Università di Pisa

Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Corso a Libera Scelta - 6 CFU

Cloud Computing

Professore:
Prof. Antonio Brogi

Autore:
Filippo Ghirardini

Anno Accademico 2023/2024

Contents

Cloud Computing

Realizzato da: Ghirardini Filippo

A.A. 2023-2024

---

1 Introduzione

1.1 Service-based economy

L'economia basata sui servizi (*Everything as a service*) si fonda sulla tendenza degli ultimi 30 anni di passare dai *beni* ai *servizi*.

Esempio 1.1.1 (Bicicletta)**.** Una signora compra una bicicletta dal venditore. Questa diventa di sua proprietà e si deve occupare della manutenzione. La bicicletta diventa un **servizio** (*CicloPi*) e la signora paga per usare la bici che però non è più sua (e non deve più preoccuparsi di manutenzione e furto).

Esempi più vicini all'informatica sono il passaggio dai supporti fisici per la musica allo streaming o i dispositivi di memorizzazione passati ai Cloud Drive.

1.2 Service contracts

Quando usiamo un servizio non vogliamo sapere come viene implementato. L'unica cosa che ci interessa è cosa è specificato sul **contratto di utilizzo**. Nella maggior parte dei casi l'utente non lo legge.

1.2.1 Quality of Service

Ci sono più fornitori che ci danno lo stesso servizio ma con qualità del servizio diverse. Dobbiamo chiederci se il prezzo più basso vale la pena del sacrificio della qualità.

1.2.2 Service Level Agreement

Sono i contratti di servizio che includono anche il livello di **affidabilità di servizio**. In questa situazione abbiamo tre figure:

- *Programmatore*
- *Business expert*: colui che sa il livello di affidabilità in base al mercato
- *Legale*: colui che sa come scriverlo

Esempio 1.2.1 (Google SLA)**.** Google Compute Engine fornisce un **Service Level Objective** (SLO) del 99.95%. In caso di non raggiungimento del SLO si viene rimborsati con del credito in percentuale a quanto si è distanti dal target.

Montlhy Uptime Percentage	**Rimborso**
$95.00\% - < 99.95\%$	10%
$90.00\% - < 95.00\%$	25%
$< 90.00\%$	100%

Se ad esempio l'1 e il 2 Aprile dalle 8am alle 5pm (orario di lavoro) non era disponibile il servizio, a quanto ammonta il rimborso in credito?
Dobbiamo prima capire cos'è il **Montly Uptime Percentage** secondo l'SLA:

Definizione 1.2.1 (Monthly Uptime Percentage)**.** *Total number of minutes in a month, minus the number of minutes of* ***Downtime*** *suffered from all* ***Downtime Period*** *in a month, divided by the total number of minutes in a month.*

Definizione 1.2.2 (Downtime Period)**.** *A period of one of more consecutive minutes of* ***Downtime****.*

Definizione 1.2.3 (Downtime)**.** *For virtual machines instances: loss of external connectivity or persistent disk accessfor the Single Instance or, with respect to Instances in Multiple Zones, all applicable running instances.*

Inoltre il cliente deve richiedere entro 60 giorni il rimborso fornendo anche dei **log file** che mostrino il **Downtime Period** assieme alla data e all'orario.

a

Definizione 1.2.4 (Legge del pesce rosso)**.** *Quando un fornitore di servizi non garantisce in alcun modo il prodotto.*

Esempio 1.2.2 (Microsoft Service Agreement)**.** Microsoft non è responsabile per alcuna perdita di dati o interruzione di servizi e non li garantisce in alcun modo (vedi art.6 e art.11).

Esempio 1.2.3. Supponiamo di avere un servizio che si appoggia a due servizi cloud, ognuno disponibile in maniera indipendente il 90% del tempo. Il servizio è probabile che non sarà disponibile per 4 ore e mezza al giorno:

$$(1 - (0.90 * 0.90)) * 24 = 4.56$$

1.3 Cloud

Un primo fattore da tenere in considerazione è la **service demand**, che cambia nel tempo.
Prima del cloud c'era il problema di dimensionare il servizio in base a delle stime, che possono essere di due tipi:

- **Overprovisioning**: utilizzare un infrastruttura che possa sopportare anche i carichi massimi previsti. In questo caso abbiamo un problema di **spreco di risorse**.
- **Underprovisioning**: quando l'infrastruttura non è sufficiente per tutti i momenti di carico. Questo causerebbe la perdita di clienti che dopo qualche tentativo fallito di utilizzo abbandonerebbero il servizio.

Il **cloud** ci fornisce risorse apparentemente infinite disponibili su richiesta.

Definizione 1.3.1 (Cloud)**.** *Secondo il NIST il Cloud Computing è un **modello** per consentire l'accesso ubiquo, conveniente e a richiesta di risorse computazionali.*
Le idee chiave sono le seguenti:

- *Messa in comune di strutture autogestite, utilizzate come servizi*
- *Risorse virtuali **scalabili** disponibili attraverso internet a diversi clienti*
- ***Separazione** dei servizi dalla tecnologia alla base*

*Dal punto di vista economico sfrutta il principio **pay-per-use**, che permette all'utente di convertire i costi da Capital Expenses a Operation Expenses. Questo garantisce elasticità e trasferimento dei rischi al fornitore dei servizi.*

1.3.1 Service models

Analizziamo ora i modelli di servizio, prendendo come esempio la pizza:

- **IaaS**: fornisce server virtualizzati, archiviazione e rete e li gestisce. Il cliente è responsabile per tutti gli altri aspetti, quali OS e applicazioni. Nel nostro caso sarebbe comprare una pizza da cuocere.
- **PaaS**: fornisce l'intera piattaforma come servizio (VM, OS, servizi, SDKs) e gestisce l'infrastruttura, l'OS e l'abilitazione del software. Il cliente è responsabile dell'installazione e la gestione delle applicazioni. Nel nostro caso sarebbe una pizza d'asporto.
- **SaaS**: fornisce software a richiesta, disponibile tramite API. Viene gestita l'infrastruttura, l'OS e l'applicazione, quindi il cliente non è responsabile di nulla. Nel nostro caso sarebbe la pizzeria.

Vediamo il confronto in quanto a successo *revenue* e *market share*:

1.3.2 Deployments models

La scelta del deployment model può essere **pubblico**, **privato** o **ibrido**. Il modello privato garantisce un maggiore controllo sui dati mentre quello pubblico una maggiore scalabilità. La versione ibrida divide i dati tra pubblico e privato in base alle esigenze di sicurezza e scalabilità.

Tipo	1H22 Rev	1H23 Rev	1H22 Msh	1H23 Msh
IaaS	55.1 BUSD	64.4 BUSD	20.8%	20.4%
PaaS	44.5 BUSD	56.8 BUSD	16.8%	18.0%
SaaS SW	121.9 BUSD	141.2 BUSD	46.0%	44.7%
SaaS HW	43.3 BUSD	53.2 BUSD	16.4%	16.9%

1.3.3 Domande

Ci si deve fare alcune domande sulla confidenzialità dei nostri dati:

- Dove saranno fisicamente conservati i nostri dati?
- La privacy e l'integrità dei nostri dati sono garantite? Come?
- Come possiamo sapere se c'è stato un problema?

sulla disponibilità dei servizi:

- Cosa succede se il fornitore del servizio non lo fornisce?
- SPoF (Single Point of Failure): quando la nostra architettura sfrutta un solo servizio che potrebbe fallire

Le risposte le troviamo tutte nel SLA.

1.3.4 Vendor Lock-In

Definizione 1.3.2 (Vendor Lock-In)**.** *Il Vendor Lock-In è quando si rende un cliente dipendente da un venditore per prodotti o servizi, rendendogli impossibile il cambio ad un altro gestore senza affrontare costi esorbitanti.*

Cosa succede se si vuole cambiare provider? Il software continuerà a funzionare? Si potrà trasferire facilmente i propri dati? Serve sempre un **exit plan**.

2 Container

I container sono un metodo per creare la virtualizzazione e l'isolamento delle risorse in maniera più semplice.
La struttura è simile a quella della virtualizzazione, dove al posto dell'hypervisor abbiamo un **container manager**. La differenza principale è, però, che tutti i container operano sopra ad un unico OS, permettendo di risparmiare molte risorse sia in termini di performance che di spazio.

2.1 Storia

- UNIX **chroot** permetteva isolamento dal filesystem
- FreeBSD **jail** estese l'isolamento di chroot ai processi
- Google inizia a sviluppare i **CGroups** per il kernel di Linux
- I **Linux Container** forniscono una soluzione completa
- **Docker** aggiunge il concetto di immagini portabili e grafica semplice da utilizzare

2.2 Docker

Docker è una piattaforma che permette di sviluppare ed eseguire applicazioni in un ambiente isolato sfruttando i container.
I componenti software sono gestiti come **immagini** in sola lettura su cui vengono creati ed eseguiti i **container**. In aggiunta possono essere montati dei **volumi** per garantire la persistenza dei dati.

2.2.1 Images

Sono in sola lettura e vengono salvate in un **docker registry**, pubblico o privato, strutturati in repositories che contengono ognuna ogni versione di un determinato software.

2.2.2 Swarm mode

Docker prevede la **swarm mode** per gestire un gruppo di container (swarm). I **nodi** possono essere:

- *Managers*, che delegano le task ai workers
- *Workers*, che eseguono le task a loro assegnate

L'utente può definire lo stato desiderato dei vari servizi dell'applicazione. Ogni nodo avrà un DNS name univoco. Swarm si occuperà di fare *load balancing* e di mantenere la coerenza degli stati secondo quello desiderato.

3 FaaS

L'approccio FaaS è un mercato in crescita e ad oggi esistono molteplici opzioni tra le quali scegliere. In particolare le suddividiamo in due categorie: *commerciali* (e.g. AWS Lambda) e *open source* (e.g. Apache Openwisk).
Per fare una scelta consapevole, dobbiamo analizzare le proposte da due punti di vista: quello commerciale e quello tecnico.

3.1 Business view

3.1.1 Licensing

Tutte le piattaforme **open source** usano licenze abbastanza permissive, come ad esempio Apache 2.0. Quelle **commerciali** invece usano software proprietario, fatta eccezione per MS Azure Functions che ha alcune componenti libere.

	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

3.1.2 Installazione

Tra le opzioni commerciali solamente Azure functions ha dei servizi installabili *on-premises*. Quelle open source supportano invece molteplici host, con Kubernetes tra i più popolari.

	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[3], faasd, Kubernetes, OpenShift

3.1.3 Source code

Tutte le piattaforme open source sono hostate su GitHub ed implementate 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

3.1.4 Interfaccia

Tutte le piattaforme forniscono CLI, mentre API e GUI non sono sempre presenti in quelle open source. Inoltre in queste ultime il metodo di amministrazione cambia molto tra l'una e l'altra.

		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.

3.1.5 Community

Le piattaforme open source sono molto ben votate su GitHub ma poco cercate e diffuse su StackOverflow, a differenza di quelle commerciali.

		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.

3.1.6 Documentazione

Tutte le piattaforme forniscono la documentazione per l'utilizzo e il deploy del servizio ma non tutte le loro architetture sono documentate.

		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.

3.2 Technical view

3.2.1 Sviluppo

I linguaggi più comuni nello sviluppo in ambito FaaS sono Java, NodeJS e Python, con il supporto di Docker.

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.

3.2.2 Versioning

La maggior parte delle piattaforme implementano un sistema di versioning in maniera implicita, al contrario di quelle commerciali:

- **Dedicated mechanisms**
- **Implicit mechanisms**

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

3.2.3 Event sources

Tutte le piattaforme supportano la chiamata di funzioni tramite richieste **sincrone** HTTP, mentre solo alcune supportano quelle **asincrone**. Più della metà non supportano il salvataggio dei **dati delle richieste**. Gli **scheduler**, gli **stream processing** e il **messaging** sono ampiamente supportati. Più della metà delle piattaforme documentano la possibilità di creare **eventi personalizzati**.

3.2.4 Function orchestration

Più della metà delle piattaforme supportano il function orchestration tramite DSL proprietari o funzioni apposite.

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.

3.2.5 Testing e debugging

La maggior pare delle piattaforme supporta testing funzionale e debugging delle funzioni. La differenza sta nella sofisticatezza di questi sistemi, che è più elevata per le piattaforme commerciali.

3.2.6 Logging

Le piattaforme commerciali usano strumenti ad hoc per monitorare lo stato del servizio mentre quelle open source si rifanno a servizi terzi.

	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

3.2.7 Application delivery

La maggior parte delle piattaforme segue un approccio dichiarativo per automatizzare il deploy delle applicazioni. Solo quelle commerciali supportano la pipeline di tipo CI/CD (Continuous Implementation/Continuous Deployment).

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	Kubernetes (P)[a]	n/s
Kubeless	Kubernetes (P)[a], Serverless Framework (T)	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.

3.2.8 Riuso

Solamente AWS Lambda e MS Azure Functions prevedono un marketplace per condividere e riutilizzare funzioni.

3.2.9 Gestione degli accessi

Le piattaforme commerciali supportano nativamente l'autenticazione e il controllo delle risorse. Quelle open source di solito sfruttano features del sistema operativo.

4 PaaS

L'approccio PaaS prevede che i fornitori esterni gestiscano sia hardware che software e che l'utente fornisca solamente i dati e l'applicativo.
I **vantaggi** sono:

- Ridotta gestione per l'utente
- Manutenzione automatica
- Load balancing, scaling e distribuzione più efficienti
- Più facilità nell'adottare nuove tecnologie

Gli **svantaggi** invece:

- Disponibilità del servizio molto dipendente dal fornitore
- Vendor lock-in
- In balia di eventuali cambiamenti da parte del fornitore

4.1 Heroku

Heroku è una piattaforma cloud basata sulla gestione di un sistema di container, con data services integrati e un ampio ecosistema, per sviluppare ed eseguire app moderne.