Cloud Green Computing

Stefano Piccoli

22 maggio 2022

Indice

Parte I

Cloud Computing

Capitolo 1

IaaS (Infrastructure as a Service)

1.1 Virtualizzazione

La **virtualizzazione** rende possibile al sistema operativo di un server di eseguire su uno **strato virtuale** (**Hypervisor**).
Questo permette di eseguire molteplici **macchine virtuali**, ognuna con il proprio sistema operativo, sullo stesso server fisico.

1.2 Hypervisor

L'**hypervisor** crea lo strato di **virtualizzazione** che rende la virtualizzazione server possibile e contiene la **Virtual Machine Manager (VMM)**.

Tipologie

- **Type 1**: caricata direttamente sull'hardware, può eseguire più virtual server, usato per data center o server
    - Hyper-v
    - ESX/ESXi
    - XenServer
- **Type 2**: caricata in un sistema operativo eseguito sull'hardware, greater overhead, usato per desktop e laptop
    - Workstation
    - Virtual Server

- Fusion

1.3 Amazon Elastic Compute Cloud 2 (EC2)

- Mette a disposizione server virtuali (**istanze**) in modo semplice, veloce ed economico
- Scelta tipo istanza e template da utilizzare (Windows/Linux) e numero istanze con AWS management console (o librerie SDK)
- **Opzioni di pagamento**: on demand, istanze riservate, istanze spot
- **Sicurezza** (Virtual Private Cloud - VPC
- **Storage persistente**: Amazon Elastic Block Store (EBS)
- **Autoscaling**

1.4 Amazon Simple Storage Service (S3)

- Fornisce uno **storage sicuro e facile** da usare
- Diverse **classi di memorizzazione** (standard / standard infrequent access / glacier)
- **Controllo** configurabile di **accesso ai dati**

1.5 Amazon Elastic Block Store (EBS)

- Blocco persistente di archiviazione di volumi di memoria usato con le istanze di Amazon EC2
- Ogni volume di Amazon EBS viene automaticamente replicato senza la sua Availability Zone in modo da offrire alta disponibilità e durata.

1.6 Dropbox exodus

- I primi 8 anni della sua vita archiviava miliardi di file su Amazon S3
- Tra il 2014 e 2016 ha costruito la propria rete di server ideata dai propri ingegneri per spostare i dati

1.6.1 L'esodo

- Hardware proprietario che archivierà petabyte di dati
- Nuovo codice ("Magic Pocket")
- Installare 50 rack di hardware al giorno
- Completare lo spostamento prima della scadenza del contratto con Amazon per evitare un rinnovo

1.6.2 Conclusioni

Dropbox è riuscita a completare lo spostamento con successo entro i tempi previsti.

Capitolo 2

Container

I **containers** sono un meccanismo di virtualizzazione differente dalle Virtual Machines poiché permetto di avere più istanze **isolate** e **volatili** che scompaiono quando interrotte.
I containers sono **leggeri**, **veloci**, più **semplici da buildare** ma **meno sicuri** delle Virtual Machines.

2.1 Docker

Docker è un'azienda che ha realizzato una piattaforma che permette di **eseguire una applicazione in ambiente "isolato"**.
Docker sfrutta la **virtualizzazione basata sui container** per eseguire in maniera isolata diverse **GUEST INSTANCES** sullo stesso sistema operativo.

2.1.1 Caratteristiche

- **Portabilità**: il software può essere impacchettato in **images**, file read only che può essere mandato in esecuzione da docker e creare quindi il container
- Possono avere più istanze separate degli spazi utente (**containers**)
- **Interfaccia** utente **semplificata**
- **Svantaggio**: sono meno isolati delle macchine virtuali, **condividono le risorse di sistema**

2.1.2 Componenti

- **Docker Engine**: permette di creare e mandare in esecuzione container
- **Docker Hub**: repository enorme che contiene molte immagini di container
- **Docker Swarm Mode**: permette di eseguire un container su più docker host e divide gli swarm node in manager e worker, permettendo una **gestione dichiarativa** della nostra **applicazione**
- **Images**: **template di sola lettura** usati per creare container, registrate in registry
    - **Stratificazione**: ogni strato può essere a sua volta una immagine
- **Registry**: **strutture di repository** che contengono insiemi di immagini per diverse versioni del sw

2.1.3 Comandi

- **PULL**: tiro un'immagine dal registry alla macchina
- **RUN**: viene creato il container dell'immagine
- **COMMIT**: salvare una nuova immagine
- **PUSH**: caricare una immagine nel registry
- **BUILD**: si crea un dockerfile che permette di creare un'immagine automaticamente

2.1.4 Swarm mode

- I nodi possono agire da **managers**, delegando tasks, o **workers**, eseguendo task assegnati.
- È possibile definire lo **stato dei vari servizi** nello stack dell'applicazione, incluso il numero di **task da eseguire in ogni servizio**
- **Swarm manager**:
    - assegna ad ogni servizio nello swarm un unico DNS name
    - bilancia il carico dei container in esecuzione
    - monitora lo stato del cluster e lo allinea con quello desiderato

Capitolo 3

PaaS (Platform as a Service)

Servizio che fornisce hardware e software per lo sviluppo di applicazioni. L'utente deve fornire solo l'applicazione e i dati

Vantaggi

- Facilità di gestione e modifica dell'applicazione
- Facilità nell'adottare nuove tecnologie

Rischi

- Disponibilità del servizio: l'interruzione del servizio da parte del fornitore comporta un immediato disservizio
- Vendor lock-in: difficoltà di cambiare servizio da parte del cliente

3.1 Heroku

Heroku è una piattaforma cloud basata su **container** con servizi integrati e un potente ecosistema che permette il deployment e running di applicazioni.

3.1.1 Dynos

I **dynos** sono container Linux virtualizzati, Heroku trasforma l'applicazione utente in diversi **dynos**.

Vantaggi

- Scalabilità
- Evitare di gestire l'infrastruttura

Premium:

- **Scaling**
- **Autoscaling**: permette di inserire politiche per quando usare lo scaling

3.1.2 Buildtime

Per sviluppare una applicazione Heroku richiede:

- **Codice sorgente**
- **Lista di dipendenze**
- **Procfile**: file di testo che indica quale comando usare per far eseguire l'applicazione

Slug: Un insieme di codice sorgente, dipendenze, supporto per output, etc...
Stack: Sistema operativo Ubuntu

3.1.3 Runtime

Nel **runtime** si prende lo slug e lo stack e vengono creati i dynos, che rappresentano le istanze utente, il dyno manager fa partire i container con il comando specificato dall'utente.

3.1.4 Esempio

1. Applicazione riceve richiesta

3.1.5 Add-ons

Gli **add-ons** sono funzionalità fornite da Heroku che possono essere aggiunte facilmente all'applicazione.

3.2 Altri PaaS

- Microsoft Azure
- OpenShift

Capitolo 4

Modelli di Business

Un **business model** descrive il razionale di come una azienda **crea**, **consegna** e **acquisisce valore**.

- **Customer Segments**: il gruppo di persone o organizzazioni a cui il servizio mira di raggiungere
- **Valuer Propositions**: cosa rende speciale il servizio
- **Channels**: le modalità in cui la compagnia raggiunge il cliente
- **Customer Relationships**: tipo di relazione che la compagnia stabilisce col cliente
- **Revenue Streams**: il flusso di entrate che la compagnia genera da ogni segmento di clientela
- **Key Resources**: le risorse più importanti richieste per il modello di business
- **Key Activities**: le attività più importanti che la compagnia deve svolgere
- **Key Partners**: la rete di fornitori e partners per il business
- **Cost Structure**: i costi che si incontrano per operare nel modello di business

4.1 Business innovation

- **Resource-driven**: ha origine da **infrastrutture o partner gia esistenti** usate per espandere o trasformare il business model

- **Offer-driven**: crea nuova value proposition che influenza altri ambiti del business model
- **Custmer-driven**: basato sulle necessità del cliente, accesso facilitato o aumento di convenienza
- **Finance-driven**: guidata dal **revenue stream**, meccanismo di prezzi o riduzione dei cost structure

Parte II

Green Computing

Capitolo 5

Introduzione

ICT è una minaccia per la sostenibilità ambientale?

- La produzione di hardware produce inquinamento
- Gas serra
- e-waste

Green Computing Pratica ambientale per calcolare la sostenibilità della computazione

- Minimizzare consumo energetico
- Progettare soluzioni efficienti energeticamente
- Riduzione e-waste

5.1 Sprechi

- 1,800 kg di materiale grezzo per produrre un personal computer (Rinoceronte)
- ICT contribuisce al 9% di consumo elettrico in Europa
- ICT contribuisce al 4% di emissioni di carbonio in Europa
- Produce il 2% delle emissioni di CO2 globali (carburante aereo)
- È previsto ulteriore aumento nei prossimi anni (esponenzialmente nel networking 20,9%)

- 50 milioni di tonnellate all'anno di e-waste (770 milioni di lavatrici)
- Il traffico di e-waste ha un traffico monetario illegale maggiore del traffico di droga
- I data centre potranno arrivare al 28% della domanda energetica per ICT.

5.2 Datacenters

- 40% di consumo energetico per il raffreddamento
- PUE (Power Usage Effectiveness) $\frac{\text{Total Facility Power}}{\text{IT Equipment Power}}$ ma **non** misura la quantità di energia rinnovabile usata

5.3 Obsolescenza programmata

Caratteristica di un prodotto volontariamente ideato per avere un "più breve" ciclo di vita. La vita del dispositivo deve durare abbastanza da soddisfare il cliente e fargli desiderare un nuovo dispositivo.

5.3.1 Cartello Phoebus

I più grandi produttori di lampadine si riunirono per accordarsi di produrre lampadine che durassero 1000 ore invece di 2500 ore, anche se erano tutti in grado di produrre lampadine più longeve.

5.3.2 Obsolescenza percepita

Il cliente è convinto di aver bisogno di un prodotto aggiornato, anche se effettivamente quello attuale è perfettamente funzionante.

5.4 Progettare dispositivi efficienti energeticamente

- Minimizzare processi di comunicazione, interazioni GUI-user
- Eliminare controlli non necessari
- Scegliere linguaggi di programmazione adatti

- Minimizzare movimenti di dati, usare efficacemente il caching

Capitolo 6

Faas (Function As A Service)

6.1 AWS Lambda

- Eseguire codice senza fornire o gestire server, senza preoccuparsi dell'amministrazione
- Si occupa di scalare ed eseguire e patchare il codice
- È possibile impostare il codice in modo da attivarsi attraverso altri servizi AWS
- Pagamento solo per il tempo di calcolo

6.2 Tipi di FaaS:

Commerciali

- AWS Lambda
- Google Cloud Functions
- MS Azure Functions

Open Source

- Apache Openwhisk
- Fission
- Fn
- Knative
- Kubeless
- Nuclio
- OpenFaaS

6.3 Due viste

Business view

- Licensing
- Installation
- Source code
- Interface
- Community
- Documentation

Technical view

- Development
- Version management
- Event sources
- Function orchestration
- Testing and debugging
- Observability
- Application delivery
- Code reuse
- Access management

6.4 Business View

6.4.1 Business View: License

- Opens Source: License permissive con Apache 2.0
- Commerciali: License proprietarie
- MS Azure Functions: anche progetti open sources

	License	Type
Apache Openwhisk	Apache 2.0	Permissive
AWS Lambda	AWS service terms	Proprietary
Fission	Apache 2.0	Permissive
Fn	Apache 2.0	Permissive
Google Cloud Functions	Google Cloud platform terms	Proprietary
Knative	Apache 2.0	Permissive
Kubeless	Apache 2.0	Permissive
MS Azure Functions	SLA for Azure Functions	Proprietary
Nuclio	Apache 2.0	Permissive
OpenFaaS	MIT	Permissive

6.4.2 Business View: Installazione

- Commerciali: Solo Azure ha alcune parti installabili in locale
- Piattaforme installabili supportano piu host, Kubernetes è il più supportato

	Type	Target hosts
Apache Openwhisk	Installable	Docker, Kubernetes, Linux, MacOS, Mesos, Windows
AWS Lambda	as-a-service	n/a
Fission	Installable	Kubernetes
Fn	Installable	Docker, Linux, MacOS, Unix, Windows
Google Cloud Functions	as-a-service	n/a
Knative	Installable	Kubernetes
Kubeless	Installable	Kubernetes, Linux, MacOS, Windows
MS Azure Functions	as-a-service, installable	Linux, Kubernetes, MacOS, Windows
Nuclio	as-a-service, installable	Kubernetes
OpenFaaS	Installable	Docker[a], faasd, Kubernetes, OpenShift

6.4.3 Business View: Source Code

- Commerciali: MS Azure Function è l'unica parzialmente open source
- Open Source: Hostate su GitHub e implementate maggiormente in Go

	Availability	Open source repository	Programming language
Apache Openwhisk	Open source	GitHub	Scala
AWS Lambda	Closed source	n/a	n/a
Fission	Open source	GitHub	Go
Fn	Open source	GitHub	Go
Google Cloud Functions	Closed source	n/a	n/a
Knative	Open source	GitHub	Go
Kubeless	Open source	GitHub	Go
MS Azure Functions	Open source[a]	GitHub	C#
Nuclio	Open source	GitHub	Go
OpenFaaS	Open source	GitHub	Go

6.4.4 Business view: Interface

- Tutte le piattaforme forniscono CLI
- API e GUI non sempre fornite

		Apache Openwhisk	AWS Lambda	Fission	Fn	Google Cloud Functions	Knative	Kubeless	MS Azure Functions	Nuclio	OpenFaaS
Type	cli	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	api	✓	✓	✓	✓	✓	✓	×	✓	×	✓
	gui	×	✓	×	✓	✓	×	×	✓	✓	✓
App. Man.	create	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	retrieve	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	update	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	delete	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Plat. Adm.	deployment	✓	×	✓	✓	×	✓	✓	×	✓	✓
	configuration	✓	×	✓	✓	×	✓	✓	×	✓	×
	enactment	✓	×	✓	✓	×	✓	✓	×	✓	✓
	termination	×[a]	×	×[a]	✓	×	×[a]	×[a]	×	×[a]	×[a]
	undeployment	×[a]	×	×[a]	×[a]	×	×[a]	×[a]	×	×[a]	×[a]

[a]Termination/undeployment can be achieved by stopping/uninstalling the platform instance with host commands.

6.4.5 Business view: Community

- OpenFaaS, Apache Openwhisk e Knative hanno la valutazione più alta su GitHub in stelle, contributors e commits rispettivamente
- Stackoverflow mostra un drastica differenza tra commerciali e open sources

		Apache Openwhisk	AWS Lambda	Fission	Fn	Google Cloud Functions	Knative	Kubeless	MS Azure Functions	Nuclio	OpenFaaS
GitHub	Stars	4.8K	n/a	5.2K	4.6K	n/a	3K[a]	5.8K	n/a	3.3K	17.9K
	Forks	932	n/a	487	349	n/a	637[a]	591	n/a	332	1.5K
	Issues	274	n/a	215	125	n/a	223[a]	164	n/a	51	62
	Commits	2.8K	n/a	1.2K	3.4K	n/a	4.7K[a]	1K	n/a	1.4K	1.9K
	Contributors	180	n/a	104	86	n/a	185[a]	89	n/a	55	147
SO	Questions	198	16.8K	7	25	10.1K	71	8	7.2K	3	29

[a]Values for the function hosting component of Knative, i.e., Knative Serving.

6.4.6 Business view: Documentation

- Tutte le piattaforme forniscono deployment dell'applicazione e documentazione d'uso della piattaforma
- Platform development e architettura non sono sempre documentate in piattaforme open sources

		Apache Openwhisk	AWS Lambda	Fission	Fn	Google Cloud Functions	Knative	Kubeless	MS Azure Functions	Nuclio	OpenFaaS
App.	Development	✓	✓	✓	✓	✓	×	✓	✓	×	×
	Deployment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Platform	Usage	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Development	✓	×	✓	×[a]	×	✓	✓	×	×[a]	✓
	Deployment	✓	×	✓	✓	×	✓	✓	✓	✓	✓
	Architecture	✓	×	✓	✓	×	×	✓	×	✓	✓

[a]Only providing guidelines/code of conduct for contributing to the project.

6.5 Technical View

6.5.1 Technical view: Development

- Java, Node.js e Python sono i piu supportati runtimes, Docker è inoltre popolare per personalizzazione runtime.
- IDEs e text editor sono principalmente offerti dalle piattaforme **commerciali**.

Classification of considered FaaS platforms, based on the *Development* category in the *technical* view of our classification framework. D and E are used to denote that quotas are set for *deployment package size* and *execution time*, respectively. The abbreviation "n/s" stays for "not specified", meaning that no related information is in the documentation.

	IDEs and Text Editors	Client Libraries	Quotas
Apache Openwhisk	Visual Studio Code[a], Xcode[a]	Go, Node.js	E[b]
AWS Lambda	AWS Cloud9, Eclipse, Toolkit for JetBrains, Visual Studio, Visual Studio Code	Go, Java, MS .NET, Node.js, Python, Ruby, C++	D, E
Fission	n/s	n/s	n/s
Fn	n/s	Go	n/s
Google Cloud Functions	n/s	Dart, Go, Java, MS .NET, Node.js, Python, Ruby	D, E
Knative	n/s	n/s	n/s
Kubeless	Visual Studio Code	n/s	n/s
MS Azure Functions	Visual Studio, Visual Studio Code	Java, MS .NET, Python	D, E[b]
Nuclio	Jupyter Notebooks	Go, Java, MS .NET, Python	n/s
OpenFaaS	n/s	n/s	n/s

[a]Deprecated/No longer maintained.
[b]Bounded by default, but can be configured to unset quota.

6.5.2 Technical view: Versioning

- Open source: versioning implicito
- Commerciali: versioning dedicato

Classification of considered FaaS platforms, based on the *Version Management* category in the *technical* view of our classification framework. D and I are used to denote the possible values *dedicated mechanisms* and *implicit versioning*, respectively. The abbreviation "n/s" stays for "not specified", meaning that a platform does not explicitly mention the versioning of serverless applications.

	Application versions	Function versions
Apache Openwhisk	I	I
AWS Lambda	D	D
Fission	I	I
Fn	I	D
Google Cloud Functions	I	I
Knative	n/s	D
Kubeless	n/s	I
MS Azure Functions	I	I
Nuclio	n/s	D
OpenFaaS	n/s	I

6.5.3 Technical view: Function Orchestration

- Tutte le piattaforme supportano invocazioni di funzioni HTTP-based **sincrone**, le **asincrone** invece da poche piattaforme
- Piu della metà delle piattaforme open source non supporta data store event sources
- Scheduler, stream processing platforms e messaging sono supportate dalla maggior parte delle piattaforme
- Più di metà delle piattaforme permette di integrare sorgenti di eventi custom
- Più della metà dei FaaS supporta function orchestration usando altre DSLs o orchestrating functions.

Classification of considered FaaS platforms, based on the *Function Orchestration* category in the *technical* view of our classification framework. C denotes *custom DSL*, and O denotes *orchestrating function*, with the list of supported programming languages for developing orchestrating functions given in square braces. The abbreviations "n/s" and "n/a" stay for "not specified" and "not applicable", respectively.

	Orchestrator	Workflow definition	Control flow described	Quotas
Apache Openwhisk	Openwhisk composer	O [JavaScript, Python]	✓	Execution time
AWS Lambda	AWS step functions	C	✓	Execution time, I/O size
Fission	Fission workflows	C	✓	n/s
Fn	Fn Flow	O [Java]	✓	n/s
Google Cloud Functions	n/s	n/a	n/a	n/a
Knative	Knative eventing	C	✓[a]	n/s
Kubeless	n/s	n/a	n/a	n/a
MS Azure Functions	Azure durable functions	O [C#, JavaScript]	✓	n/s
Nuclio	n/s	n/a	n/a	n/a
OpenFaaS	n/s	n/a	n/a	n/a

[a]Only sequence and parallel execution are supported.

6.5.4 Technical view: Testing & debugging

- La maggior parte delle piattaforme viste supporta testing funzionale e debug delle funzioni
- Commerciali: offrono operazioni più sofisticate
- Open Sources: test calls e log-based debugging

6.5.5 Technical view: Observability

- Commerciali: tool dedicati al monitoraggio e logging
- Open source: richiedono integrazione di tool di terze parti
- Più della metà dei FaaS supporta function orchestration usando altre DSLs o orchestrating functions.

	Monitoring	Logging	Tooling Integr.
Apache Openwhisk	Kamon, Prometheus, Datadog	Logback (slf4j)	n/s
AWS Lambda	AWS CloudWatch	AWS CloudTrail, CloudWatch	n/s
Fission	Istio + Prometheus	Fission Logger + InfluxDB, Istio + Jaeger	Using a service mesh
Fn	Prometheus, Zipkin, Jaeger	Docker container logs	Push-based
Google Cloud Functions	Google Cloud Operations	Google Cloud Operations	n/s
Knative	Prometheus + Grafana, Zipkin, Jaeger	ElasticSearch + Kibana, Google Cloud Operations	Push-based
Kubeless	Prometheus + Grafana	n/s	n/s
MS Azure Functions	Azure Application Insights	Azure Application Insights	n/s
Nuclio	Prometheus, Azure Application Insights	stdout, Azure Application Insights	Push-based, pull-based
OpenFaaS	OpenFaaS Gateway + Prometheus	Kubernetes cluster API, Swarm cluster API, Loki	Pull-based

6.5.6 Technical view: Application Delivery

- La maggior parte delle piattaforme segue un approccio dichiarativo al deployment automatico delle applicazioni
- Commerciali: supportano nativamente CI/CD tool
- Open sources: Solo OpenFaaS integra CI/CD, le altre no
- Più della metà dei FaaS supporta function orchestration usando altre DSLs o orchestrating functions.

Classification of considered FaaS platforms, based on the *Application Delivery* category in the *technical* view of our classification framework. P and T denote *Platform-native tooling* and *third party tooling*, respectively. The abbreviation "n/s" stays for "not specified", meaning that no related information is documented.

	Deployment automation	CI/CD
Apache Openwhisk	wskdeploy (P)	n/s
AWS Lambda	AWS Cloud Formation (P), AWS SAM (P)	AWS CodePipeline(P)
Fission	Fission CLI (P)	n/s
Fn	Fn CLI (P)	n/s
Google Cloud Functions	Google Cloud Deployment Manager	Cloud Build (P)
Knative Kubeless	Kubernetes (P)[a] Kubernetes (P)[a], Serverless Framework (T)	n/s n/s
MS Azure Functions	Azure Resource Manager (P)	Azure Pipelines (P), Azure App Service (P), Jenkins (T)
Nuclio	nuctl (P)[a]	n/s
OpenFaaS	faas-cli (P)	OpenFaaS Cloud (P), faas-cli (P), Circle CI (T), CodeFresh (T), Drone CI (T), GitLab CI/CD (T), Jenkins (T), Travis (T)

[a]Using Kubernetes specification with Custom Resource Definitions.

6.5.7 Technical view: Application Delivery

- La maggior parte delle piattaforme segue un approccio dichiarativo al deployment automatico delle applicazioni
- Commerciali: supportano nativamente CI/CD tool
- Open sources: Solo OpenFaaS integra CI/CD, le altre no
- Solo AWS Lambda e MS Azure Functions sono affiliati a un function marketplace

6.5.8 Technical view: Application Delivery

- La maggior parte delle piattaforme segue un approccio dichiarativo al deployment automatico delle applicazioni
- Commerciali: supportano nativamente CI/CD tool
- Open sources: Solo OpenFaaS integra CI/CD, le altre no

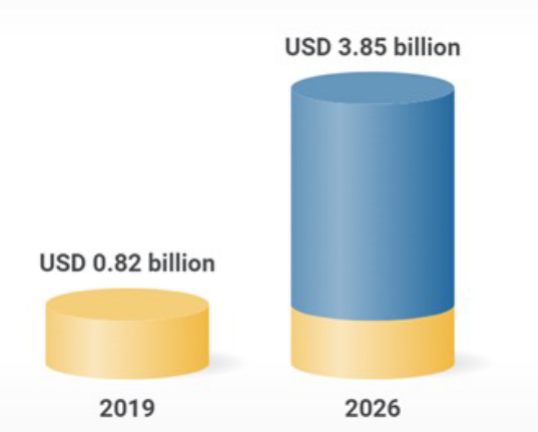
6.5.9 Technical view: Code Reuse

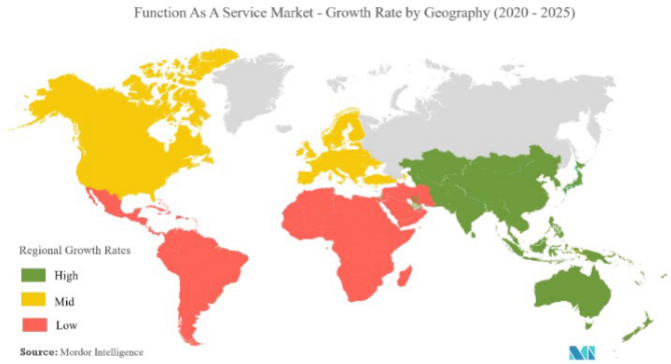
- Solo AWS Lambda e MS Azure Functions sono affiliati a un function marketplace

6.5.10 Technical view: Access Management

- Commerciali: supportano nativamente autenticazione e controllo di accesso alle risorse
- Open Source: usano funzioni offerte dal ambiente di hosting per garantire autenticazione e accesso alle risorse

6.6 FaaS Market





Capitolo 7

Software sostenibile

7.1 Nozioni di base

Efficienza:

$$\varepsilon = \frac{\text{num.calcoli}}{\text{elettricità}}$$

Efficienza $\neq$ Correttezza

Sostenibilità: Lo sviluppo sostenibile è lo sviluppo in grado di soddisfare i bisogni del presente senza compromettere la possibilità delle future generazioni di soddisfare i propri.

Impronta ecologica massima consentita secondo l'accordo? 2000 kg CO2 annui per persona.

Quanta CO2 produce un cittadino europeo ogni anno? 10000 kg.

Quanto CO2 produce l'invio di una mail? 20 g.

Quanta CO2 produce una ricerca sul web? 1 g.

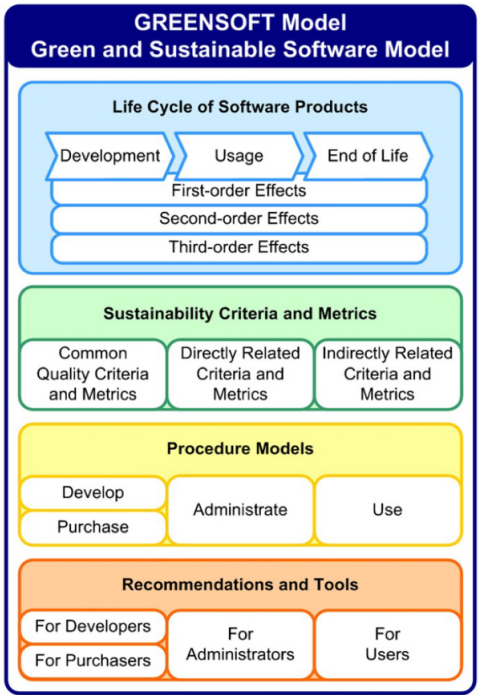
Ingegnere del software Persona che applica i principi dell'ingegneria del software per progettare, sviluppare, mantenere, testare e valutare il software informatico.

7.2 Triangolo di ferro

Gli ingegneri del software devono perseguire questi obiettivi contrastanti continuamente: **costo**, **tempo**, **qualità**.

7.3 Il modello Greensoft

Il modello **Greensoft** è un modello olistico per lo sviluppo, utilizzo e dismissione di software sostenibile.



7.4 Ruoli e collaborazione

- Diversi ruoli possono essere ricoperti da una stessa persona
- Ruoli ricoperti da persone diverse necessitano collaborazione tra queste
- **DevOps fonde i ruoli di sviluppatore, operatore e architetto**

7.5 Caso di studio

7.5.1 Architetto

Gli **architetti** sono coloro che determinano i requisiti funzionali e non-funzionali del sistema e che progettano le interazioni tra componenti.

- Possono includere la sostenibilità nei requisiti non-funzionali:
  - **Scalabilità**: possibilità di adattare il sistema alla domanda
  - **Performance**: tempo di risposta nel Service Level Agreement
- Come?
  - scomponendo i servizi in **unità scalabili** (per risorsa: calcolo, rete, storage)
  - rendere possibile un **bin packing** ottimo sull'utilizzo delle risorse di calcolo

7.5.2 Bin packing

- Problema difficile: **NP-hard**
- Corrisponde a minimizzare il numero di server per far girare i servizi che compongono il sistema
- L'architetto può disegnare componenti più piccole che consentono di risolvere il problema del bin packing evitando frammentazione
- La soluzione del problema verrà poi delegata a livelli più bassi gestiti da altri ruoli

7.5.3 CDN 101

Mantenere i dati in punti di presenza locali:

- Minor distanza per raggiungere i dispositivi dei clienti
- Minore utilizzo di banda
- Miglior utilizzo della rete, mantenendo contenuti popolari in cache

7.5.4 Scelte implementative

- Implementare servizi che richiedano un quantitativo ragionevole di risorse anche per favorire **bin packing**
- Disaccoppiare implementazione da infrastruttura in modo da facilitare la migrazione a Cloud provider differenti

7.5.5 Scegliere un linguaggio

Trovare un bilanciamento tra **energia consumata**, **tempo di esecuzione** e **memoria utilizzata**.

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

7.5.6 Sviluppatore

- Può cercare di ottimizzare l'efficienza del software
- Disaccoppiare applicazione da dettagli sull'infrastruttura
- Ridurre lo spreco di risorse durante lo sviluppo

7.5.7 Operatore

Coloro che gestiscono l'infrastruttura e garantiscono la disponibilità dell'applicazione

- Offrono risorse infrastrutturali adeguate a garantire le performance desiderate
- Utilizzano al meglio le risorse per contenere i costi e favorire la sostenibilità
- Garantiscono backup e ripristino
- Implementano logs e monitoraggio
- Ottimizzare l'uso delle risorse
- Fare scelte infrastrutturali che garantiscano performance e facilitino il bin packing

7.6 Quadro normativo

- Le leggi e i regolamenti possono favorire una transizione sostenibile
- Le emissioni dovute al ciclo di vita di un software sono una esternalità negativa
- E' un costo indiretto che viene pagato da terzi non coinvolti nel ciclo di vita di quel software
- Le leggi possono tassare le esternalità negative
- Le leggi possono obbligare alla trasparenza per creare consapevolezza tra i clienti

7.7 EU vs the World

L'Unione Europea ha definito una direttiva su «Corporate sustainability reporting», all'interno del Green Deal europeo.

- Obbliga le aziende a relazionare sulla sostenibilità ambientale, sarà in vigore dal 2023.

Capitolo 8

Microservizi

Un **microservizio** ha l'obiettivo di accorciare il *lead time* per nuove funzionalità e aggiornamenti.

- Accelera il rebuild e il redeployment
- Riduce corde fra silos funzionali (vari gruppi di un'azienda)
- Riuscire a scalare efficacemente

Svantaggi

- Eccesso di comunicazione
- Complessità
- Difficoltà nell'evitare duplicazione dei dati

8.1 Monoliti

Per **monolite** si intende la tipica applicazione di un'azienda il quale cambiamento di una piccola parte richiede il **rebuilding** e il **redeployment** dell'intero monolite. Inoltre lo **scaling** è richiesto per l'intera applicazione.

8.2 Microservizi

8.2.1 Service-orientation

Sviluppare un'applicazione come un insieme di servizi

- Ogni servizio è eseguito in container separati
- Comunicazione con meccanismi leggeri (HTTP request-response e code asincrone per disaccoppiamento)
- Poliglotta (linguaggi diversi)

8.2.2 Organizzare servizi intorno alle business capability

- Cross-functional team

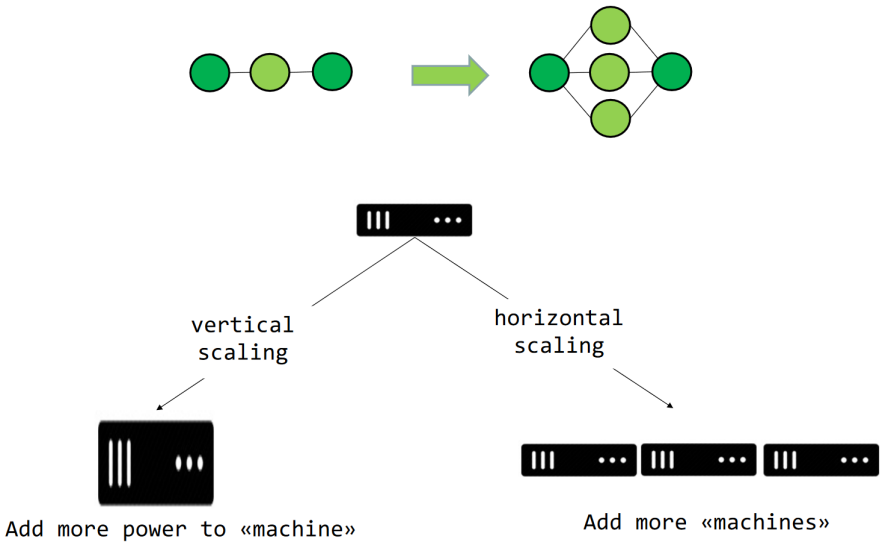
8.2.3 Decentralizzazione dei dati

- Ogni servizio ha un suo database
- Eventual consistency e compensation invece di transazioni distribuite

8.2.4 Servizi sviluppabili indipendentemente

- Possibilità di modificare il servizio senza necessariamente cambiare gli altri servizi

8.2.5 Servizi scalabili in orizzontale



8.2.6 Servizi resistenti agli errori

- Tolleranza agli errori è un requisito
- Ogni chiamata di servizio può fallire
- Chiamate sincrone tra servizi possono indurre a problemi di downtime

8.3 CAP Theorem

In presenza di una **partizione** di rete, non puoi avere sia **accessibilità** che **consistenza**.

8.3.1 Approccio Netflix

Per replicare i dati in n nodi Netflix:

- Scrive dove è momentaneamente possibile, poi sistema le mancanze
- Usa il quorum: (n/2+1) devono rispondere

8.4 Chaos monkey

Il metodo **chaos monkey** termina casualmente istanze e container che vengono eseguiti all'interno dell'ambiente di produzione

Capitolo 9

Kubernetes

9.1 Containers

- I container forniscono un meccanismo leggero per isolare lo l'ambiente di un'applicazione
- Le immagini dei container possono essere eseguiti in modo affidabile su ogni macchina, fornendoci la portabilità dallo sviluppo al deployment
- Più carichi di lavoro possono essere piazzati sulla stessa macchina fisica, raggiungendo elevati utilizzi di memoria e CPU

9.2 Kubernetes (Container Orchestration)

- Gestisce l'intero ciclo di vita dei singoli container, **avviando o spegnendo le risorse in base alla necessità**. Se un container si spegne inaspettatamente K8 ne lancia un altro al suo posto.
- Fornisce un **meccanismo** alle applicazioni per **comunicare tra loro** anche quando i container sottostanti vengono creati e distrutti.
- Dato un gruppo di carichi di lavoro per container da eseguire e un insieme di macchine su un cluster, esamina ogni container e **determina la macchina ottimale a cui schedulare il carico di lavoro**

9.3 Design principles: Dichiaratività

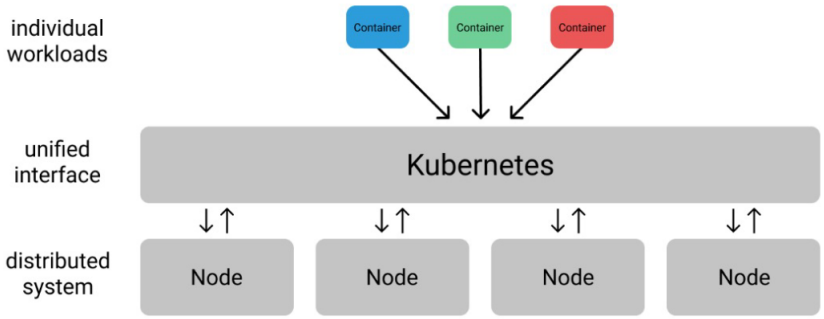
K8s troverà quando l'attuale stato del sistema non rispetta le nostre aspettative e interverrà per sistemare il problema (**self-healing**).

Definiamo gli stati desiderati da una collezione di oggetti

- Ogni oggetto ha una specifica nella quale forniamo lo stato desiderato e uno stato che riflette lo stato corrente dell'oggetto
- K8 esegue costantemente domande a ogni oggetto per assicurarsi che il suo stato sia uguale alla specifica
    - Se un oggetto non risponde, K8 avvia una nuova versione per sostituirlo
    - Se un oggetto si è allontano dalle specifiche, K8 emette i comandi necessari per riportare l'oggetto allo stato desiderato

9.4 Design principles: Distribuzione

K8s fornisce una interfaccia unificata per interagire con un cluster di macchine. Non ci dobbiamo preoccupare della comunicazione tra le singole macchine.



9.5 Design principles: Disaccoppiamento

- I container dovrebbero essere sviluppati avendo un solo obiettivo in mente (**microservice-based architecture**)
- K8s supporta nativamente l'idea di disaccoppiare i **servizi** che possono essere **scalati e aggiornati indipendentemente**

9.6 Design principles: Strutture immutabili

- Per ottenere il massimo dai container e dall'orchestrazione, dovremmo distribuire una **struttura immutabile**
- Durante il ciclo di vita di un progetto dovremmo **usare la stessa immagine per il container** (e modificare solo le configurazioni esterne all'immagine)
- I container sono progettati per essere **effimeri**, **pronti per essere rimpiazzati da altri container** in qualsiasi momento
- Mantenendo la struttura immutata rendiamo più facile il **roll back** delle applicazioni **a stati precedenti**, possiamo semplicemente aggiornare la nostra configurazione per usare una vecchia immagine del container.

9.7 K8s Objects

Gli oggetti di K8s possono essere definiti in **manifesti** (YAML o JSON).

9.7.1 K8s objects: Pod

Un **Pod** consiste in

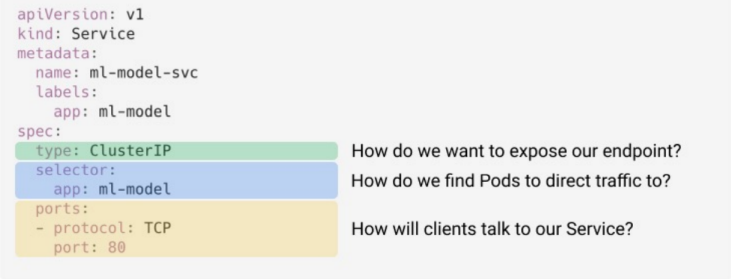
- uno o più container strettamente in relazione
- un livello di rete condiviso
- un volume di filesystem condiviso

9.7.2 K8s objects: Deployment

Un **deployment** object include una collezione di **Pods** definiti da un template e il numero di copie del template che si vogliono eseguire. Il cluster proverà sempre ad avere n Pods disponibili.

9.7.3 K8s objects: Service

Ogni Pod è assegnato ad un unico indirizzo IP che possiamo usare per comunicare con esso.
Service fornisce un endpoint stabile per direzionare il traffico al Pod desiderato anche se il Pod subisce aggiornamenti, scaling o failures.



9.7.4 K8s objects: Ingress

Per esporre la nostra applicazione a traffico esterno verso il nostro cluster, definiamo un **Ingress** object.

9.8 K8s Control plane

Ci sono due tipi di macchine in un cluster:

- **Master node**: macchina che contiene la maggior parte dei componenti del control plane
- **Worker node**: macchina che esegue i workloads delle applicazioni0

9.8.1 K8s Control plane: Master node

L'utente fornisce nuovi object specification all'**API server** del master node:

- Il server API convalida le richieste di aggiornamento e agisce come interfaccia unificata per richieste sullo stato corrente del cluster
- Lo stato del cluster è archiviato in una key-value distribuita **etcd**

Lo **scheduler** determina dove gli oggetti dovrebbero essere eseguiti. Lo scheduler:

- Richiede il server API i cui oggetti non sono stati assegnati ad una macchina
- Determina a quali macchine quegli oggetti dovrebbero essere assegnati
- Risponde all'API server di riflettere il compito

Il **controller-manager** monitora lo stato dei cluster attraverso i server API, se lo stato attuale differisce dallo stato desiderato, il **control manager** effettuerà dei cambiamenti attraverso il server API per guidare il cluster verso lo stato desiderato.

9.8.2 K8s Control plane: Worker node

Kubelet:

- Agisce come agente del nodo che comunica con il server API per vedere quali container workloads sono stati assegnati al nodo
- Responsabile di far eseguire i workload assegnati ai Pods
- Quando un nodo entra la prima volta in un cluster, annuncia l'esistenza del nodo al server API in modo che lo scheduler possa assegnare il Pod a esso

Kube-proxy abilita i container a comunicare tra loro attraversando vari nodi sul cluster

9.9 Quando non usare K8s

- Se puoi eseguire workload su una singola macchina
- Se i calcoli sono leggeri
- Se non necessiti alta disponibilità e puoi tollerare downtime
- Se non prevedi molte modifiche al servizio
- Se hai un monolite e non prevedi di spezzarlo in microservizi

9.10 K8s vs Docker Swarm

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 environments where simplicity and fast development is favored	Preferred for environments where medium to large clusters are running complex applications