Simpler to install Softer learning curve	Features auto-scaling Higher fault tolerance Huge community Backed by Cloud Native Computing Foundation (CNCF)
Preferred in enviroments where simplicity and fast development is favored	Preferred for enviroments where medium to large clusters are running complex applications

Capitolo 5

FaaS

Il meccanismo di "*Function as a Service*", o **FaaS**, conosciuto anche come "*serverless*[1] computing", ha avuto molto successo nel mondo dell'ICT e dello sviluppo del software e molte altre compagnie hanno iniziato a offrirlo come servizio.

"More than 20% of global enterprises will have deployed serverless computing technologies by 2020." [`Gartner, Dec 2018`]

"The most prominent manifestation of serverless computing is function platform ad a service, or fPaaS. Gartner predicts that half of global enterprises will have deployed fPaaS by 2025, up from 20% today." [`Gartner, Dec 2020`]

5.1 AWS Lambda

Permette di **eseguire codice** senza eseguire il provisioning o gestire i server, il tutto **senza amministrazione**. Una volta caricato il codice, **Lambda si occuperà di eseguirlo e scalarlo** con disponibilità elevata. È possibile configurare il codice in modo che **si attivi automaticamente da altri servizi AWS** o chiamarlo direttamente da qualsiasi app Web o mobile. **Si paga solo per il tempo di elaborazione** che si consuma.
Facile da usare:

- carica la tua funzione/progettala in AWS IDE/selezionala dall'elenco di esempi predefiniti;
- seleziona l'origine dell'evento da monitorare (ad es. Bucket S3).

[1] **Serverless** indica il fatto che la gestione del server è trasparente a chi ne usufruisce.

Poi Lambda attiva automaticamente la funzione quando si verifica un evento; gestisce tutte le capacità, la scalabilità, l'applicazione di patch e l'amministrazione dell'infrastruttura per eseguire il codice e pubblica metriche e registri in tempo reale. Tutto ciò con un servizio low cost (canone basso per richiesta).

Prezzo

- Richieste: $0,20 per 1 milione di richieste.
- Durata (dipende dalla quantità di memoria allocata alla funzione): $0,0000166667 per ogni GB/sec.

Se hai assegnato 512 MB di memoria alla tua funzione, l'hai eseguita 3 milioni di volte in un mese e ha funzionato per 1 secondo ogni volta, i tuoi addebiti saranno:

$$3.000.000/1.000.000 \times \$0,20 = \$0,60$$

$$+$$

$$(3.000.000 \times 1) \times (512/1024) \times \$0,0000166667 = \$25,00$$

Livello di utilizzo gratuito

- 1 milione di richieste gratuite al mese.
- 400.000 GB/sec. di tempo di elaborazione al mese.

Se hai assegnato 512 MB di memoria alla tua funzione, l'hai eseguita 3 milioni di volte in un mese e ha funzionato per 1 secondo ogni volta, e hai avuto l'utilizzo gratuito per sfruttare le tue spese saranno:

$$(3.000.000 - 1.000.000)/1.000.000 \times \$0,20 = \$0,40$$

$$+$$

$$((3.000.000 \times 1) \times (512/1024) - 400.000) \times \$0,0000166667 = \$18,33$$

Perché il nome "lambda"?

AWS Lambda prende il suo nome dal *lambda calcolo*:

$$\lambda x.\ f(x)$$

f è il supporto a tempo di esecuzione offerto da AWS, mentre x è la funzione caricata su AWS.

5.2 Confronto fra 10 piattaforme FaaS

Commercial	Open source
AWS Lambda	Apache Openwhisk
Google Cloud Functions	Fission
MS Azure Functions	Fn
	Knative
	Kubeless
	Nuclio
	OpenFaaS

5.2.1 Business view

Licensing

Tutte le piattaforme open source usano una licenza permissiva: quella più usata è Apache 2.0.

Tutte le piattaforme FaaS commerciali usano licenze proprietarie, con MS Azure Functions che rilascia anche alcune delle sue componenti come progetti open source.

Installazione

Tra le piattaforme commerciali, solo Azure Functions ha alcune parti installabili on-premises.

Le piattaforme FaaS open source sono tutte installabili e permettono di avere una certa portabilità nell'installazione (supportano più target): Kubernetes è la più supportata.

Codice sorgente

Tutte le piattaforme open source sono ospitate su GitHub e la maggior parte sono implementate in Go.

Azure Functions è l'unica piattaforma commerciale che è parzialmente open source.

Interface

Tutte le piattaforme supportano CLI (*Command Line Interface*), mentre non sempre sono disponibili le API e le GUI. Per quanto riguarda l'amministrazione, quindi dal punto di vista delle operazioni che possiamo eseguire sulla

piattaforma per amministrarla (deployment, configurazione, terminazione), l'offerta varia considerevolmente.

Community

OpenFaaS, Apache Openwhisk e Knative hanno rispettivamente i rating più alti su GitHub in termini di stars, contributori e commits. Le domande su Stackoverflow mostrano una notevole differenza in interessi tra piattaforme commerciali e quelle open source.

Documentazione

Tutte le piattaforme forniscono la documentazione necessaria a eseguire il deployment di un'applicazione e usare la piattaforma stessa; tuttavia non sempre vengono documentati lo sviluppo e l'architettura della piattaforma.

5.2.2 Technical view

Development

Java, Node.js e Python sono i runtime più supportati, con Docker che sta diventando piuttosto popolare per aiutare a personalizzare i runtime. La possibilità di usare IDEs e editor di testo è principalmente offerta dalle piattaforme commerciali.

Versioning

La maggior parte delle piattaforme open source, a differenza delle piattaforme commerciali, implementano l'implicit versioning senza offrire meccanismi dedicati.

Event sources

Tutte le piattaforme supportano l'invocazione sincrona di una funzione basata su HTTP, mentre le invocazioni asincrone sono supportate da poche piattaforme. Più di metà delle piattaforme open source non supportano i database come sorgenti di eventi. Sono ampiamente supportati meccanismi di scheduling per processare flussi di dati e messaggi. Soltanto metà delle piattaforme permette di avere sorgenti di eventi personalizzate.

Orchestrazione delle funzioni

Più della metà delle piattaforme FaaS supporta l'orchestrazione di funzioni: alcune offrono dei DSL (*Domanin Specific Language*) *ad-hoc*, altre dei workflow.

Testing & debugging

Le piattaforme commerciali offrono delle opzioni più avanzate: funzioni di testing e debugging integrate all'interno dell'ambiente di sviluppo.

Monitoraggio

Le piattaforme commerciali offrono strumenti dedicati per eseguire monitoraggio e logging delle funzioni, mentre quelle open source si affidano all'integrazione di strumenti di terze parti.

Application delivery

La maggior parte delle piattaforme recensite segue un approccio dichiarativo per automatizzare la distribuzione delle applicazioni. Le piattaforme commerciali supportano nativamente CI/CD attraverso strumenti dedicati, mentre le piattaforme open source (a parte OpenFaasS) non documentano l'integrazione con le pipeline CI/CD.

Code reuse

Solo AWS Lambda e MS Azure Functions offrono un marketplace delle funzioni integrato.

Access management

Le piattaforme commerciali supportano nativamente autenticazione e controllo di accesso alle risorse.

Le piattaforme open source si affidano, tipicamente, alle features offerte dall'ambiente che ospita il FaaS.

Capitolo 6

Microservizi

Due motivazioni principali per l'adozione di microservizi.

1. Cercare di abbreviare il cosiddetto ***lead time*** (tempo che passa da quando si comincia a immaginare una funzionalità al momento in cui il primo utente "fa il primo click" su tale funzionalità) per nuove features o aggiornamenti a un'applicazione:
   - accelerare *rebuild* e *redeployment*;
   - ridurre le cosiddette *corde* tra silos funzionali.
2. Riuscire a **scalare** efficacemente un'applicazione con un numero di utenti enorme, anche nell'ordine delle centinaia di milioni.

Ma che cosa sono i microservizi?

In passato una tipica applicazione enterprise consisteva in un'interfaccia utente (lato client), l'applicazione vera e propria (lato server) e il database. L'applicazione lato server, tipicamente chiamata ***monolite***, si occupava di gestire le richieste HTTP, di implementare la logica di dominio, di interrogare e aggiornare il database (se necessario) e, infine, di mandare le risposte alle richieste del client.

Un'architettura monolitica comporta che anche un solo piccolo cambiamento di una parte richieda il **rebuild e il redeployment dell'intera applicazione**: tutto ciò potrebbe risultare costoso, non solo in termini di risorse, ma anche in termini di tempo necessario. Inoltre, scalare un monolite richiede di **scalare l'intera applicazione** piuttosto che scalare quella singola parte che necessita di più risorse.

Essenza dei microservizi

Le applicazioni sono sviluppate come insieme di servizi, si tratta di un'architettura modulare. Ognuno di questi servizi viene eseguito in un ambiente (processo) separato, tipicamente in un container; la comunicazione fra tali processi avviene con meccanismi semplici e leggeri, solitamente si usano protocolli di tipo REST. Un'altra delle caratteristiche distintive è il ***poliglottismo***: servizi diversi possono essere scritti in linguaggi diversi; così come tipi di dati diversi possono essere memorizzati con tecnologie diverse.

Si chiamano microservizi ma il termine "*micro*" esiste per motivi storici: la dimensione non è veramente il punto più importante. Se si vuole sapere davvero la dimensione di un microservizio si può misurare la dimensione del team che lo gestisce.

Uno degli aspetti più innovativi è l'idea di organizzare i servizi intorno alle ***business capabilities*** e quindi avere dei **cross-functional teams**: team indipendenti (per quanto possibile) l'uno dall'altro e conseguentemente funzionalità di un'applicazione distribuite su questi microservizi.

"*Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations*"
M. Conway, 1968

Un'ulteriore novità dei microservizi è la **decentralizzazione della gestione dei dati**:

- lasciare che ogni servizio gestisca il proprio database;
- "*eventual consistency and compensations*" invece di transazioni distribuite;

Inoltre, alla base dei microservizi, vi è la ***deployability indipendente***, ovvero possibilità di eseguire il deployment di una funzionalità mentre l'intero applicativo è in esecuzione, e la ***scalabilità orizzontale***, cioè la possibilità di scalare un singolo servizio.

Per concludere si può parlare di ***fault resilient services***: progettare le applicazioni in modo tale che riescano tollerare i fallimenti e quindi riuscire a garantire una *resilienza* del sistema quanta più alta possibile di fronte a fault che certamente avverranno all'interno del sistema.

Test (bravely) *Chaos Monkey* è uno strumento che permette di testare in modo coraggioso il funzionamento di un'applicazione. Perché in modo "*coraggioso*"? È possibile configurarlo in modo tale da uccidere istanze di mac-

chine virtuali, o direttamente container, che vengono eseguiti nell'ambiente di **produzione**.

Dev-Ops

Per anni lo sviluppo del software si è basato su un gruppo di *developers* e un gruppo di *operators*, le persone che gestiscono i rapporti con i clienti. In **Dev-Ops** questa distinzione cessa di esistere.

"*You build it, you run it*"

Strumenti a disposizione:

- Version Control System (VCS), e.g. Git & Github;
- Continuous Integration & Continuos Development (CI&CD), e.g. Jerkins;
- Infrastructure as Code (IaC), e.g. Puppet, Chef;
- Application Performance Monitoring (APM), e.g. NewRelic.

Osservazioni conclusive

Don't even consider microservices unless you have a system that's too complex to manage as a monolith. [M. Fowler]

Lo stile architetturale basato sui microservizi è un'idea molto importante da tenere in considerazione per le applicazioni enterprise. Ci sono molti vantaggi, come abbreviare il **lead time** per aggiornare le applicazioni e riuscire a **scalare in maniera efficace** davanti a numeri drammatici di richieste.

Tra gli svantaggi principali vi sono l'**overhead** dovuto alla comunicazione e la **complessità** del sistema finale; è facile "sbagliare i tagli" nella progettazione dell'architettura, quando si eliminano le dipendenze durante il partizionamento (se ce ne sono troppe si parla di *wrong cuts*). Inoltre la **decentralizzazione** dei dati comporta controlli per verificarne l'integrità e per ridurne al minimo la duplicazione.

"*A poor team will always create a poor system.*"

Capitolo 7

Datacenters

Adiabatic cooling

L'idea è di utilizzare un sistema di raffreddamento basato sull'evaporazione dell'acqua per *pre-raffreddare* l'aria dell'ambiente e che viene utilizzato in modo intermittente: soltanto nei periodi in cui la temperatura all'interno del datacenter è più alta, in particolare durante le ore più calde del giorno e in primavera inoltrata/estate. In questo modo si riesce a raggiungere l'85% di secchezza dell'ambiente.

Power Usage Effectiveness

Il Power Usage Effectiveness (PUE) è l'efficacia nell'utilizzo dell'energia e si misura come:

$$\text{PUE} = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}}$$

dove Total Facility Power indica tutta l'energia utilizzata all'interno del datacenter e IT Equipment Power l'energia usata solo per le apparecchiature informatiche.

Nota bene: PUE è un indicatore per l'energia utilizzata, ma non su quanto sia rinnovabile tale energia.

Datacenter management

L'aspetto di gestione di un datacenter coinvolge diverse attività:

- pianificazione, sia per la manutenzione sia per l'aggiornamento di tutti i dispositivi hardware e software;
- installazione e gestione dei rack e delle connessioni;

- gestione del cabling.

Capitolo 8

Green Computing

La produzione di dispositivi ICT causa **inquinamento** e genera **gas serra** e **e-waste**. Il **green computing** è un *insieme di pratiche per arrivare ad avere un ICT sostenibile a livello ambientale*, minimizzando il consumo elettrico, progettando dispositivi efficienti (a livello energetico) e riducendo l'e-waste.

Per produrre un singolo computer sono necessari 1800 Kg di materiale grezzo. [`news.un.org`]

L'ICT rappresenta il 9% del consumo europeo di elettricità e il 4% delle emissioni europee di carbonio. [`ictfootprint.eu`]

Nature prevede che nel 2030 l'ICT rappresenterà il 20.9% del consumo dell'elettricità mondiale.

Il $\sim$40% del consumo di energia di un datacenter è solo per il raffreddamento. Si ricorda che il PUE non indica l'uso di energie rinnovabili.

Obsolescenza programmata

L'obsolescenza pianificata è quando un *prodotto è deliberatamente progettato per avere una durata ridotta*: la durata della vita deve essere abbastanza lunga da sviluppare il bisogno duraturo di un cliente e il cliente deve considerare che il prodotto è un prodotto di qualità, anche se alla fine deve essere sostituito.

Obsolescenza percepita

L'obsolescenza percepita si verifica quando *un cliente è convinto di aver bisogno di un prodotto aggiornato, anche se il suo prodotto esistente funziona bene.*

The e-waste tragedy

50 milioni di tonnellate di rifiuti elettronici ogni anno (≈770 milioni di lavatrici). Il traffico di rifiuti elettronici è più grande del commercio di droga anche se l'esportazione di rifiuti elettronici è stata vietata da più di 30 anni.

Clicking clean

GreenPeace periodicamente rilascia un rapporto chiamato *Clicking Clean* in cui cerca di arrivare a una valutazione di come si comportano i grandi player del mondo ICT rispetto alla sostenibilità ambientale.

100% *renewably powered digital infrastructures*
or
dangerous climate changes?

Greenpeace ha effettuato il primo benchmarking delle prestazioni energetiche del settore IT nel 2009, sfidando i più grandi attori di Internet a sostenere la propria crescita con il 100% di energia rinnovabile.

Apple, Facebook e Google hanno assunto impegni rinnovabili al 100% nel 2013, ai quali si sono uniti nel 2017 quasi 20 società Internet. Perché?

- Aumento della competitività in termini di costi delle energie rinnovabili (contratti a lungo termine).
- Crescente preoccupazione per il cambiamento climatico tra (dipendenti e) clienti.

L'acquisto diretto di energia rinnovabile da parte delle società negli Stati Uniti è aumentato notevolmente dal 2010, superando i 3,4 GW nel 2015.

Ostacoli a Internet alimentato al 100% da fonti rinnovabili: mancanza di trasparenza e mancanza di accesso alla fornitura di energia rinnovabile (RE).

Nel 2017, Apple era leader per il 3° anno consecutivo tra gli operatori di piattaforme per quanto riguarda la sostenibilità. Sia Apple che Google continuano a guidare il settore nell'abbinare la loro crescita con una fornitura equivalente o maggiore di energia rinnovabile. Switch leader tra gli operatori di colocation per i suoi sforzi per trasferire la sua flotta di data center alle energie rinnovabili il più rapidamente possibile.

Non è chiaro se Amazon Web Services (AWS) sia effettivamente sulla buona strada per diventare alimentato da fonti rinnovabili (mancanza di trasparenza e rapida crescita in Virginia e in altri mercati ampiamente serviti da energia sporca). Le società meno trasparenti, come AWS, Tencent, LG CNS, Baidu, sono anche tra le più dominanti nei rispettivi mercati.

Capitolo 9

From the Cloud to the IoT

L'**Internet of Things** (**IoT**) è un'area in continua in espansione che prevede la connessione di dispositivi elettronici o cyber-fisici (Android wearables, Google Nest, Amazon Echo, Domotica) al fine di creare una serie di applicazioni innovative impressionanti come *embedded AI*, *sistemi di guida autonoma*, *flotte di droni*, *smart agricolture*, *smart city*, ...

"*More than 10 bilion active IoT devices in 2021, more than 25.4 bilion estimated for 2030.*" [`dataprot.net`]

Tutte le applicazioni di IoT, non solo la parte di sensoristica o di attuazione, ma anche reti di sensori che sono dispiegate vicino al terreno, producono una quantità di dati enorme; fortunatamente il cloud è abbastanza grande per accoglierli. Tuttavia per alcune applicazioni questo tipo di architettura non è molto efficace: è necessario filtrare/processare tali dati prima di arrivare al cloud; la latenza, cioè il tempo che intercorre tra la rilevazione di un dato e l'attuazione determinata dalla sua elaborazione, deve essere veramente breve (si pensi ad esempio ai veicoli intelligenti).

Tradizionalmente il deployment delle applicazioni avveniva tramite due modelli principali:

1. IoT + Edge (o edge computing): l'intera applicazione è collocata al confine (edge) della rete; l'idea è di processare i dati direttamente laddove vengono prodotti.
    - Latenze basse.
    - Limitata capacità di calcolo.
    - Difficoltà nella condivisione dei dati.
2. IoT + Cloud: i dati vengono mandati al cloud per l'elaborazione.

- Enorme potenza di calcolo.
- È necessaria una connessione fra i dispositivi coinvolti.
- Latenza più alta.
- Bottleneck dovuti all'ampiezza di banda

A partire da tutto ciò è partita l'idea di questa continuità tra l'IoT e il Cloud che viene spesso chiamata *Fog Computing*.

Il **Fog Computing** mira ad estendere il Cloud verso l'IoT per supportare meglio le applicazioni IoT *latency-sensitive* e *bandwith-hungry*.

C'è un interesse molto grande nel mondo industriale e in quello della ricerca. Ci sono molte sfide aperte:

- deployment adattivo di applicazioni: mandare in esecuzione un'applicazione multi-servizio su un'infrastruttura grande, distribuita ed eterogenea non è semplice;
- la gestione delle applicazioni durante il ciclo di vita del sistema;
- il monitoraggio distribuito su infrastrutture offerte anche da provider diversi;
- problemi di trust, di privacy, di sicurezza in generale;
- fault resilience;
- testbeds: infrastrutture su cui è possibile sperimentare il fog davvero;
- aspetti di business model.

Deployment delle applicazioni

Le applicazioni di nuova generazione sono tipicamente multi-servizio (microservizi), quindi consistono di più "pezzi" che devono essere mandati in esecuzione sui nodi di un'infrastruttura eterogenea. In generale le applicazioni hanno dei requisiti, sia hardware che software, e dei vincoli di qualità.

Dall'altra parte abbiamo l'infrastruttura con le sue capabilities, che ha tre caratteristiche molto importanti: è eterogenea, i nodi non sono tutti uguali, è enorme ed è dinamica.