

# Cloud-based Applications

Mauro Bussini

Service Delivery Manager @icubedsrl

Senior Software Architect

<http://zenprogramming-it.blogspot.it>

[maurob@icubed.it](mailto:maurob@icubed.it)

@mauro\_bussini

openloop



# I 6 ruoli nel team di sviluppo software (1)

Quando si approccia lo sviluppo di un sistema, sono 6 i ruoli (o le figure) che normalmente compongono il team:

- **«Product owner»** – il proprietario del prodotto finale. Può essere egli stesso l'utente utilizzatore (nella maggior parte dei casi non lo è) ma è certamente la persona che decide «cosa vuole» e svolge il ruolo principale perché è quello che trasmette in primis i concetti funzionali e beneficerà maggiormente dal prodotto realizzato
- **«Business Analysts»** – si pongono come figura «di mezzo» tra gli owner del prodotto e il team tecnico. Hanno un ruolo fondamentale perché devono comprendere i bisogni del business e tradurre – per quanto possibile – tali bisogni in un linguaggio comprensibile ad un team di tecnici che spesso non conoscono affatto il business
- **«Software/solution Architect»** - progetta i sistemi, identifica il modo in cui gli sviluppatori dovrebbero costruire il sistema e si pone come guida per tutti gli aspetti tecnici relativi al progetto. Il suo lavoro è importante perché serve a realizzare – almeno concettualmente – lo schema architetturale della soluzione, predisponendo il terreno su cui tutto l'apparato sarà costruito. Un cattivo lavoro fatto dall'architetto, spesso e volentieri, sfocia nel fallimento del progetto o comunque in aumenti di costi o ritardi sulla consegna



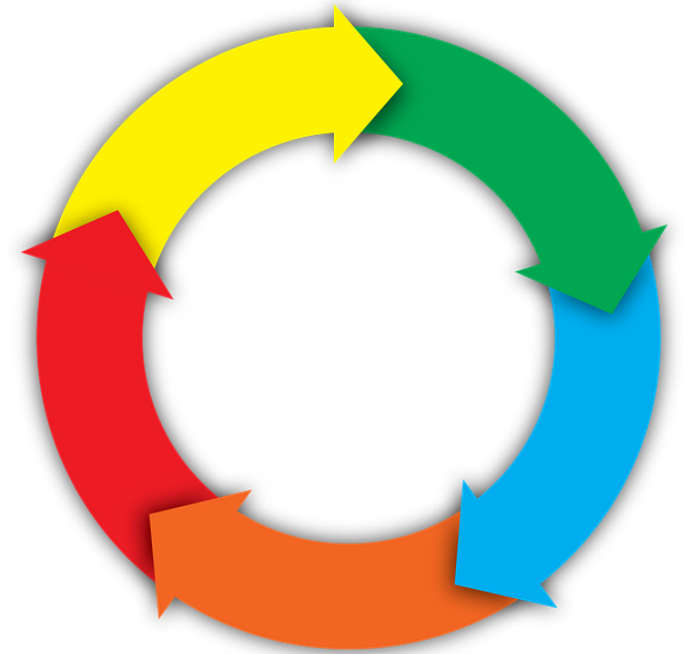
# I 6 ruoli nel team di sviluppo software (2)

- **«Sviluppatori»** – Ereditano i disegni funzionali e tecnici redatti dagli analisti e dall'architetto, spesso aiutano loro stessi ad arricchire tali documenti, quindi li trasformano da semplice concetto - vivo solo su carta - in modelli software eseguibili e in soluzione funzionante. Inutile dire che è il ruolo più importante e percepibile, perché è l'unica figura della catena il cui lavoro è veramente tangibile ed evidente a tutti.
- **«Team QA» (Quality Assurance)** – Detto anche team di «Testing», si assicura che il software scritto dal team di sviluppo corrisponda ai requisiti definiti in fase di analisi. Si assicura che il sistema risponda sia in termini di logica funzionale, sia in termini di performance e di sicurezza. Interagisce con gli sviluppatori in ogni momento, iterando il processo di costruzione/riparazione del sistema attraverso continui raffinamenti (rework) fino alla soddisfazione piena dei requisiti
- **«Team Operations»** – Prende il software finito (e testato) e predispone gli ambienti di esecuzione temporanea (Staging) e finali (Production) occupandosi della configurazione di quelli che sono i requisiti tecnici di esecuzione, e mantenendo la soluzione nel corso del tempo in termini di fruibilità da parte degli utenti finali



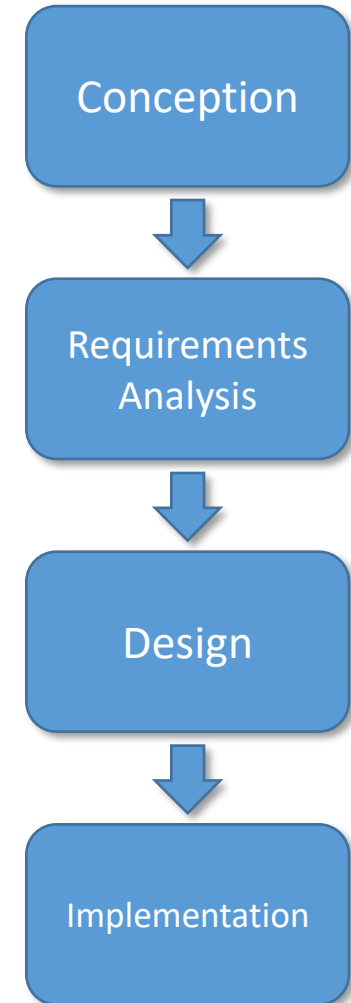
# Software Lifecycle

- E' definito come «una serie di fasi che caratterizzano il corso dell'esistenza di un prodotto software».
- E' il processo che definisce il ciclo di vita di un sistema informativo, dalla sua nascita, alla dismissione in quanto sistema obsoleto.
- E' composto da diverse fasi, ciascuna importante quanto le altre, e quindi meritevoli di una considerazione che **spesso viene meno**; soprattutto dal punto di vista del tempo a loro dedicato.
- Si parte con il concetto di «Conception», seguono numerose fasi operative «intermedie», per arrivare alla fase finale che solitamente è detta «Death» o «Retirement» del sistema.
- La definizione delle fasi intermedie è la parte del Lifecycle di un sistema complesso, che cambia a seconda del «**Software Development Process Model**» utilizzato.



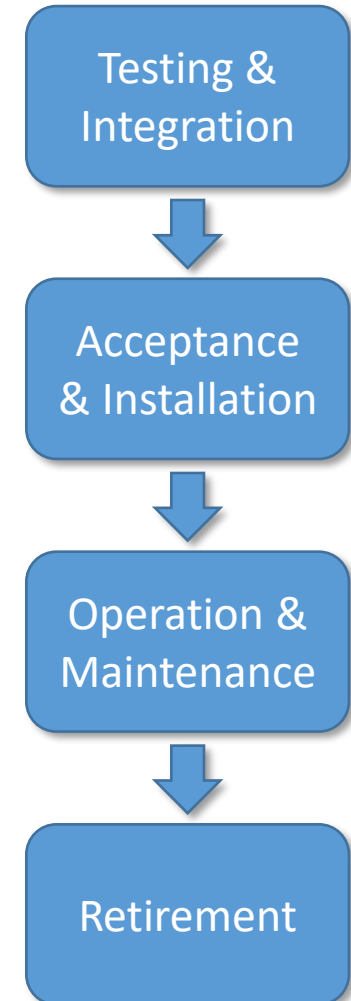
# Costruzione: le fasi del processo (1)

- Conception:
  - "Voglio costruire una casa dove vivere e crescere una famiglia»
- Requirements Analysis:
  - «Che tipo di casa voglio?»
  - «Quante stanze da letto e quanti bagni?»
  - «Ci sarà una cantina o un garage?»
  - «Quale sarà la metratura del giardino?»
- Design:
  - Scelta dell'architetto
  - Disegno della struttura
  - Selezione dell'impresa di costruzione
- Implementation:
  - Posa delle fondamenta
  - Costruzione del tetto
  - Costruzione dei muri esterni ed interni
  - Impianti idraulici e elettrici



# Costruzione: le fasi del processo (2)

- Testing & Integration:
  - Ispezioni sulla struttura
  - Certificazione dell'impianto elettrico e idraulico
  - Certificazione dell'abitabilità
- Acceptance & Installation:
  - Verifica da parte del proprietario
  - Pianificazione del trasloco
- Operation & Maintenance:
  - Trasloco effettivo della famiglia
  - Attività da parte del costruttore per sistemare i problemi di costruzione che possono essere occorsi durante il processo
- Retirement:
  - Dopo molti anni di utilizzo, rinnovamento della costruzione...
  - ...oppure demolizione della casa per avviare una nuova costruzione



# L'importanza di Analisi e Design



Produrre dei buoni modelli di analisi e disegno di un sistema complesso non è mai semplice.

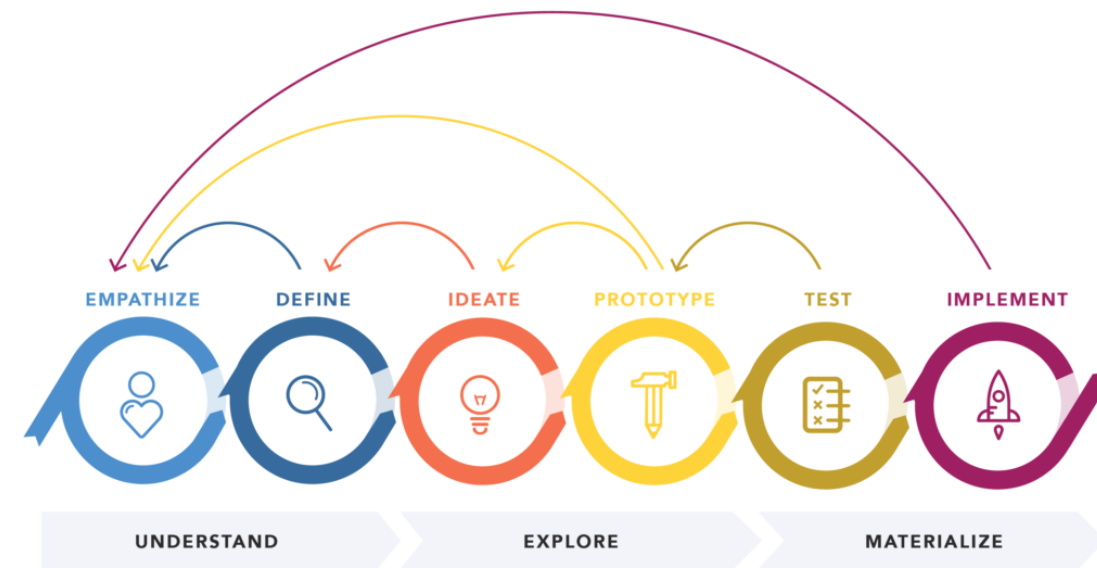
- E' un processo che passa prima di tutto dall'applicazione della **teoria**
- E' fortemente indirizzato dall'esperienza maturata con la **pratica**
- Richiede **sforzo** (parecchio) sia da parte degli utenti finali, che del team tecnico
- E' una fase che **non va mai sottovalutata** nonostante non produca risultati tangibili a breve termine

# Metodologie di analisi «iterative»

Sebbene sia possibile realizzare uno sviluppo orientato agli oggetti utilizzando un modello «waterfall», la maggior parte dei sistemi object-oriented vengono sviluppati con un approccio «iterativo», dove i processi di "analisi e progettazione" sono spesso considerati **contemporaneamente**.

Con i modelli iterativi è possibile lavorare su diverse fasi in **parallelo**. Ed esempio è possibile - e non viene considerato come una fonte di errore - lavorare sull'analisi, il design e addirittura codificare tutto nello stesso giorno, accettando di avere problemi di **impatto minimale** tra una fase e l'altra.

La logica che guida il modello iterativo è che lo sviluppo di un software è un **processo intensivo di conoscenza**, dove fasi come l'analisi non possono essere realmente comprese senza aver affrontato problemi di progettazione; dove è ammesso che problemi di codifica possano **cambiare il design** e dove il test può fornire informazioni su come il codice o addirittura il disegno deve essere modificato.





# Metodologie «iterative»: caratteristiche

- Processo «iterativo» 😊:
  - Ogni fase è iterata più di una volta nel corso dell'esecuzione dell'intero processo
- Di dice che sono «Risk-Driven»
  - Il modello di valutazione dei rischi è «build-in» nel modello stesso, e lo considera parte integrante del processo
  - Ogni rischio è valutato nella pianificazione dell'iterazione
  - Si propongono rimedi per limitare gli impatti dei rischi
  - La fase di pianificazione è considerata in ogni iterazione, è dinamica ed adattiva alle esigenze del progetto
- Ogni iterazione ha diversi obiettivi da portare a compimento prima della conclusione:
  - Incrementare le funzionalità percepite del sistema finale
  - Ridurre i rischi accumulati in precedenza in modo che impattino in maniera minima durante le iterazioni successive



# Manifesto Agile

Riassume in maniera semplificata i principi su cui si basa la metodologia Agile, mostrando quanto si possa arrivare ad un prodotto di qualità dando valore al dialogo e collaborazione tra le persone.

- Rimarca come un software funzionante è più importante di qualsiasi documentazione.
- Quanto è importante fondare il processo sulla collaborazione con il cliente finale e la sua esperienza
- In ultimo - probabilmente la cosa più importante – enfatizza come è fondamentale essere in grado di «rispondere efficacemente al cambiamento», indirizzando le chieste funzionali dell'ultimo minuto senza per forza sconvolgere il processo di costruzione del sistema.

## The Agile Manifesto

<b>Individuals and interactions</b>	over	Processes and Tools
<b>Working Product</b>	over	Comprehensive Documentation
<b>Customer Collaboration</b>	over	Contract Negotiation
<b>Responding to change</b>	over	Following a plan

*That is, while there is value in the items on the right, we value the items on the left more.*

[www.agilemanifesto.org](http://www.agilemanifesto.org)

# I Principi su cui si basa Agile (1)

- L'obiettivo è soddisfare il cliente consegnando continuamente versioni funzionanti del sistema
- Accettare i cambiamenti in corsa
- Consegnare software frequentemente, ogni due settimane o ogni due mesi (meglio 2 weeks)
- Deve esistere una collaborazione tra il team tecnico e chi conosce il business
- É fondamentale mantenere i partecipanti motivati
- Preferire una comunicazione faccia a faccia rispetto la scrittura di documenti



# I Principi su cui si basa Agile (2)

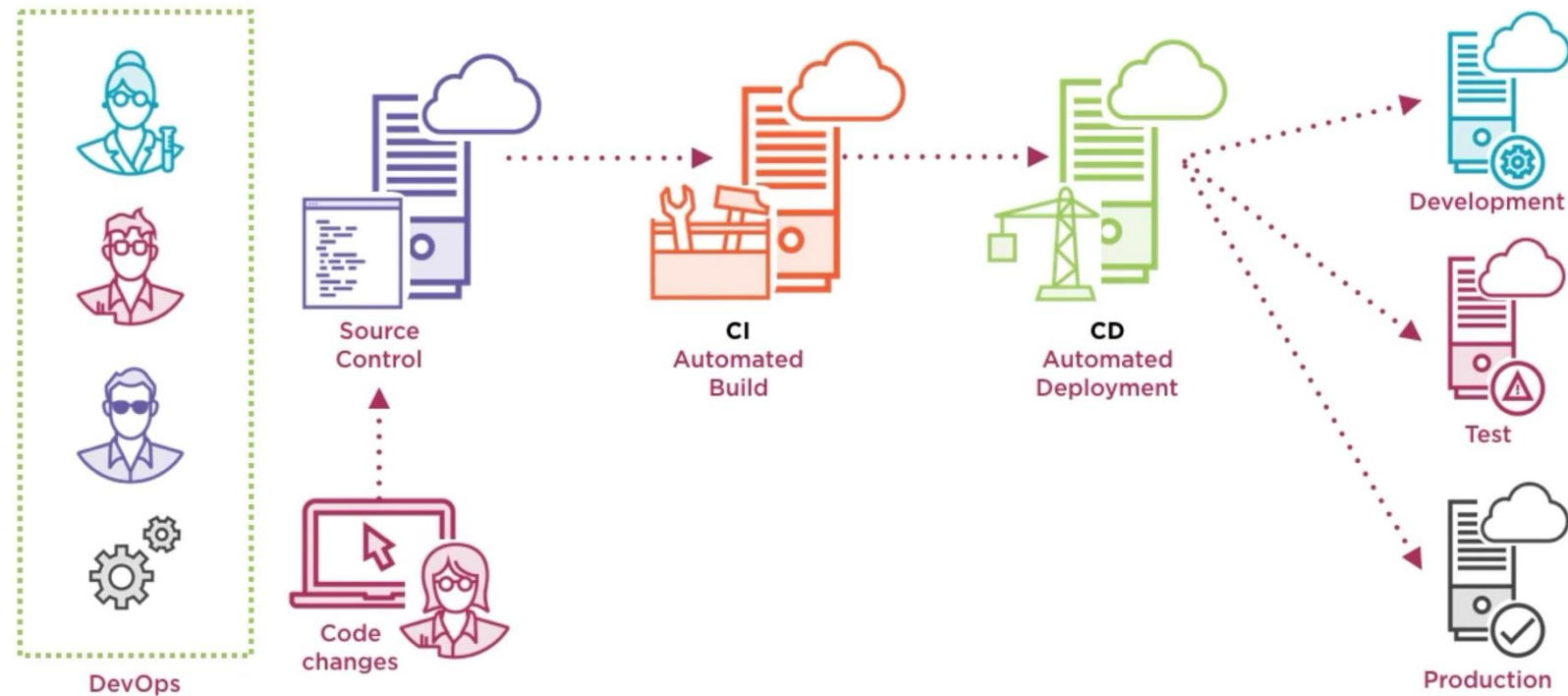
- La misurazione del progresso di un progetto si misura con dell'avanzamento delle funzioni del sistema
- Il processo Agile richiede che il progetto sia sostenibile dal team: evitare lavoro overtime!
- Dare molta importanza all'eccellenza tecnica, sempre
- Mantenere il design il più semplice possibile
- Le migliori architetture derivano da team auto organizzati piuttosto che supervisionati
- Ad intervalli regolari il team si ferma per valutare la qualità del suo lavoro e cerca di porre rimedio agli errori commessi



# Automation: Source control, CI e CD

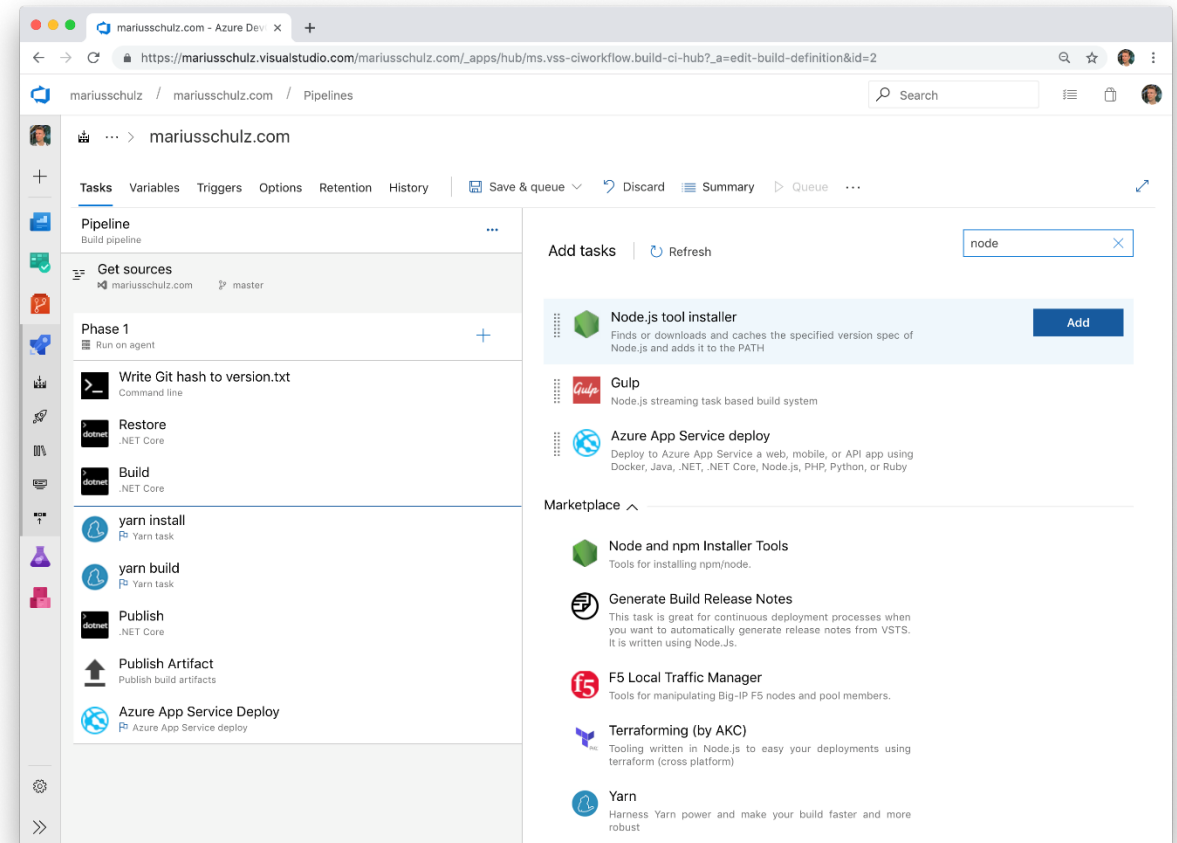
Con il termine «automation» si identificano una serie di operazioni automatiche fatte sul sistema software che permettono – partendo dal codice sorgente dell'applicazione – di mettere a disposizione dell'utente finale un sistema funzionante ed operativo in un lasso di tempo brevissimo.

L'automazione sfrutta una serie di meccanismo in grado di procedere alle varie attività di deploy senza l'intervento umano, ma partendo da configurazioni ben definite.



# Automation: CI e CD con Azure DevOps

- L'utilizzo di Azure DevOps per la gestione dell'ALM influenza le fasi di pianificazione, sviluppo, distribuzione e operatività.
- Ogni fase è basata sulle altre e nessuna fase è assegnata a un ruolo specifico.
- In una cultura DevOps effettiva ogni ruolo è coinvolto in qualche misura in ogni fase.



# Welcome to “the Cloud”...

---

Il mondo Cloud si basa su principi e teorie che sono vecchie di decenni.

Il cloud moderno ha però solo circa 15 anni

Inizia quando Amazon entra nel mercato nel 2006

Nasce con la necessità di scalare per incrementare i propri servizi al pubblico

Ma l'idea del cloud, si sviluppa sulla disponibilità di Amazon di potenza di calcolo, inutilizzata fuori dai periodi di picco





# Cosa significa esattamente "Cloud"?

---

- Costruire applicazioni cloud native che scalano in maniera automatica
- Migrare e modernizzare applicazioni esistenti
- Ripartire le architetture enterprise tra cloud pubblico e privato e gestirle come unica entità – hybrid cloud



# Caratteristiche base del cloud (1/2)

- **Cloud è self service**
  - Gli sviluppatori non devono mettersi in fila per le risorse: le possono richiedere in autonomia.
  - Vuol dire «comprare» potenza di calcolo e servizi quando servono
- **Cloud è agilità**
  - Il cloud consente agli sviluppatori di non fermarsi perché ci sono dipendenze dal lavoro di altri team.
  - Questa indipendenza è ottenuta grazie ad innovazioni architetturali, come i microservices, e alla possibilità di sfruttare maggiormente un «self service provisioning».
- **Cloud è scalabilità**
  - È rendere un'applicazione libera dai limiti di una singola VM o dell'hardware fisico



# Caratteristiche base del cloud (2/2)

- **Cloud è un set di servizi**
  - Il cloud è servizi che consumiamo liberamente, focalizzandoci sul valore aggiunto che possiamo offrire ai clienti.
  - Terze parti creano servizi utilizzabili facilmente attraverso un meccanismo di «consumption».
  - Noi possiamo utilizzare questi servizi facilmente all'interno delle nostre applicazioni.
- **Cloud è diventare serverless**
  - Il cloud è scrivere codice e lasciare che il cloud lo esegua al meglio.
  - Noi costruiamo la logica, creiamo le API per consumarla e la rendiamo utilizzabile ad altri all'interno dell'azienda.
  - La scalabilità è virtualmente illimitata.
  - Paghiamo solo per l'effettivo consumo.



# Il cloud è molto più di un computer in rete

- Molta gente confonde il termine cloud con "qualcosa su internet"
- In realtà, cloud vuol dire qualcosa di molto specifico
- La definizione più semplice di cloud è «un datacenter» che è pieno di hardware replicabile, che nessuno tocca più dopo averlo installato...solo per buttarlo via quando fallisce
- Nel mezzo, deployment, aggiornamenti, manutenzione e gestione sono del tutto automatizzati



# I vantaggi del cloud computing

Il cloud computing rappresenta un grande cambiamento rispetto alla visione tradizionale delle aziende in materia di risorse IT. Cos'è il cloud computing? Perché è così diffuso? Ecco sei motivi comuni per cui le organizzazioni ricorrono ai servizi di cloud computing:

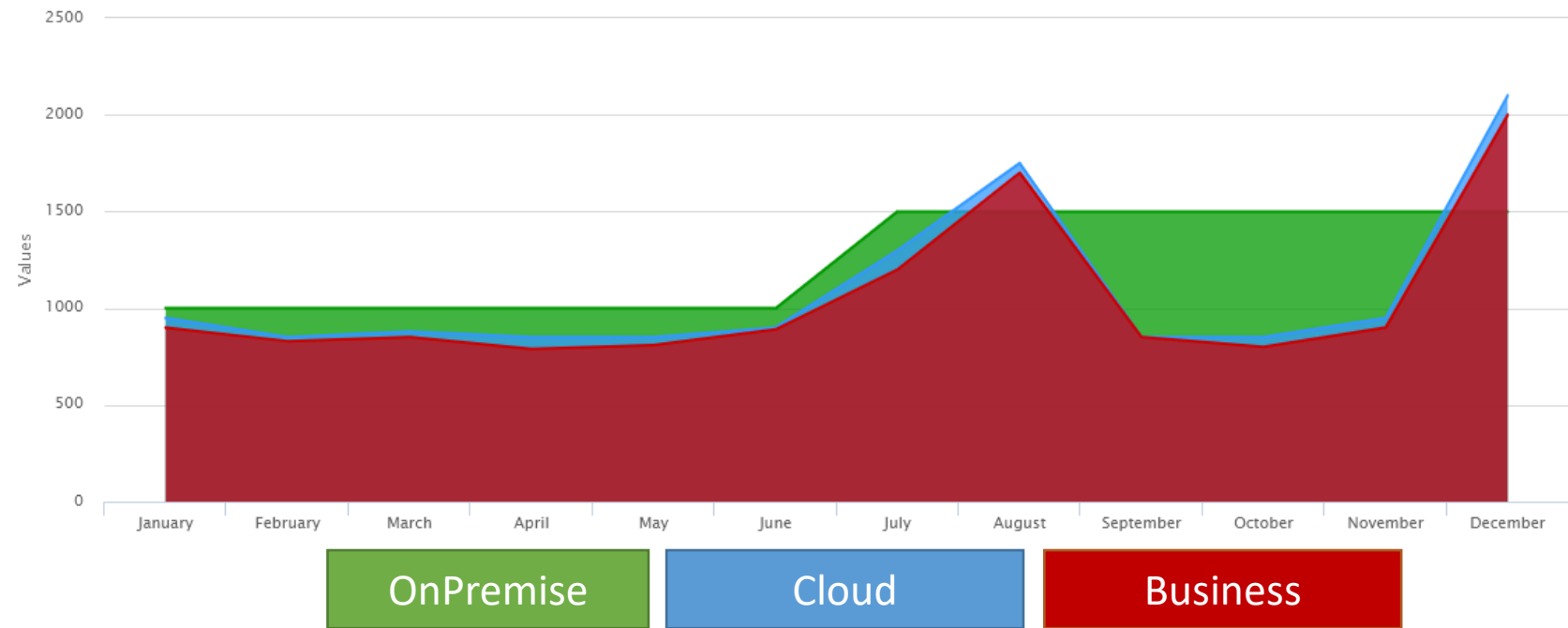
1. **Costo:** Il cloud computing elimina le spese di capitale associate all'acquisto di hardware e software e alla configurazione e alla gestione di data center locali, che richiedono rack di server, elettricità 24 ore su 24 per alimentazione e raffreddamento ed esperti IT per la gestione dell'infrastruttura. I conti tornano in fretta.
2. **Velocità:** La maggior parte dei servizi di cloud computing viene fornita in modalità self-service e on demand, quindi è possibile effettuare il provisioning anche di grandi quantità di risorse di calcolo in pochi minuti, in genere con pochi clic del mouse, e questo garantisce alle aziende eccezionale flessibilità senza la pressione legata alla necessità di pianificare la capacità.
3. **Scalabilità globale:** I vantaggi dei servizi di cloud computing includono la possibilità di usufruire di scalabilità elastica. In materia di cloud questo significa fornire la giusta quantità di risorse IT, ad esempio una quantità maggiore o minore di potenza di calcolo, risorse di archiviazione e larghezza di banda, proprio quando è necessario e dalla posizione geografica appropriata.
4. **Produttività:** I data center locali richiedono in genere molto spazio per rack e impilamento dei server, nonché configurazione di hardware, applicazione di patch software e altre attività di gestione IT dispendiose in termini di tempo. Il cloud computing elimina la necessità di molte di queste attività, consentendo ai team IT di dedicare il loro tempo al raggiungimento di obiettivi aziendali più importanti.
5. **Prestazioni:** I più grandi servizi di cloud computing vengono eseguiti su una rete mondiale di data center sicuri, aggiornati regolarmente all'ultima generazione di hardware, veloce ed efficiente. Questo offre diversi vantaggi rispetto a un singolo data center aziendale, tra cui latenza di rete ridotta per le applicazioni e maggiori economie di scala.
6. **Affidabilità:** Il cloud computing aumenta la semplicità e riduce i costi di backup dei dati, ripristino di emergenza e continuità aziendale, grazie alla possibilità di eseguire il mirroring dei dati in più siti ridondanti nella rete del provider di servizi cloud.



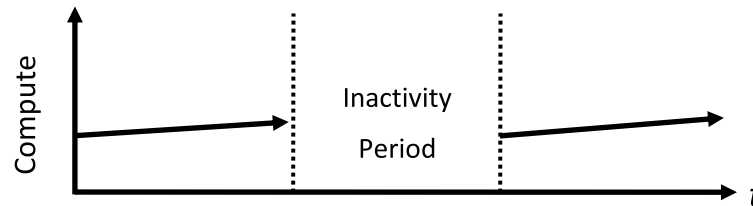
# Scalabilità e costi: on-premise vs cloud

Con una soluzione tradizionale la scalabilità si raggiunge solo aumentando le capacità hardware.

Con una soluzione moderna, ogni componente scala indipendentemente ed è più semplice arrivare a scalare solo i componenti dove c'è vera necessità di maggiori risorse

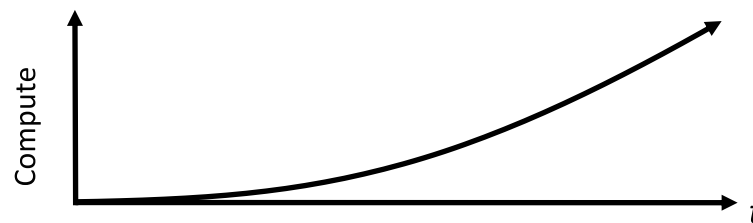


# Pattern di cloud computing



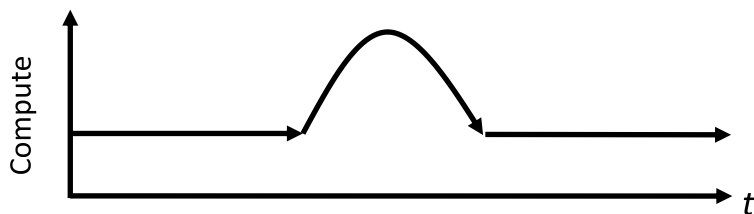
## On and Off

On & off workload (ie batch job)  
La capacità over provisioned è sprecata  
Il Time to market può essere difficile



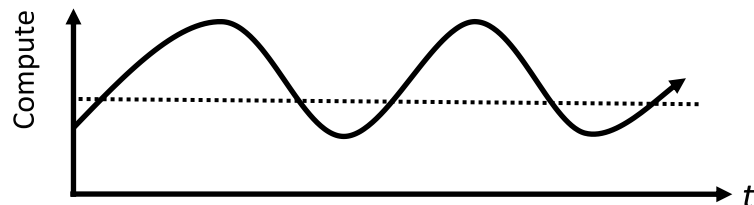
## Growing Fast

Servizi di successo devono poter crescere/scalare  
Stare al passo con la crescita è uno dei problemi più difficili da affrontare per l'IT – ie: hardware



## Unpredictable Bursting

Non attesi/non stimati picchi nella richiesta  
I picchi non gestiti impattano sulle performance  
Non possiamo fare provisioning per i casi estremi



## Predictable Bursting

Servizi con stagionalità ben definita  
Picchi dovuti a incremento della domanda



# Spostare le applicazioni chiave, dove serve

Il cloud consente di spostare applicazioni e workload:

- Modernizzandoli in maniera più semplice.
- Offrendone i servizi attraverso API.
- Gestendone più facilmente la scalabilità
- Posizionandole in maniera strategica vicino all'utente



# Moderno, sicuro, monitorabile

- Fare cloud significa modernizzare
  - Far crescere il business attraverso nuovi use case
  - Esporre servizi via API, «Developer-Friendly»
- Fornire soluzioni senza «lock-in»
  - Si possono spostare le applicazioni nel provider che fornisce maggior rapporto tra costi e benefici
  - Sono implementabili architetture «ibride» (cloud-onpremise) oppure «multiprovider»
- E' sicuro per definizione
  - Non dobbiamo gestire noi la sicurezza fisica
  - Ci possiamo focalizzare su applicazioni e infrastrutture
- I costi sono monitorabili
  - Attraverso analytics e analisi di carico
- Permette di avere maggiore automazione
  - DevOps diventa una necessità (ripagata abbondantemente)





# IaaS, PaaS e SaaS



IaaS

Infrastructure-as-a-Service

host



PaaS

Platform-as-a-Service

build



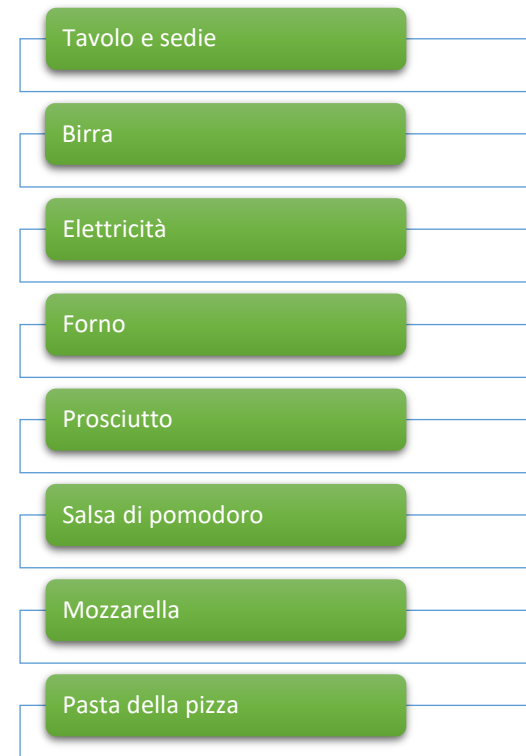
SaaS

Software-as-a-Service

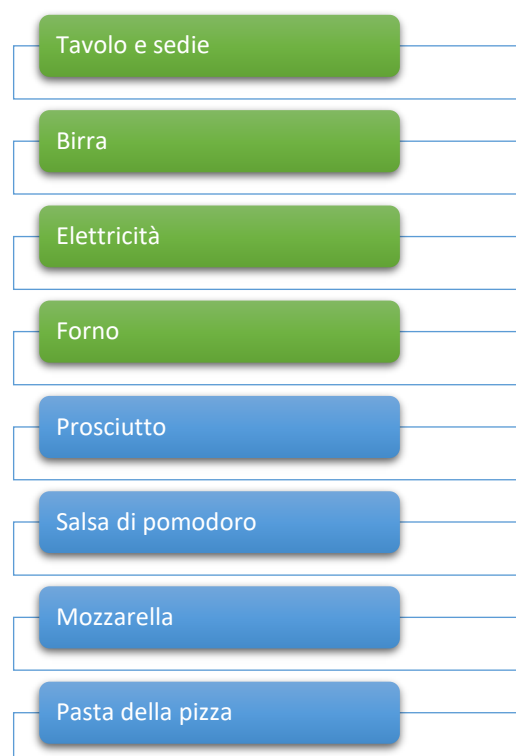
consume

# IaaS, PaaS e SaaS: una metafora

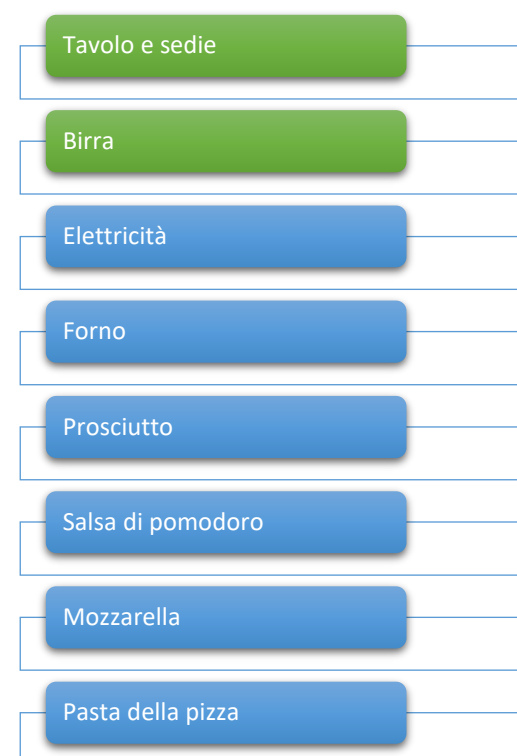
## ONPREMISE



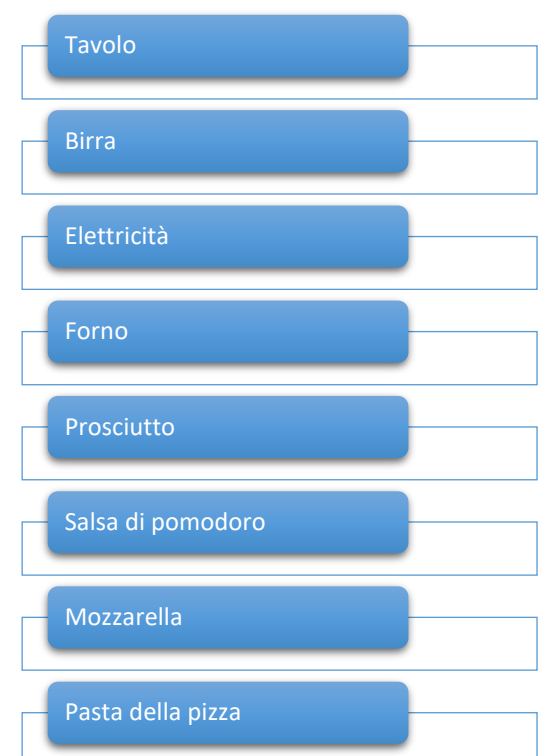
## IAAS



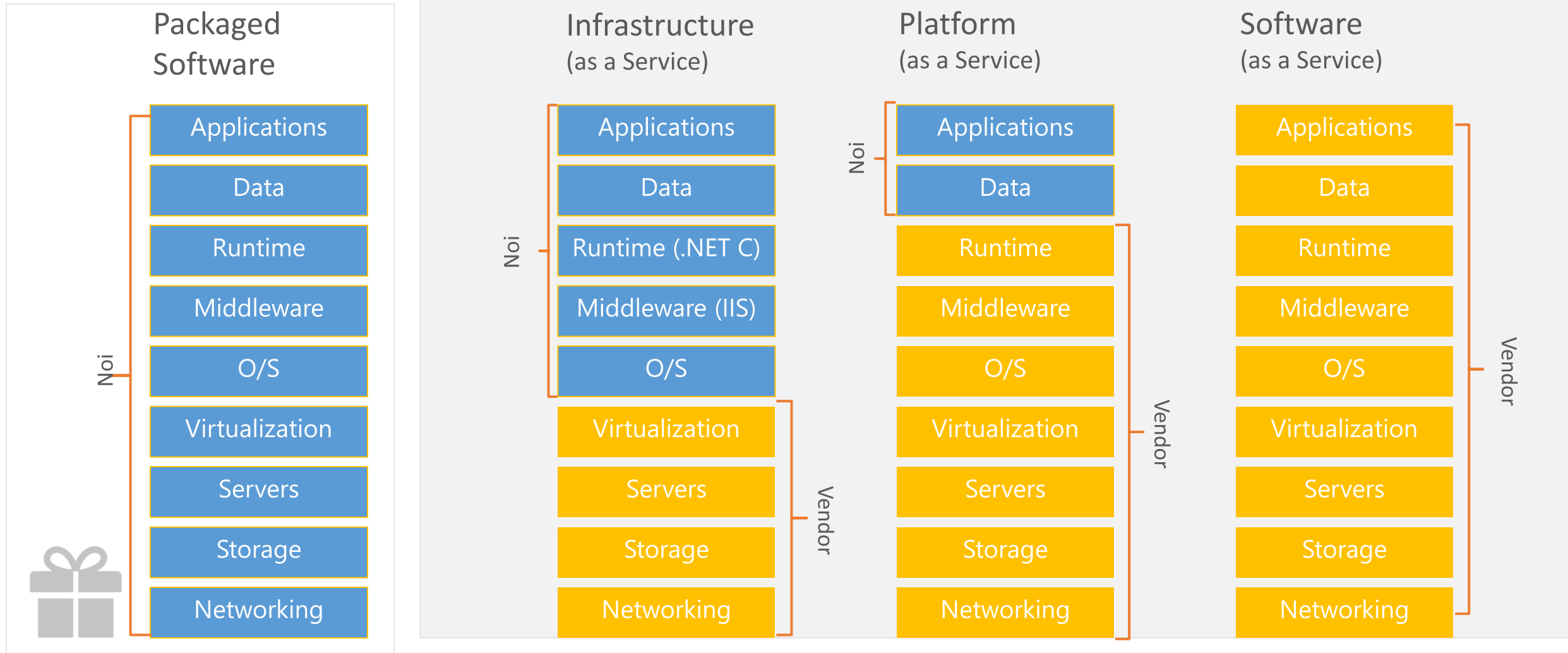
## PAAS



## SAAS



# Una visione più dettagliata



# Porting di applicazioni nel cloud

L'approccio più semplice per portare applicazioni «onpremise» nel cloud è quello di spostare tutte le risorse locali in corrispondenti risorse nel datacenter

- Si applica un approccio denominato «Lift and Shift»
- Usare una soluzione IAAS è certamente più semplice perché non necessita di alcun cambio architetturale

Ma il «vero cloud» sono le soluzioni PAAS e Serverless

- Scalabili
- Georeplicabili
- Focus sulle funzionalità e non sull'infrastruttura



# Gli approcci più scalabili

## Platform as a Service

- Permette di fornire un focus sulle applicazioni che sviluppiamo
- Gestiamo solo codice
- Permette una scalabilità virtualmente infinita

## Serverless

- Il focus è al 100% sul codice e non sulla applicazione
- Si realizzano «funzioni» mirate all'obiettivo di business
- La scalabilità, virtualmente infinita, è applicata con un modello «pay-per-call»



**60+** regions worldwide **140** available in 140 countries

Available region

Announced region

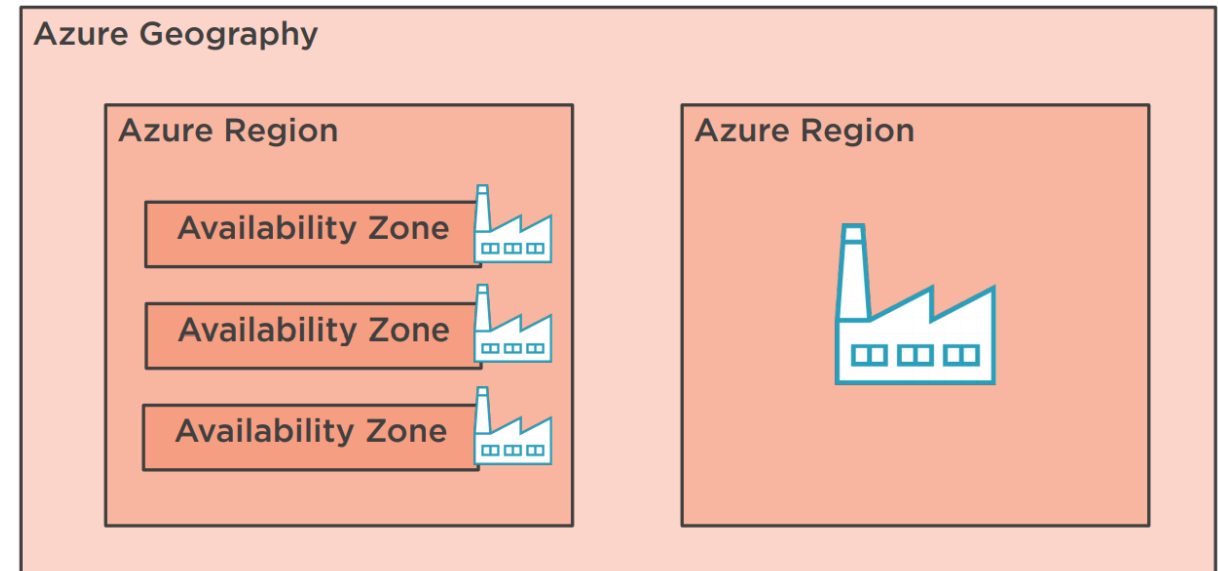
Availability Zones

Conta  
datacenter  
distribuiti in  
tutto il globo,  
vicino agli  
utenti, e in  
posizioni  
logisticamente  
strategiche

# Geography, Region e Availability Zone

Il concetto di geoposizionamento in Azure è forte:

- Geography è un supercontenitore di regioni
- Region è un contenitore di zone di disponibilità
- Availability Zone contiene le risorse fisiche (ma potrebbe non essere specificata ed essere quindi il "default")

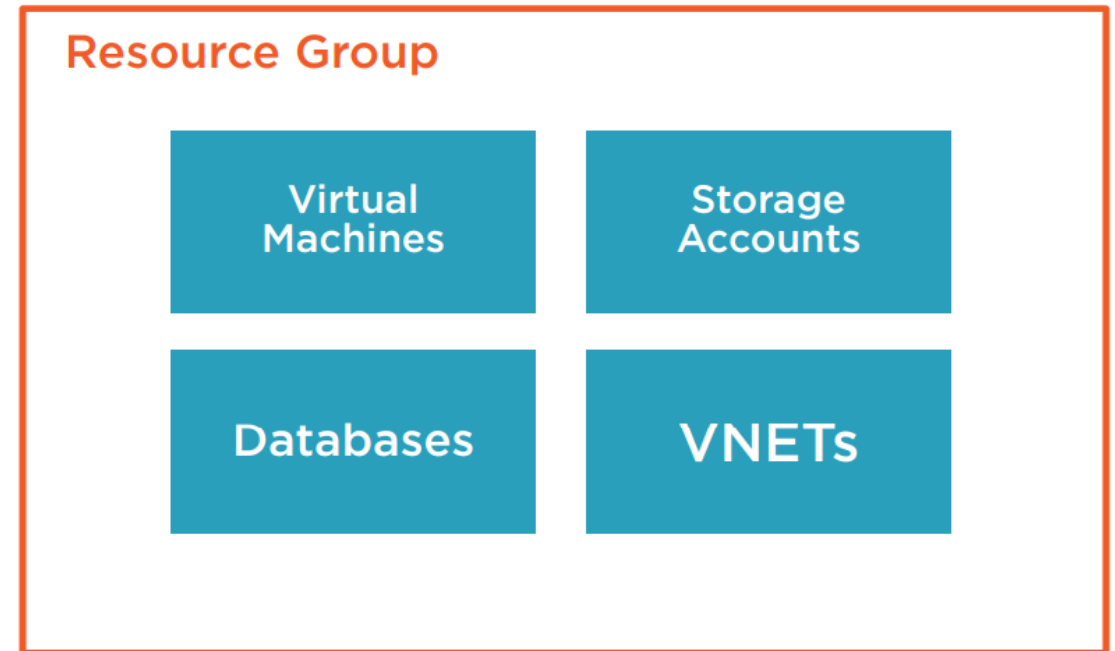


# Resource Groups

E' un contenitore logico per le risorse che possono essere create in Azure.

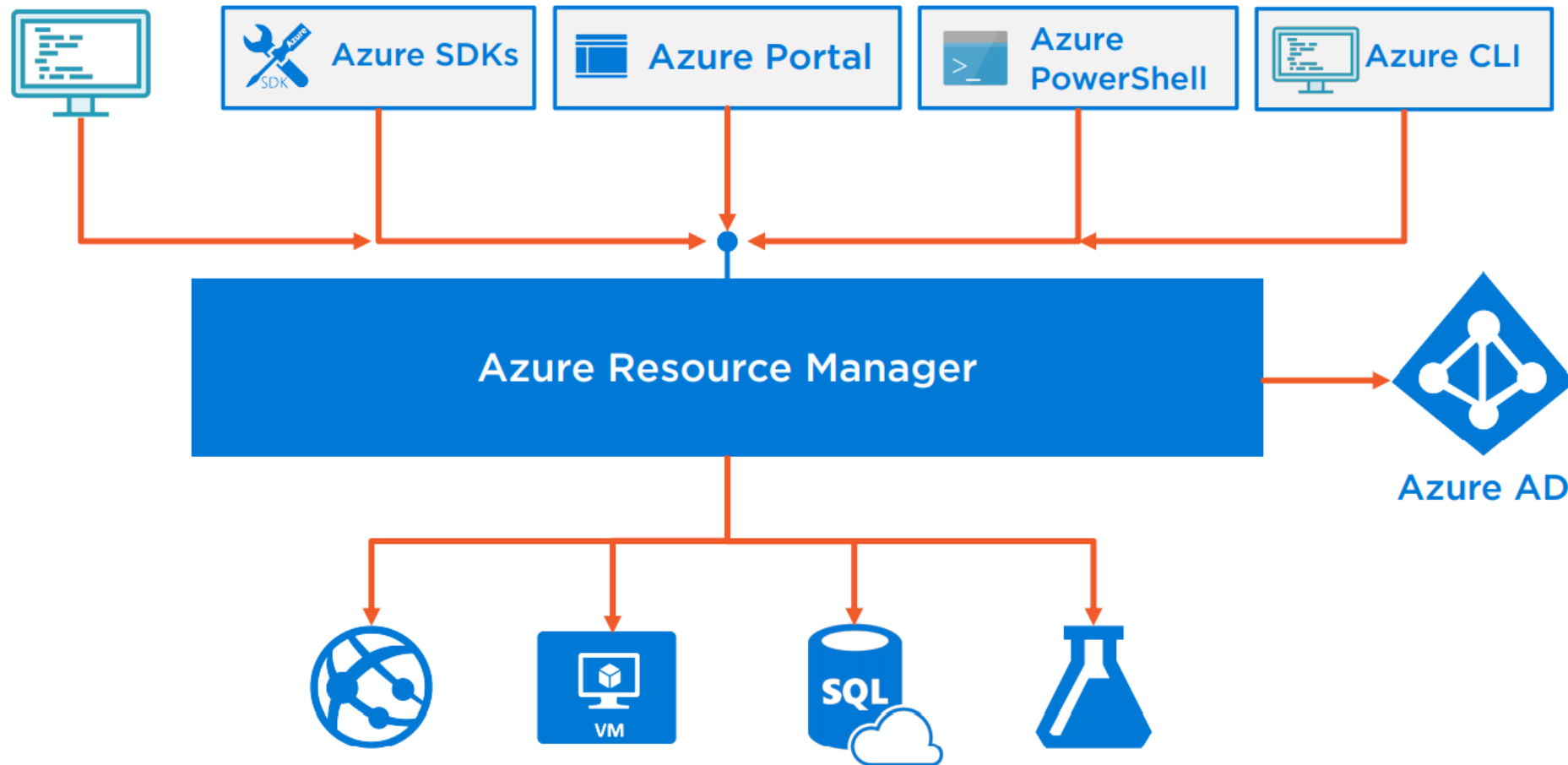
E' contenuto (solo come metadata) all'interno di una Region, ma può contenere risorse afferenti ad altre Regions.

Una risorsa può essere legata a un solo Resource Group, e cancellando questo tutte le risorse vengono eliminate.





# Come interagire con Azure?



# Azure Virtual Machines: overview

Le macchine virtuali (VM) di Azure sono una delle risorse di calcolo «scalabile» presene su Azure.

La scelta di utilizzare una macchina virtuale invece che un servizio «managed» solitamente è dovuta alla necessità di avere un maggior controllo sull'ambiente, in termine di configurazione e monitoraggio.

Una VM di Azure offre la flessibilità della virtualizzazione senza dover acquistare e gestire l'hardware fisico su cui è in esecuzione la macchina virtuale.

E' responsabilità dell'utente gestire la macchina nella sua interezza:

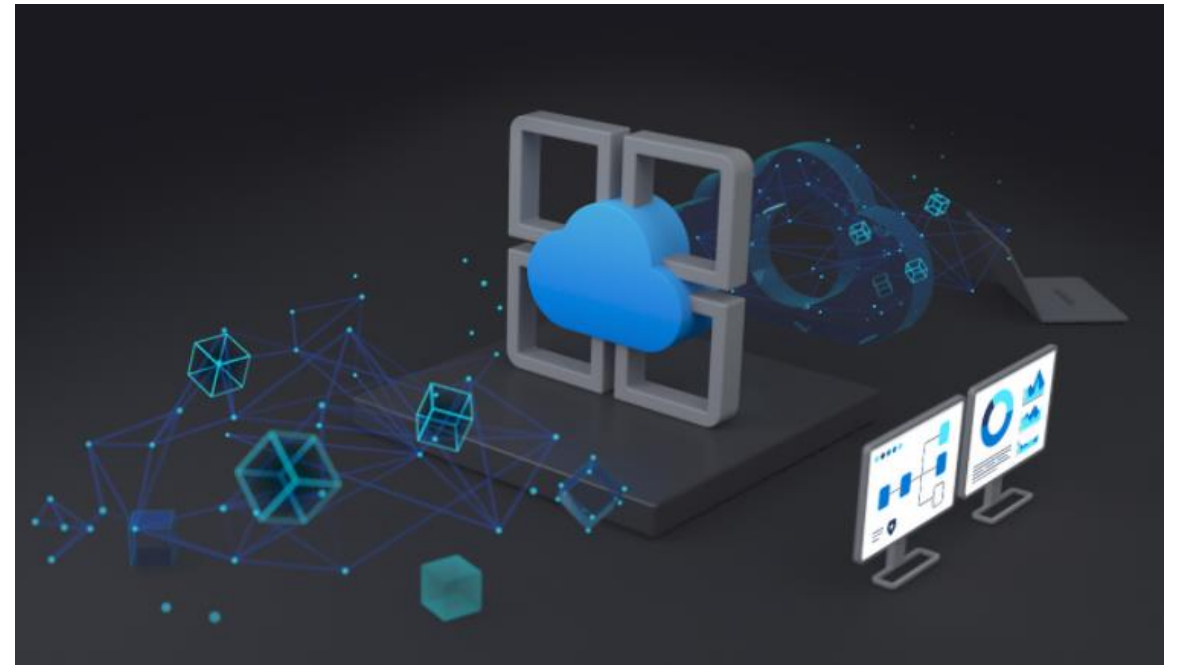
- Configurazione
- Aggiornamento
- Installazione software di middleware
- Gestione e manutenzione delle applicazioni sul server



# Azure App Services: API e service workers

Attraverso questo servizio «managed» si possono creare, distribuire e dimensionare rapidamente applicazioni Web e API basate su .NET, .NET Core, Node.js, Java, Python o PHP in contenitori o in esecuzione su Windows o Linux.

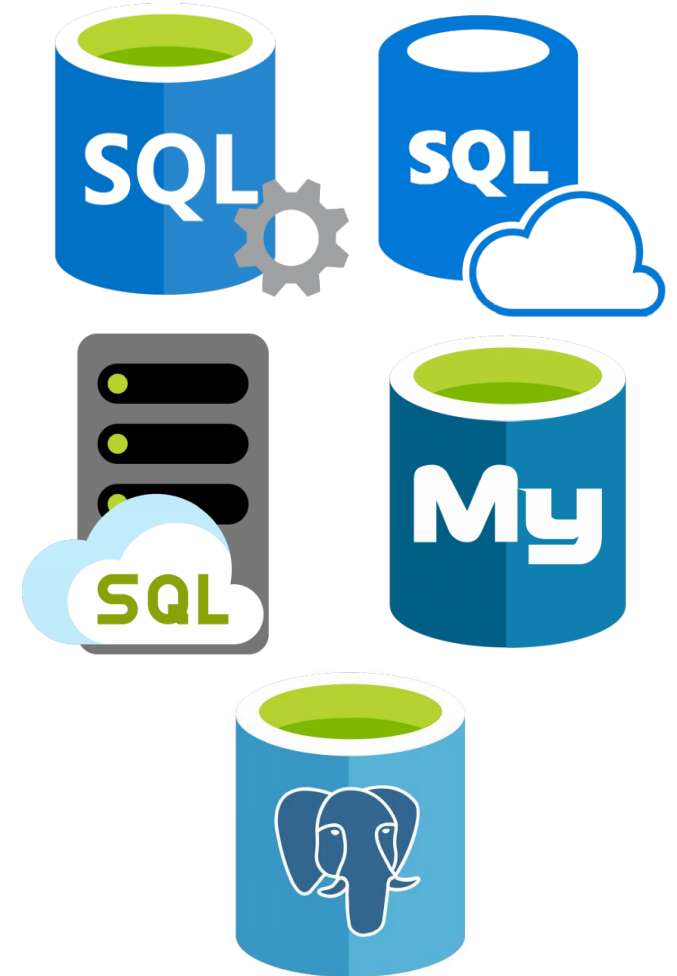
Tutto lo stack è pronto e disponibile in pochi istanti, e lo sviluppatore si deve solo preoccupare dello strato applicativo, senza curarsi dell'ambiente che è completamente in carico a Microsoft.



# I database relazionali in Azure

Essendo la tipologia di database tradizionalmente più usata, esistono numerosi diversi servizi dedicati:

- SQL Server su Macchina Virtuale: approccio classico e non gestito del motore Microsoft (IaaS)
- SQL Azure: completamente gestito come PaaS
- SQL Managed Instance: è SQL Server standard ma completamente gestito e isolato all'interno di Azure
- Azure database per MySQL: istanza gestita del celebre database Oracle
- Azure database per PostgreSQL: particolarmente utile per soluzioni GIS e Geometry Data



# Azure SQL: relazionali “as a service”

Rappresenta l'unica soluzione “nativa managed” per la gestione di database relazionali su Azure.

E' a tutti gli effetti una istanza di SQL Server gestita interamente da Azure, con tutte le funzioni del prodotto standard.

La realtà è che implementa le funzioni della “prossima versione” di SQL Server disponibile per le soluzioni “on premise”.



# Database documentali: i documenti

Il concetto fondamentale è quello di «Documento». Ogni implementazione differisce nei dettagli della definizione di documento, ma comunque in generale si assume che siano incapsulati e codificati i dati o le informazioni in base ad uno standard.

Le codifiche più comuni sono XML, YAML, JSON, e BSON così come formati binari tipo PDF o Microsoft Office.

I documenti all'interno della base di dati sono simili ai record dei database relazionali, ma sono meno rigidi.

Non è richiesta l'adesione ad uno schema standard: questo rende i database documentali molto flessibili.

```
{  
  Nome:"Mario",  
  Indirizzo:"Via Veneto 10",  
  Hobby:"Calcio"  
}
```

```
{  
  Nome:"Luca",  
  Indirizzo:"Via del Popolo 20",  
  Figli:[  
    {Nome:"Annamaria", Eta:3},  
    {Nome:"Luigi", Eta:2}  
  ]  
}
```

# CosmosDB: il database documentale

Negli ultimi 7-8 anni i database documentali sono ritornati prepotentemente sul mercato.

Non sono un concetto nuovo: rappresentano i primi database immessi sul mercato, sul finire degli anni '70 e inizio degli anni '80.

La loro velocità e flessibilità li ha resi molto popolari, e non esiste un solo servizio cloud che non ne implementa una versione.

Microsoft ha prima proposto "DocumentDB", che è stato mutato in CosmosDB, l'attuale implementazione "as a service" presente su Azure.

Particolarità di CosmosDB sono le sue API: si passa da un approccio "Core SQL", ad un linguaggio aderente al 100% con le API di MongoDB, probabilmente la soluzione documentale più celebre e conosciuta del mercato, onpremise e in cloud.



# CosmosDB: la flessibilità

La soluzione è talmente flessibile che permette di accedere alle informazioni contenute utilizzando 5 differenti API:

- SQL API: linguaggio strutturato simile a SQL-ANSI, con clausole relazionali
- Cassandra: database nato in Facebook che entra nel mondo dei database wide-column
- MongoDB: celebre incarnazione di un database documentale
- Gremlin: supporto per database a grafi
- Azure Table Storage: servizio precedentemente inserito tra i blob storage

SQL API

Cassandra

MongoDB

Gremlin

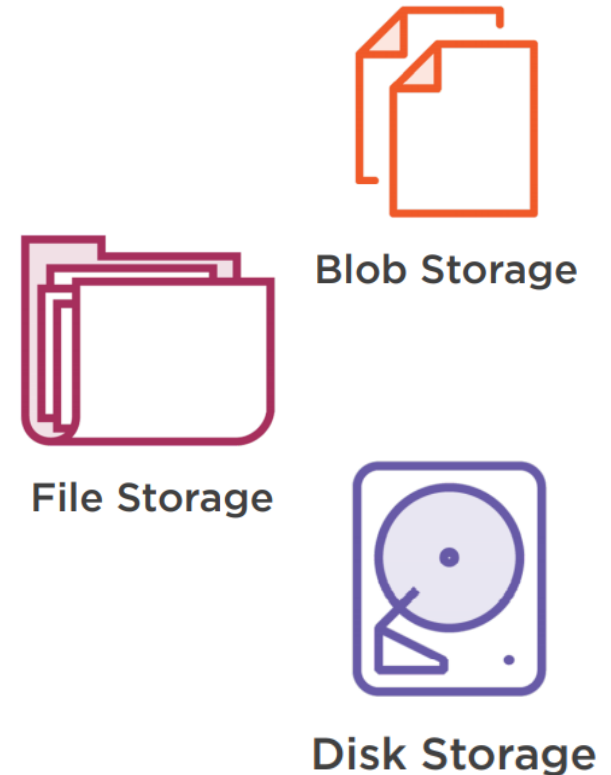
Azure Table  
Storage



# Storage Account: tipologie di servizio

Esistono 5 differenti tipi di servizio che sono contenuti all'interno di uno Storage Account.

- Blob Storage: dedicati all'archiviazione di dati non strutturati e di larga scala
- File Storage: simili ai Blob Storage, ma possono essere "montati" come risorse di rete (share) su machine virtuali
- Disk Storage: vengono utilizzati come dischi per le machine virtuali e offrono un eccellente bilanciamento di velocità di lettura/scrittura
- Esistono anche Table Storage e Queue Storage, e sono la versione più «leggera» di servizi presenti in Azure per l'archiviazione semi-strutturata e i meccanismi di coda.



# BlobStorage: archiviazione massiva

Azure Blob Storage ti aiuta a creare dei "Data Lake" per le tue necessità di archiviazione massiva di dati.

Ottimizza i costi con uno storage ritagliato sulle esigenze applicative, offre archiviazione a lungo periodo, scalabilità e flessibilità di poter avere un equilibrio tra prestazioni e costi:

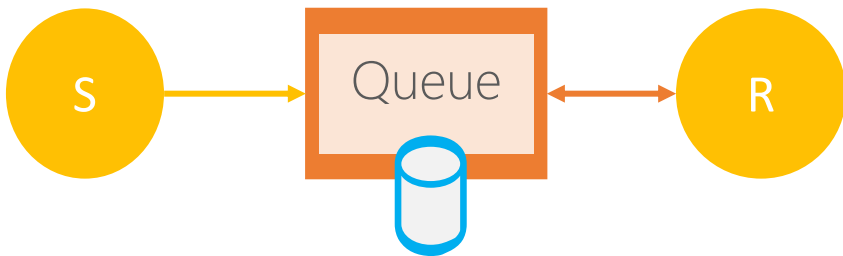
- Scalabilità e geo-replica
- Sicurezza integrate
- Ottimizzato per grosse quantità di dati
- Gestione semplice ed immediata



# EventBus: code ed eventi

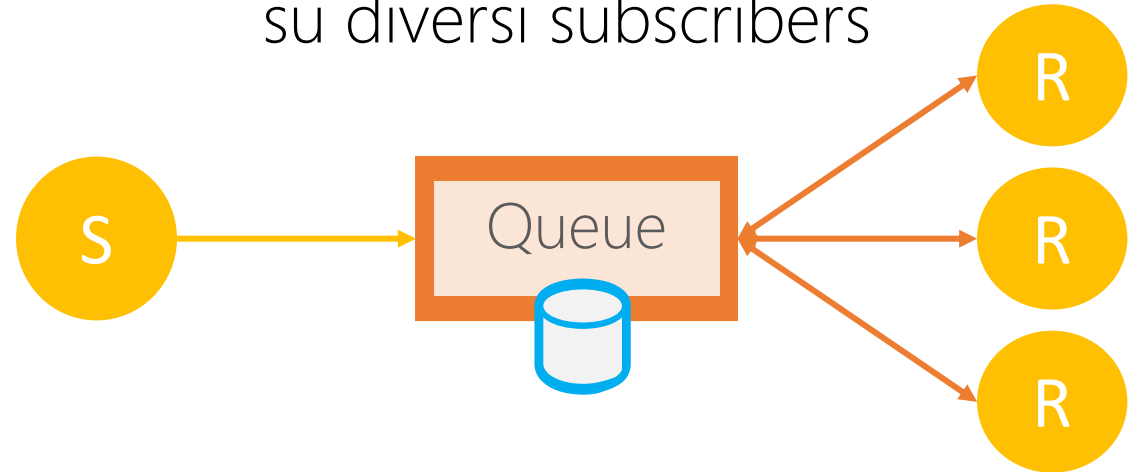
## Queue (coda)

- Comunicazione asincrona
- Permette di scalare il lavoro evitando che lo stesso task venga eseguito più volte



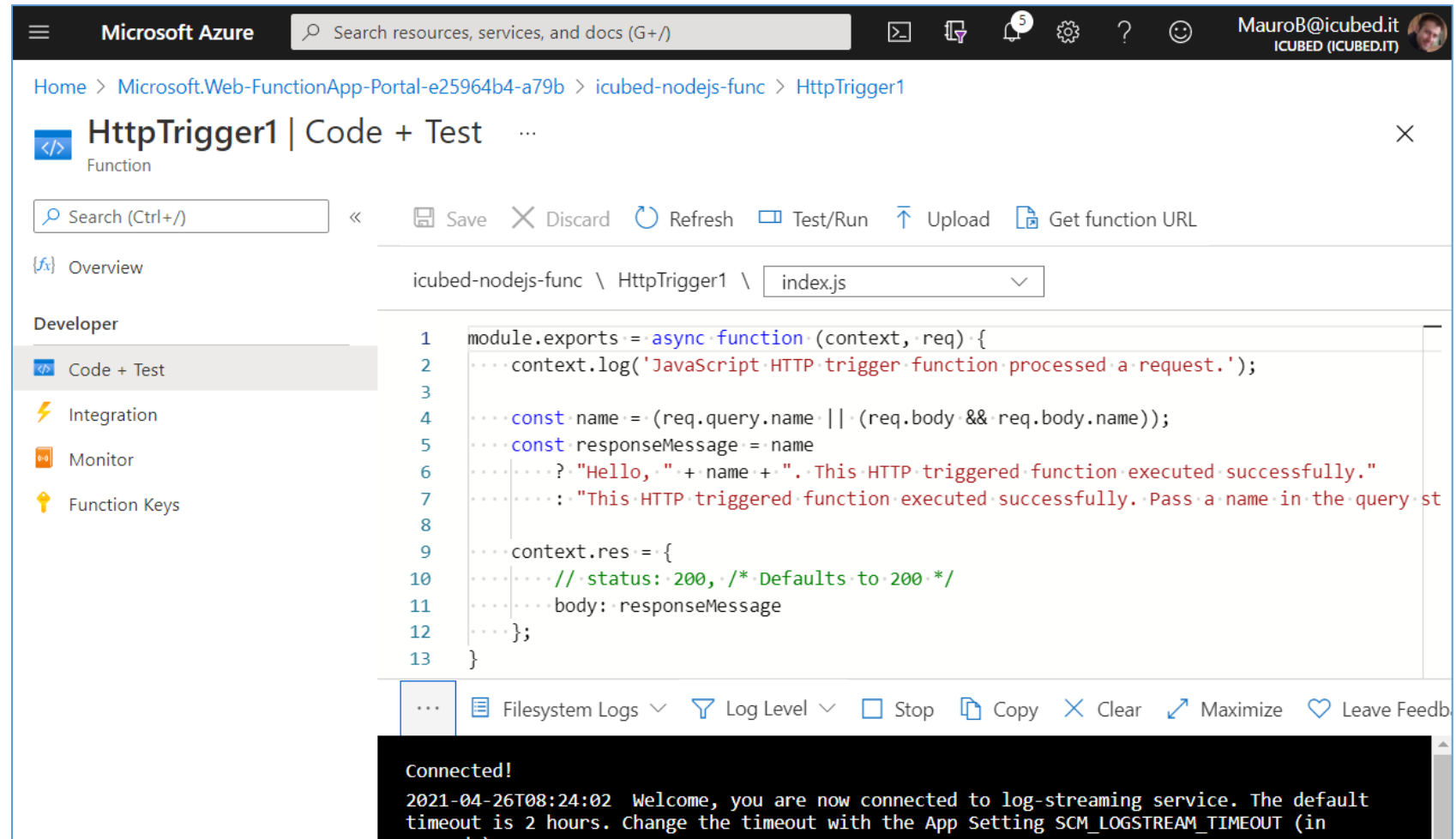
## Events (eventi)

- Comunicazione asincrona
- Permette di parallelizzare il lavoro da completare eseguendo un «broadcast» su diversi subscribers



# Azure: Azure Functions

Anche in Azure le Azure Functions, che sono l'espressione del server-less computing, fanno uso di NodeJs per la sua immediatezza



The screenshot displays the Microsoft Azure portal interface for a function named 'HttpTrigger1'. The breadcrumb navigation shows the path: Home > Microsoft.Web-FunctionApp-Portal-e25964b4-a79b > icubed-nodejs-func > HttpTrigger1. The main title is 'HttpTrigger1 | Code + Test'. Below the title, there's a search bar and a set of action buttons: Save, Discard, Refresh, Test/Run, Upload, and Get function URL. The left sidebar shows the 'Developer' section with options: Overview, Code + Test (selected), Integration, Monitor, and Function Keys. The main area shows the code for 'index.js' in the 'icubed-nodejs-func \ HttpTrigger1 \ ' directory. The code is as follows:

```
1 module.exports = async function (context, req) {
2   ...context.log('JavaScript HTTP trigger function processed a request.');
```

```
3
4   ...const name = (req.query.name || (req.body && req.body.name));
5   ...const responseMessage = name
6   ...? "Hello, " + name + ". This HTTP triggered function executed successfully."
7   ...: "This HTTP triggered function executed successfully. Pass a name in the query st
8
9   ...context.res = {
10   ...// status: 200, /* Defaults to 200 */
11   ...body: responseMessage
12   ...};
13 }
```

At the bottom, there's a status bar with buttons: Filesystem Logs, Log Level, Stop, Copy, Clear, Maximize, and Leave Feedback. A black notification box at the bottom right says: 'Connected! 2021-04-26T08:24:02 Welcome, you are now connected to log-streaming service. The default timeout is 2 hours. Change the timeout with the App Setting SCM\_LOGSTREAM\_TIMEOUT (in seconds)'.

# I sistemi informatici come motori del business

Da 30 anni a questa parte il mondo IT ha sempre permesso al business di funzionare.

E' innegabile che un'azienda non possa fare a meno di un sistema informatico, per quanto piccola possa essere la sua esigenza.

E il processo di ALM è sempre stato questo:

- Raccolta dei requisiti
- Implementazione
- Deploy e messa in esercizio



# Deploy: le origini (1)

Negli anni 2000 (e in alcuni casi ancora adesso) eravamo abituati a rilasciare le nostre applicazioni sull'hardware, nudo e crudo.

Spesso e volentieri una macchina di Produzione con «n» applicazioni a bordo, in alcuni casi ambienti separati e ridondati.

Per ogni nuova applicazione doveva essere fatto un approvvigionamento di hardware: uno sforzo considerevole in termini di CAPEX e OPEX di una azienda.



# Deploy: le origini (2)

Oltre a costi economici da sostenere, nasce una problematica importante: quanto potente deve essere l'hardware per sostenere l'applicazione che sarà messa sopra?

E' una risposta a cui nessuno sa rispondere – nemmeno oggi – quindi l'IT fa quello che è più ragionevole: dotarsi di un hardware estremamente performante per non rischiare poi di incorrere in problemi prestazionali.

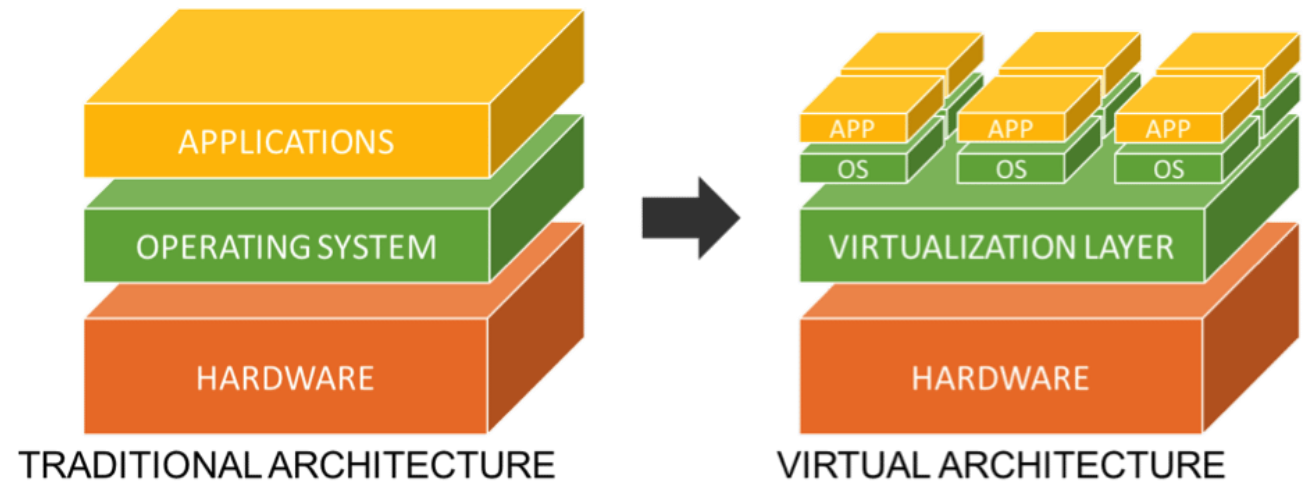
Nel consegue un aumento ancora maggiore del capitale investito, **senza certezza di ritorno**.

Nella maggior parte dei casi i server arrivavano ad utilizzare il 10/20% della loro effettiva potenza!



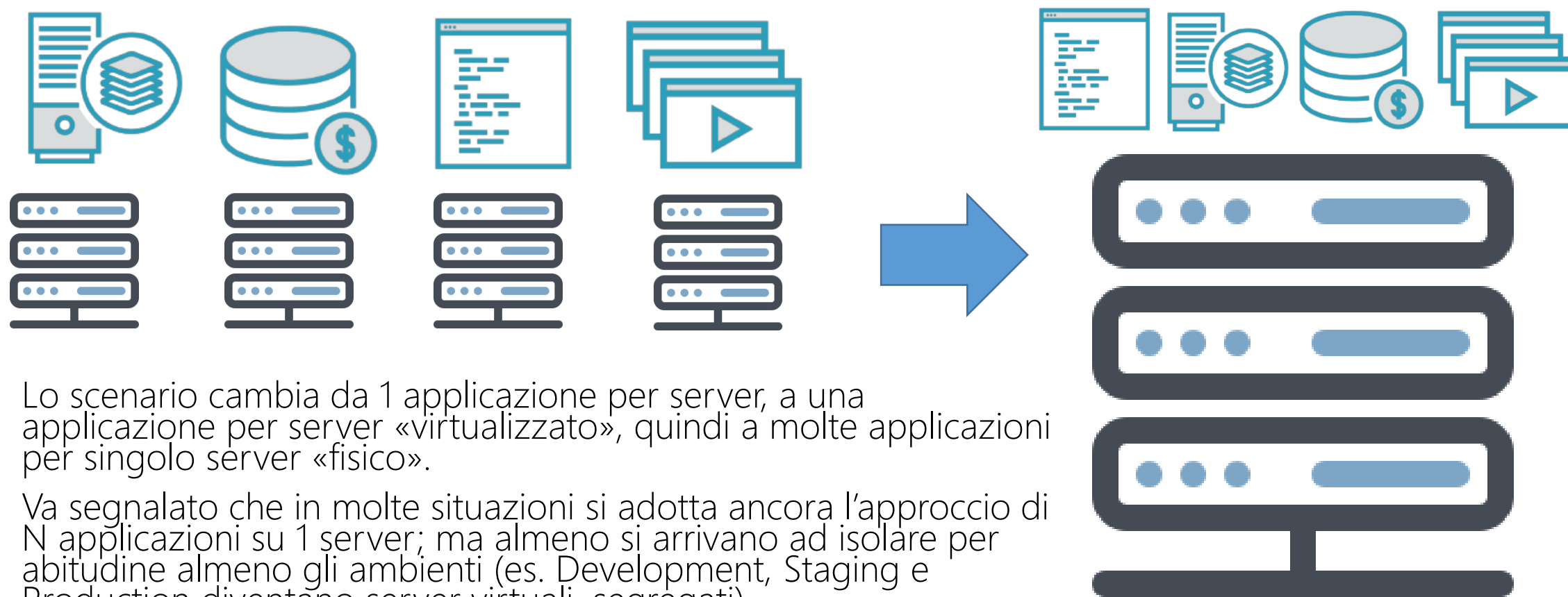
# Deploy: l'avvento della virtualizzazione (1)

L'introduzione dei sistemi di virtualizzazione (VMWare su tutti) sposta la problematica. Finalmente il team di operations poteva tranquillamente giustificare l'acquisto di hardware ad alte prestazioni, con la consapevolezza che poche risorse sarebbero andate «sprecate».





# Deploy: l'avvento della virtualizzazione (2)



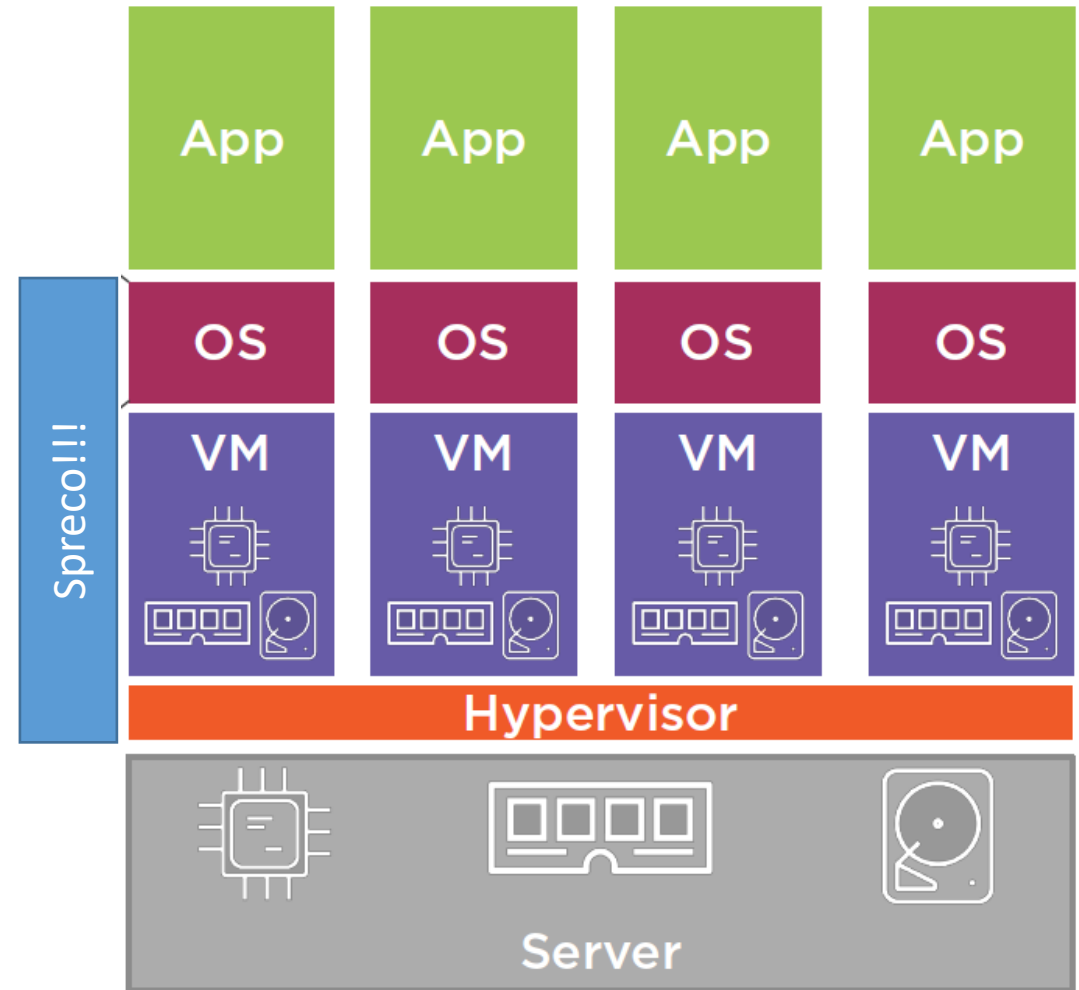
Lo scenario cambia da 1 applicazione per server, a una applicazione per server «virtualizzato», quindi a molte applicazioni per singolo server «fisico».

Va segnalato che in molte situazioni si adotta ancora l'approccio di N applicazioni su 1 server; ma almeno si arrivano ad isolare per abitudine almeno gli ambienti (es. Development, Staging e Production diventano server virtuali, segregati)

# Deploy: il problema della virtualizzazione

Nonostante la virtualizzazione sia stato un enorme passo in avanti nell'ottimizzazione del consumo delle risorse hardware, non rappresenta il punto di arrivo:

- La virtualizzazione dell'hardware richiede risorse
- I sistemi operativi delle varie macchine richiedono risorse
- E' necessario attendere un tempo di «boot» considerevole per avere l'ambiente pronto per l'uso





# Container 101: cosa è un container?

I «containers» offrono un meccanismo di pacchettizzazione logico in cui le applicazioni possono essere astratte dall'ambiente in cui vengono effettivamente eseguite.

Questo disaccoppiamento consente di distribuire facilmente e in modo coerente le applicazioni basate su container, indipendentemente dal fatto che l'ambiente di destinazione sia un data center privato, il cloud pubblico o persino il laptop personale di uno sviluppatore.

La containerizzazione fornisce il concetto di «Separation of Concerns»:

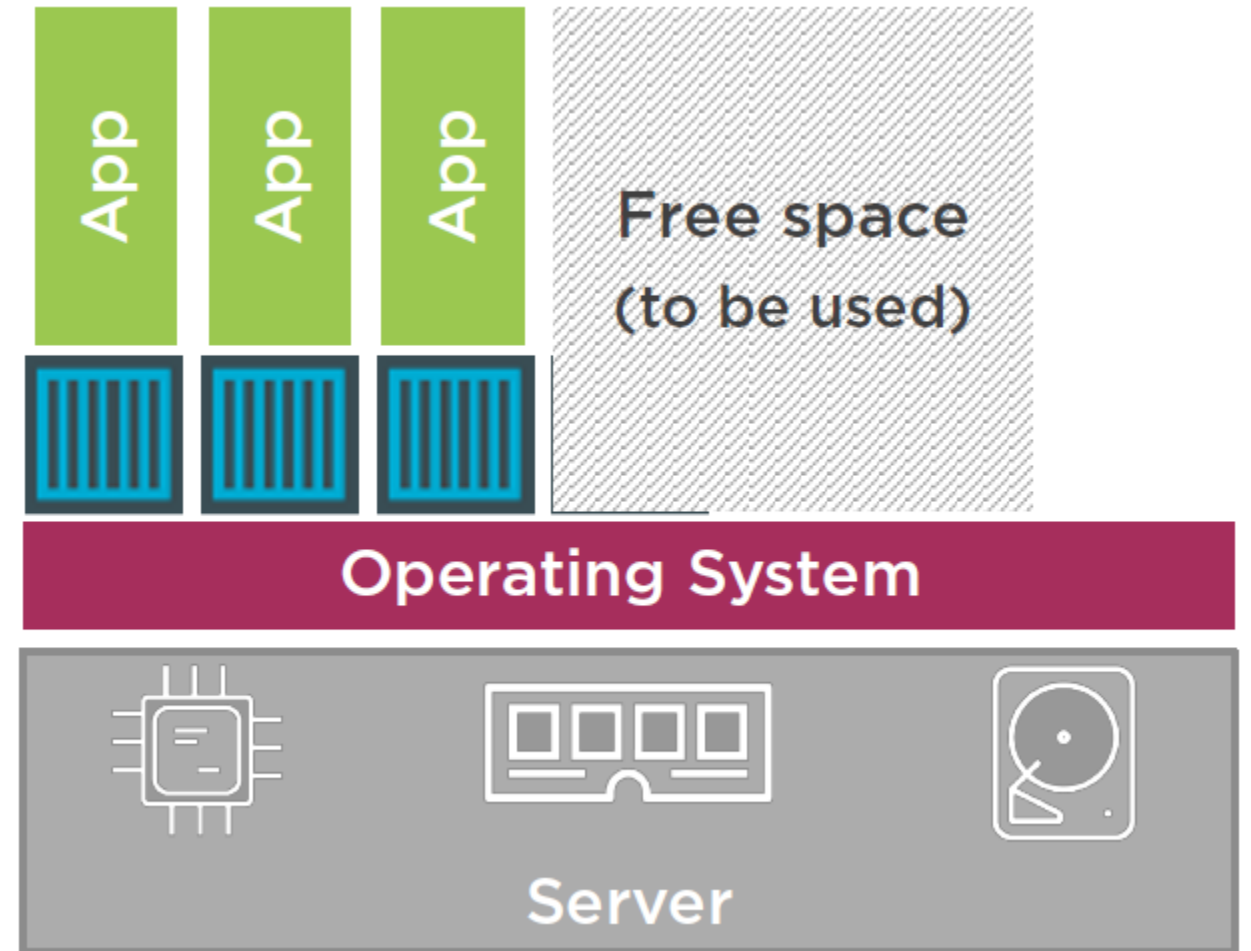
- gli sviluppatori si concentrano sulla logica dell'applicazione e sulle dipendenze
- il team di operation può concentrarsi sulla distribuzione e la gestione, senza preoccuparsi dei dettagli dell'applicazione: versioni software specifiche, o configurazioni particolari dell'app.

(sorgente: Google Cloud)

# Container: un approccio più ottimizzato

Una architettura basata su container, invece che virtualizzazione, ottimizza le risorse hardware a disposizione.

- I container fanno uso delle risorse fisiche presenti sul computer host
- Non hanno bisogno di un sistema operativo dedicato ed isolato
- L'isolamento è «logico» ma la condivisione è alla base di tutto
- Considerato che non è necessario «inizializzare un ambiente», il tempo di boot di un container è nettamente inferiore

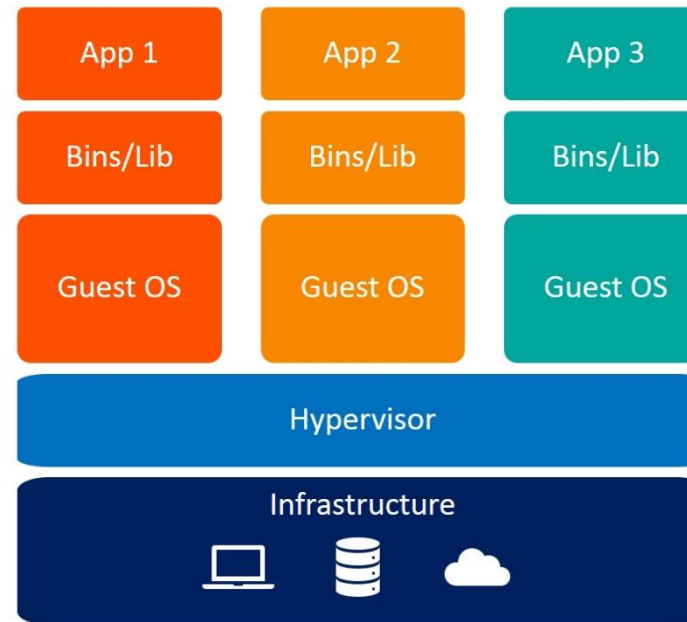


# Container e machine virtuali

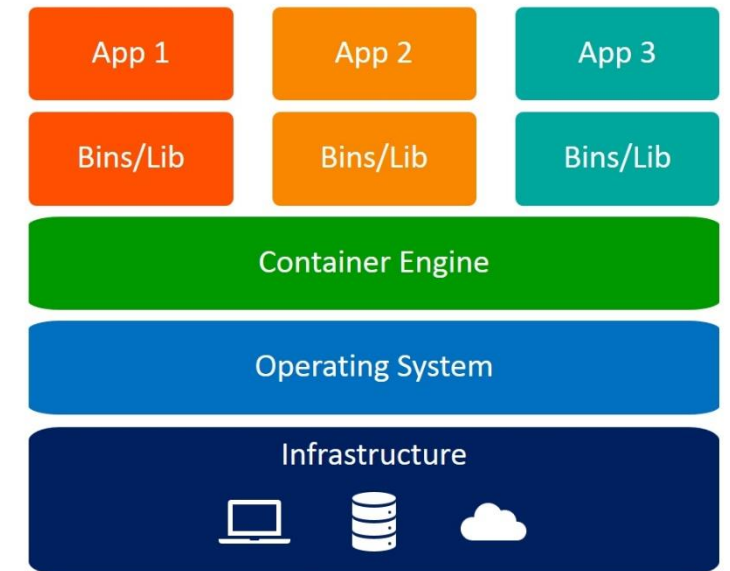
Passare da una architettura basata su macchine virtuali ad una architettura a container cambia drammaticamente le cose:

- Minor spreco di risorse
- Maggiore velocità
- Scalabilità più elastica
- Minore complessità di deploy degli applicativi
- Maggiore isolamento

In pratica, vale la frase: "Works on my machine, works everywhere..."



Virtual Machines



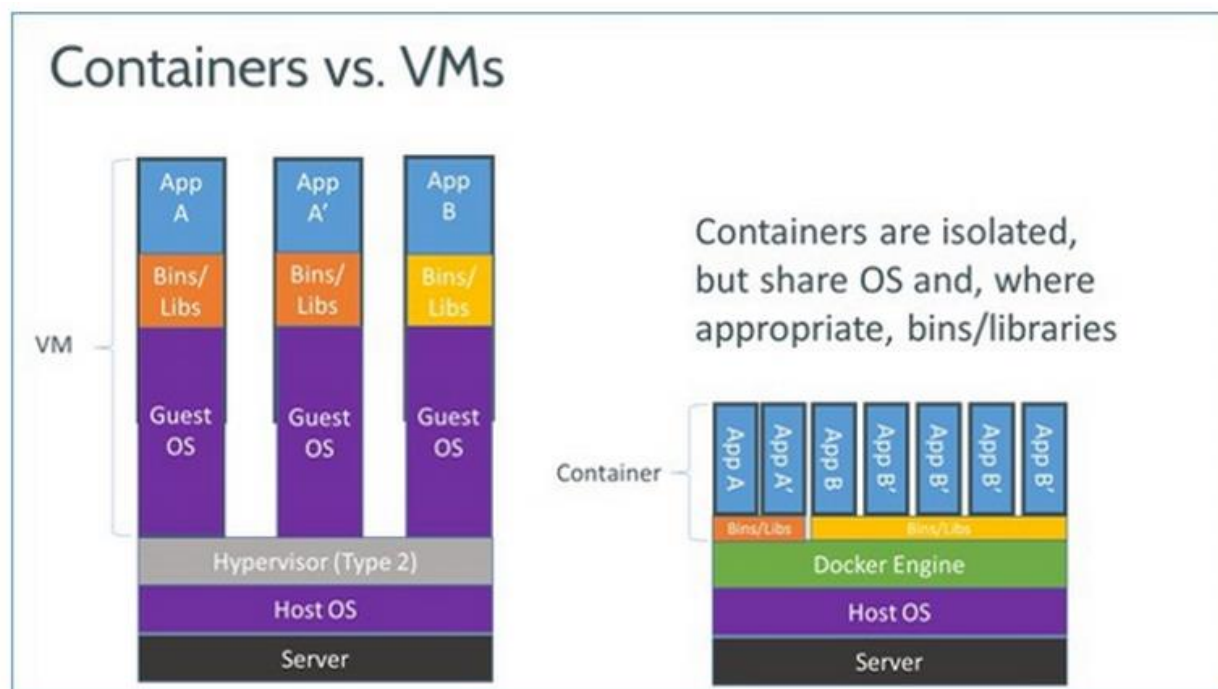
Containers

# Macchine virtuali vs Containers

Il quantitativo di risorse utilizzate in un approccio tradizionale come quello delle macchine virtuali è molto elevato.

Con l'adozione di container ogni singolo "box" è auto-consistente e comprensivo non solo del modulo applicativo, ma di tutti i requisiti tecnici (frameworks, drivers, servizi) necessari al suo corretto funzionamento.

Un orchestratore sta alla base di tutto il processo e si limita a monitorare ogni container, di fatto ignorando il suo contenuto.



# Docker: un po' di storia...

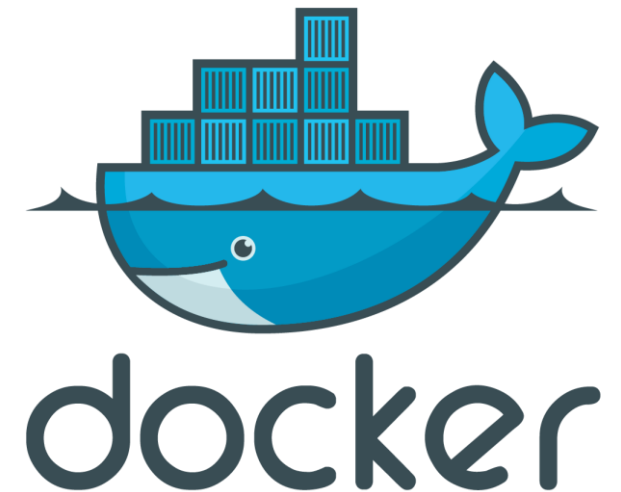
Docker Inc è l'azienda americana che ha portato alla massa la tecnologia dei container.

Inizialmente conosciuta con il nome di «DotCloud», il suo business era quello di vendere Cloud services, poggiati su AWS.

Ripresero – per uso interno – la tecnologia dei container, presente in Linux da anni, e la resero semplice ed accessibile.

Con il tempo cambiò il business target: si spostarono dal vendere servizi cloud, a diffondere e supportare la tecnologia dei container a livello globale.

L'obiettivo era creare una «demand» sul mondo dei container, e offrire consulenza per la loro adozione.



# Docker: getting started

Per iniziare ad apprendere la tecnologia dei container, il miglior modo è la pratica:

- Docker Desktop
  - Per Windows, supporta sia container Linux, che container Windows
  - Per Mac, supporta solo container Linux
  - Supporto integrato per Docker "Swarm Mode" e per Kubernetes
- Docker Native
  - Per Linux (Ubuntu, Mint, RedHat, CentOS, ecc)
- Play with Docker
  - Servizio online (browser-based) offerto da Docker Inc. e completamente gratuito

Qualunque scelta permette una esperienza tecnologica **identica** a quella di un ambiente di Produzione, con tutte le funzionalità disponibili.

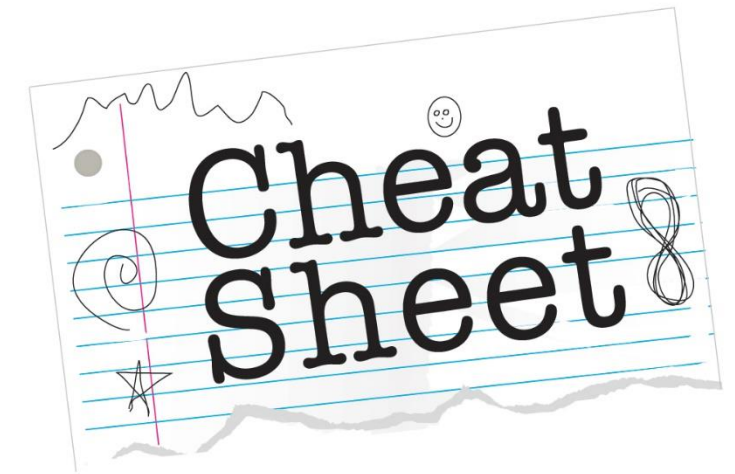




# Docker: i comandi base

Molti sono i comandi disponibili per la gestione dei containers. A seguito è riportato un breve cheat-scheet:

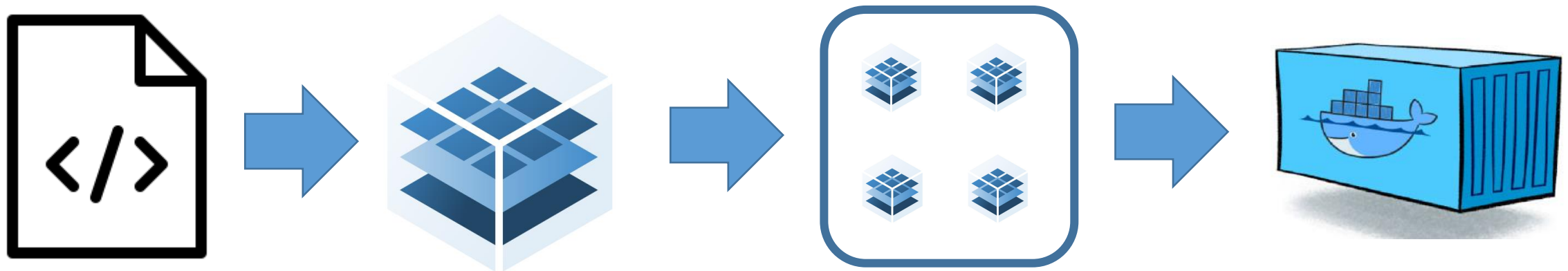
- **docker container start [nome-container]**
  - Avvio di un container esistente
- **docker container stop [nome-container]**
  - Arresto di un container esistente
- **docker container rm [nome-container]**
  - Rimozione di un container (--force se è attivo)
- **docker container ls -a**
  - Lista di tutti i container (-a è «tutti», altrimenti solo attivi)
- **docker container run -n [nome] -d [nome-immagine]**
  - Creazione container in «detached» mode su immagine
- **docker container run -n [nome] -it [nome-immagine] sh**
  - Creazione container «interattivo» con shell nel container
  - «sh» è il processo che deve essere lanciato all'avvio del container (qui la shell di bash)
  - Per uscire dal container senza terminarlo (CTRL + P + Q)



# Docker: overview demo

Per permettere una compresione della “big picture” del mondo dei containers, l’ideale è vedere in azione l’intero meccanismo, il processo “end-to-end”.

- Si parte dal codice sorgente di una applicazione esistente
- Passiamo alla creazione di una image Docker dell’applicazione
- Eseguiamo il “push” dell’immagine in un registry pubblico che ospita immagini
- Creiamo il container usando l’image create in precedenza, ricavata dal registry



# © 2022 iCubed Srl

La diffusione di questo materiale per scopi differenti da quelli per cui se ne è venuti in possesso è vietata

iCubed s.r.l.

Piazza Duca D'Aosta, 12 20124 MILANO

Phone: +39 02 57501057

P.IVA 07284390965

