

TESI DI LAUREA  
IN  
TECNOLOGIE WEB

## **Analisi e Rappresentazione del Comportamento dei Ritardi di Rete dei Principali Fornitori di Cloud Computing**

Candidato:

**Alberto Bagnacani**

Relatore:

**Chiar.mo Prof. Paolo Bellavista**

Correlatore:

**Chiar.mo Dott. Ing. Luca Foschini**

Alla mia famiglia

# Indice

|  |           |
|--|-----------|
| <b>Introduzione</b>  | <b>4</b>  |
| Obiettivi e motivazioni . . . . .  | 4         |
| <b>1 Cloud Computing</b>   | <b>6</b>  |
| Cos'è il Cloud Computing . . . . .   | 6         |
| Breve storia del Cloud Computing . . . . .                                 | 7         |
| Vantaggi e svantaggi del Cloud Computing . . . . .                         | 8         |
| Vantaggi . . . . .   | 8         |
| Svantaggi . . . . .  | 8         |
| Tipi di servizio del Cloud Computing . . . . .                             | 9         |
| IaaS . . . . .   | 10        |
| PaaS . . . . .   | 11        |
| SaaS . . . . .   | 11        |
| Analisi del comportamento dei ritardi di rete . . . . .                    | 12        |
| <b>2 Lo stack MERN</b>   | <b>13</b> |
| Scelta delle tecnologie . . . . .  | 13        |
| Introduzione . . . . .   | 14        |
| Suddivisione dei compiti . . . . .   | 14        |
| Importanza dello stack . . . . .   | 15        |
| Evoluzione e varietà in continua crescita . . . . .                        | 16        |
| Evoluzione di JavaScript . . . . .   | 16        |
| Node.js . . . . .  | 17        |
| Vantaggi di Node.js . . . . .  | 17        |
| Svantaggi di Node.js . . . . .   | 18        |
| Compagnie che utilizzano la tecnologia . . . . .                           | 18        |
| Express . . . . .  | 18        |
| React . . . . .  | 19        |
| Introduzione e breve storia . . . . .                                      | 19        |
| Vantaggi di React . . . . .  | 19        |
| Svantaggi di React . . . . .   | 21        |
| Compagnie che utilizzano la tecnologia . . . . .                           | 21        |
| MongoDB . . . . .  | 21        |
| Vantaggi di MongoDB . . . . .  | 22        |
| Svantaggi di MongoDB . . . . .   | 22        |
| Databases relazionali (SQL) vs databases non relazionali (NoSQL) . . . . . | 22        |
| Compagnie che utilizzano la tecnologia . . . . .                           | 23        |

|  |           |
|--|-----------|
| <b>3 Progetto</b>  | <b>24</b> |
| Funzionalità . . . . .   | 24        |
| Architettura . . . . .   | 24        |
| Probing . . . . .  | 27        |
| Persistenza . . . . .  | 27        |
| Back-end . . . . .   | 29        |
| Front-end . . . . .  | 29        |
| <b>4 Implementazione progetto</b>  | <b>31</b> |
| Cloud Provider . . . . .   | 31        |
| Amazon Web Services . . . . .  | 32        |
| Google Cloud Platform . . . . .  | 33        |
| Microsoft Azure . . . . .  | 34        |
| IBM Cloud . . . . .  | 35        |
| Probing . . . . .  | 36        |
| registerPing.sh . . . . .  | 36        |
| curlCsv.sh . . . . .   | 37        |
| Invio dati giornalieri . . . . .   | 37        |
| Persistenza . . . . .  | 37        |
| Memorizzazione dati . . . . .  | 38        |
| Ottimizzazione delle performances . . . . .                              | 39        |
| Back-end . . . . .   | 40        |
| Installazione del Web Server . . . . .                                   | 40        |
| Ricezione dei files giornalieri, comunicazione con il database . . . . . | 40        |
| Modellazione del dominio . . . . .                                       | 41        |
| Esposizione di endpoint e paginazione delle API . . . . .                | 41        |
| Manipolazione dati: query . . . . .                                      | 45        |
| Front-end . . . . .  | 46        |
| Interfaccia utente . . . . .   | 47        |
| Grafici . . . . .  | 50        |
| Comunicazione mediante gli endpoint con il back-end . . . . .            | 51        |
| <b>5 Risultati sperimentali</b>  | <b>53</b> |
| Analisi back-end . . . . .   | 53        |
| Salvataggio, parsing ed importazione dei dati sul database . . . . .     | 53        |
| Richieste operate agli endpoint da parte del front-end . . . . .         | 54        |
| Analisi persistenza . . . . .  | 54        |
| Impiego delle risorse di sistema . . . . .                               | 55        |
| Tempo impiegato dall'aggregazione . . . . .                              | 55        |
| Probing . . . . .  | 56        |
| Front-end . . . . .  | 56        |
| Dati raccolti e futuri . . . . .   | 57        |
| Analisi dei risultati . . . . .  | 59        |
| Maggio 2018 . . . . .  | 59        |
| Giugno 2018 . . . . .  | 59        |
| Luglio 2018 . . . . .  | 60        |
| Agosto 2018 . . . . .  | 60        |
| Risultato finale del progetto . . . . .                                  | 60        |
| <b>6 Conclusioni</b>   | <b>64</b> |

# Introduzione

## Obiettivi e motivazioni

Per molte applicazioni con un deployment cloud fondato su data center, è importante potersi basare su un modello affidabile di ritardi all'interno di differenti aree terrestri. Sfortunatamente, questo modello non è disponibile a ricercatori e professionisti.

L'infrastruttura del Cloud Computing contemporaneo è composta da data center interconnessi in tutto il mondo, che offrono le proprie risorse su base pay-per-use. Questo scenario apre nuove sfide ed opportunità per sviluppare servizi e sistemi di gestione IT su scala globale.

Dalla prospettiva dell'utente Cloud, i provider di servizi potrebbero decidere di replicare i singoli componenti dell'infrastruttura in diverse posizioni, per ottenere una maggiore affidabilità.

Dal punto di vista del fornitore, invece, diversi sforzi di ricerca si concentrano sulla pianificazione del carico di lavoro in ambienti cloud distribuiti con struttura ad inter-data center. Dunque, il caso d'uso delle connessioni ed iterazioni tra i vari nodi è molto interessante sia dal punto di vista degli utenti Cloud che dei provider.

Per quanto concerne professionisti e ricercatori, una sfida chiave è rappresentata dal capire quale sia il comportamento dei ritardi di comunicazione in un modello realistico ad inter-data center. In effetti, la latenza è soggetta a diversi fattori non controllabili, come le variazioni delle tecniche di bilanciamento del carico operate dal load balancer [1].

Il progetto proposto nasce per colmare questa lacuna, attraverso un'iniziale analisi della latenza che caratterizza le connessioni tra differenti data center, posti in zone eterogenee del globo terrestre, per diversi fornitori di servizi.

L'intento è quello di creare un sistema in grado di ottenere i dati specificati, salvarli in maniera persistente, analizzarli e presentarli all'utente finale.

Verranno esaminati i Cloud Provider protagonisti del mercato: Amazon Web Services, Google Cloud Platform, Microsoft Azure e IBM Cloud.

Per ognuno di essi, verranno predisposte delle Virtual Machines nei diversi data center e verrà avviato un programma di monitoraggio dei ritardi di rete.

Sarà quindi possibile effettuare una comparazione tra i diversi fornitori ed analizzare la loro stabilità, in termini di latenza tra le diverse regioni.

I modelli attualmente presenti non permettono di offrire una visione generale di tutti i provider e presentano uno schema statico calcolato su precedenti test o ipotesi basate sulla posizione di una regione.

Con questo progetto si otterrà un sistema di monitoraggio dinamico e costantemente aggiornato di tutte le possibili combinazioni di data center, permettendo di delineare quelle che sono le connessioni che presentano una maggiore latenza. L'obiettivo finale è quello di offrire un'unica fonte di verità per i ritardi di rete del modello ad inter-data center, per tutti i Cloud Providers. La soluzione permetterà, per esempio, di progettare il deployment dei componenti di un'infrastruttura, spostandoli da un data center all'altro in caso di malfunzionamenti o di ritardi nella comunicazione tra gli stessi.

Il Capitolo 1 introduce il concetto di Cloud Computing, la sua storia ed evoluzione, i vantaggi e gli svantaggi creati da questo paradigma nel mondo dell'informatica e le varie tipologie di servizio; vengono inoltre specificati i principali problemi che caratterizzano questo modello e le soluzioni proposte dall'elaborato. Nel Capitolo 2 vengono descritte le principali tecnologie utilizzate, esaltando le proprietà di ognuna di esse. Il Capitolo 3 analizza la struttura e l'architettura del progetto, spiegando le funzionalità che ogni livello dell'applicazione dovrà realizzare. Il Capitolo 4 illustra dettagliatamente l'implementazione di ogni area descritta, approfondendo le soluzioni e gli algoritmi realizzati. Nel Capitolo 5 vengono riportati i risultati ottenuti, ponendo particolare attenzione alle prestazioni di ogni strato dell'applicativo. Il Capitolo 6 riporta le conclusioni, analizzando gli obiettivi raggiunti e le possibilità create, riportando i limiti riscontrati e le future implementazioni.

# Cloud Computing

## Cos'è il Cloud Computing

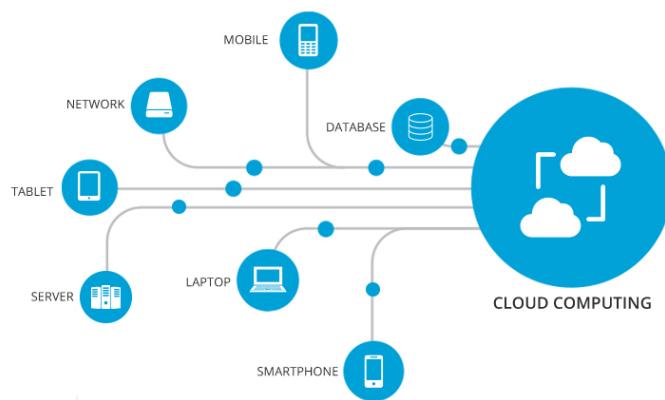


Figura 1.1: Il Cloud Computing

Sorgente: <http://www.zebinfotech.com/2018/06/22/disadvantages-are-addressed-by-cloud-computing/>

Con il termine inglese cloud computing, si indica un paradigma di distribuzione di risorse informatiche, come l'archiviazione, l'elaborazione e la trasmissione di dati, distinto dalla disponibilità su richiesta, attraverso Internet, partendo da una collezione di risorse preesistenti e configurabili.

I servizi non sono completamente configurati e messi a disposizione dal fornitore apposta per l'utente, ma gli sono assegnati, velocemente e convenientemente, grazie a procedure automatizzate, lasciando all'utilizzatore parte delle responsabilità della configurazione. Quando vengono rilasciate le risorse, esse sono riconfigurate allo stato iniziale e rimesse a disposizione nel pool condiviso, con altrettanta rapidità ed economia per il fornitore [2].

A prescindere dalla finalità d'uso, una piattaforma di servizi cloud fornisce accesso rapido a risorse informatiche flessibili e a basso costo.

Con il cloud computing, non è obbligatorio effettuare grandi investimenti in infrastrutture hardware o impiegare parecchio tempo in attività di gestione delle

stesse. È possibile invece effettuare la previsione delle risorse di elaborazione, in base a esigenze particolari. Il cloud computing consente di accedere alla quantità di risorse essenziali in modo praticamente istantaneo, pagando solo in base all'uso effettivo di queste [3].

## Breve storia del Cloud Computing

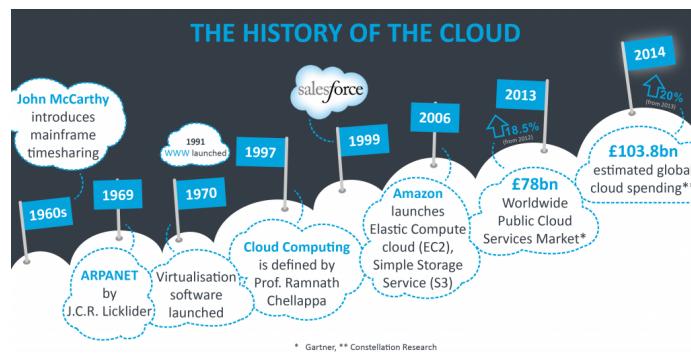


Figura 1.2: La storia del Cloud Computing

Sorgente: <https://timesofcloud.com/cloud-tutorial/history-and-vision-of-cloud-computing/>

Nel 1966, Douglas Parkhill spiega le caratteristiche dell'attuale Cloud computing nel libro "The challenge of Computer Utility".

Nel 1967 viene indetta la conferenza internazionale sulla rete Arpanet, precursore di Internet. Nel 1971 Arpanet connette 23 computer tra loro. Bisogna aspettare l'anno 1982 per la prima definizione della parola Internet e del noto protocollo TCP/IP.

È il 1991 quando il CERN (Centro Europeo di Ricerca Nucleare) rilascia la prima versione del World Wide Web. I primi a poterlo utilizzare sono università e laboratori. I computer diventano parte di una rete globale ed interconnessa.

Verso gli anni 2000 nascono le grandi aziende che cominciano ad operare nel settore: VMware, pioniera della virtualizzazione, e Salesforce.com, prima a fornire servizi in modalità cloud alle imprese.

Nell'anno 2002, BlackBerry immette sul mercato il primo precursore dello smartphone, rendendo così accessibile Internet da un dispositivo mobile. Nel 2004 Google con il suo servizio Gmail sale sul palcoscenico: il servizio di posta elettronica cloud based diventerà il più diffuso al mondo. Due anni dopo la stessa azienda inizierà ad offrire servizi ai quali accedere direttamente dal browser. Successivamente, Amazon lancia Web Services, un insieme di servizi basati sul cloud.

Nel 2008, nasce il cloud privato e nel 2011 Apple fa il suo ingresso nel mondo del cloud computing con il servizio iCloud.

Il cloud computing rappresenta la tecnologia di punta sulla quale sempre più aziende in tutto il mondo decidono di investire [4].

# Vantaggi e svantaggi del Cloud Computing

## Vantaggi

- Prospettiva economica
  - Diminuzione del TCO (Total Cost of Ownership): si risparmia sull'acquisto, l'installazione, la manutenzione e la dismissione di hardware e software
  - Migliore flessibilità: in base alle necessità delle risorse, è possibile effettuare un adeguamento contrattuale, fattore impossibile in caso di infrastrutture di proprietà
  - Possibilità di focalizzarsi sul proprio obiettivo: non è necessario assumere personale specializzato che si occupi della gestione dell'infrastruttura, in quanto questa è trasferita nel cloud
- Prospettiva tecnica
  - Maggiore scalabilità: nella necessità di ulteriori risorse (ad esempio per picchi di carico), il sistema di gestione e monitoraggio del cloud ha la possibilità di allocare dinamicamente le capacità necessarie per far fronte alla richiesta
  - Mobilità: dati ed applicazioni sono accessibili in qualsiasi luogo
  - Maggiore sicurezza: tutti i dati sono centralizzati e sottostanno alle regole di sicurezza del fornitore; pertanto si annulla la possibilità di una perdita di questi a causa della sottrazione fisica di materiale informatico nell'azienda
  - Possibilità di beneficiare di piani di Disaster Recovery: se previsto, è possibile mettersi al sicuro da eventuali problemi che potrebbero accadere nel data center

## Svantaggi

Le tecnologie su cui si basano i servizi cloud ed il relativo modello di business non hanno ancora raggiunto la maturità; per questo, i reparti di ricerca e sviluppo dei fornitori lavorano attivamente per dare ai clienti finali il livello di servizio e la qualità richiesta dai processi aziendali più delicati. I servizi cloud, inoltre, sono caratterizzati da un'infrastruttura hardware estremamente articolata: più un sistema è complesso, più diventa difficile comprenderne e gestirne i rischi nelle fasi di sviluppo e di deployment [5].

Analizziamo gli svantaggi sotto diversi punti di vista:

- Sicurezza
  - Non ci sono standard di sicurezza riconosciuti per i sistemi cloud
  - La collocazione fisica dell'hardware e del software non è nota
  - Rischio di perdita dei dati causati da problemi hardware o software
- Dipendenza (perdita di controllo)

- Nessun potere sulla qualità, la frequenza e sulle tempistiche degli interventi di manutenzione operate dal fornitore di servizio
- Migrazione verso un altro provider difficoltosa
- La valutazione delle risorse utilizzate e delle attività utente spetta completamente al provider
- Flessibilità
  - È impossibile richiedere personalizzazioni dell'infrastruttura
  - L'implementazione di aggiornamenti tecnologici è dettata dal provider, che potrebbe rallentarne o impedirne l'adozione
- Costi
  - È impossibile prevedere l'aumento in futuro dei costi
- Conoscenze
  - Sono richieste competenze specifiche per implementare e gestire i contratti con il provider
- Integrazione
  - L'integrazione con le periferiche locali e con apparati di sicurezza è complessa (e alcune volte impossibile)

## Tipi di servizio del Cloud Computing

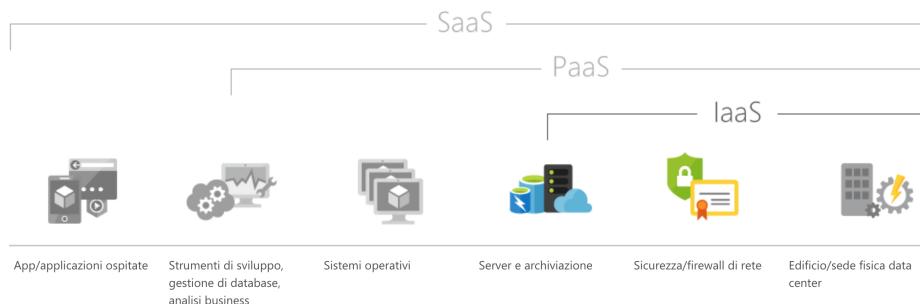


Figura 1.3: Tipi di servizio del Cloud Computing

Sorgente: <https://azure.microsoft.com/it-it/overview/what-is-iaas/>

Sono tre le ampie categorie in cui si possono raggruppare la maggior parte dei servizi di cloud computing [6]:

- infrastruttura distribuita come servizio (IaaS, Infrastructure as a Service)
- piattaforma distribuita come servizio (PaaS, Platform as a Service)
- software come un servizio (SaaS, Software as a Service)

Alcune volte si parla di stack di cloud computing, in quanto queste categorie sono innestate una sull'altra. Gli obiettivi aziendali possono essere raggiunti con una maggiore semplicità una volta a conoscenza delle differenze tra queste soluzioni.

## IaaS

Con il termine infrastruttura distribuita come servizio (IaaS, Infrastructure as a Service) si indica un'infrastruttura di calcolo immediatamente disponibile in cui viene effettuato il provisioning e che viene gestita tramite la rete Internet. Permette di ottenere rapidamente scalabilità verticale su richiesta, pagando solo per le risorse che vengono utilizzate.

Questo modello aiuta ad evitare le spese e la complessità legate all'acquisto e alla coordinazione di server fisici e ad altra infrastruttura di data center. Ogni risorsa è offerta come componente di un servizio distinto e basta affittarne una specifica solo nel momento della necessità.

### Scenari aziendali

I vari scenari in cui un modello IaaS può essere utilizzato sono:

- Test e sviluppo: è possibile configurare e smantellare rapidamente ambienti di test e sviluppo, introducendo nuove applicazioni sul mercato in tempi più rapidi. Il modello IaaS permette scalabilità verticale rapida ed economica
- Hosting di siti Web: l'esecuzione di siti Web grazie ad una soluzione IaaS può risultare meno costosa rispetto all'hosting Web tradizionale
- Archiviazione, backup e ripristino: le organizzazioni evitano il versamento di capitale per le risorse di archiviazione e la complessità di gestione di queste, che in genere richiede personale esperto in grado di gestire i dati e soddisfare i requisiti legali. Una soluzione IaaS viene in soccorso nella gestione di richieste non prevedibili e per soddisfare esigenze di archiviazione in continua crescita. Può inoltre semplificare e migliorare la pianificazione e la gestione dei sistemi di backup e di ripristino
- App Web: il modello IaaS fornisce tutta l'infrastruttura per supportare le app Web, permettendo alle organizzazioni di distribuirle rapidamente e ottenere semplice scalabilità verticale dell'infrastruttura quando la richiesta per le app non è prevedibile
- HPC (High Performance Computing): il calcolo di tipo high performance computing (HPC) in ambienti di grid computing o cluster di computer, permette di risolvere problemi complessi che coinvolgono milioni di variabili o calcoli. Alcuni esempi possono essere rappresentati dalle simulazioni di terremoti, dalla modellazione finanziaria e dalla valutazione delle progettazioni dei prodotti
- Analisi di Big Data: con Big Data si indica un set di dati di dimensioni molto grandi che contengono tendenze, associazioni e modelli preziosi. Le

operazioni di data mining permettono di individuare o estrapolare mode e statistiche. Questi modelli nascosti richiedono un'enorme quantità di potenza di elaborazione, che viene offerta dalle soluzioni IaaS

## PaaS

Il modello di piattaforma distribuita come servizio (PaaS, Platform as a Service) è un ambiente di sviluppo e distribuzione completo nel cloud, con risorse che consentono di distribuire semplici app o complesse applicazioni aziendali.

Come i sistemi IaaS, le soluzioni PaaS includono l'infrastruttura, ma anche middleware, come strumenti di sviluppo, servizi di business intelligence (BI), sistemi di gestione dei database e molto altro. Questo modello è progettato per supportare il ciclo di vita completo delle applicazioni Web: creazione, testing, distribuzione, gestione e aggiornamento.

Il modello PaaS permette di evitare le spese e la complessità legate all'acquisto di licenze software, middleware e infrastruttura delle applicazioni sottostanti.

### Scenari aziendali

Le organizzazioni usano in genere questo modello per i seguenti scenari:

- Framework di sviluppo: la soluzione specificata fornisce un framework su cui gli sviluppatori si possono basare per sviluppare applicazioni basate sul cloud. Funzionalità come scalabilità, disponibilità e capacità multi-tenant sono già incluse e permettono di ridurre la quantità di codice che gli sviluppatori devono scrivere
- Analisi o Business Intelligence: gli strumenti forniti da una soluzione PaaS consentono alle aziende di analizzare i dati ed eseguire operazioni di data mining, individuando informazioni approfondite e modelli, analizzando i risultati per migliorare le previsioni, le decisioni in materia di progettazione dei prodotti, il ritorno sugli investimenti e altre decisioni aziendali

## SaaS

Il software come un servizio (SaaS, Software as a Service) consente agli utenti di connettersi ad applicazioni basate sul cloud tramite Internet ed usarle.

L'infrastruttura sottostante, come il middleware, il software delle app e i dati di queste si trovano nel data center del fornitore di servizi. Questo gestisce l'hardware e il software e, con il contratto di servizio adeguato, garantisce la disponibilità e la sicurezza dell'app e dei dati.

### Scenari aziendali

Con il SaaS l'azienda che utilizza il servizio non controlla l'infrastruttura che supporta il software: il provider è responsabile di gestire il livello di rete, i server, lo storage ed i sistemi operativi.

Il cliente può solo decidere se limitare le funzionalità offerte stabilendo criteri quali la gestione delle identità e la prioritizzazione degli accessi tramite un insieme di configurazione dedicate.

## Analisi del comportamento dei ritardi di rete

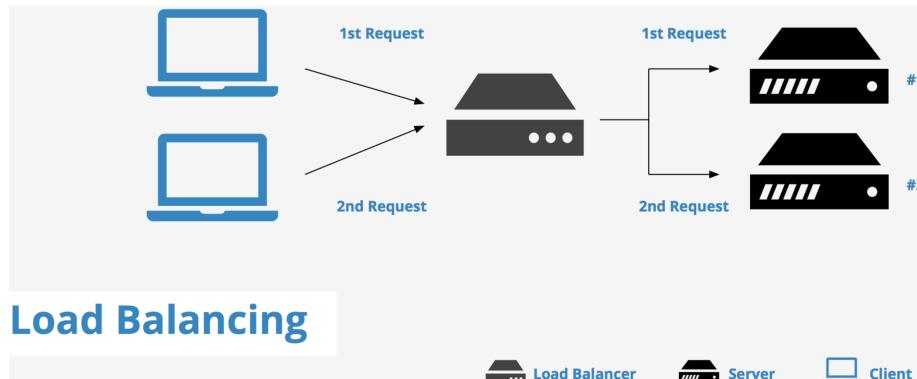


Figura 1.4: Funzionamento del load balancer

Sorgente: <https://www.keycdn.com/support/load-balancing/>

Il Cloud Computing offre vantaggi in termini di costo, flessibilità e disponibilità dei servizi per gli utenti. Queste caratteristiche fanno sì che cresca continuamente la domanda complessiva nei confronti di questo paradigma, sollevando problemi tecnici di varia natura, come l'alta disponibilità e la scalabilità.

Il load balancer<sup>1</sup> è responsabile di garantire tali proprietà, attraverso il bilanciamento automatico e trasparente dei carichi di lavoro tra più risorse di elaborazione, in maniera uniforme. L'obiettivo principale è quello di ottimizzare l'utilizzo delle risorse, massimizzare il throughput, minimizzare il tempo di risposta ed evitare il sovraccarico di una risorsa. L'utilizzo nell'ambito dei data center permette un uso più efficiente della larghezza di banda, riducendo i costi di provisioning. Le politiche di questo strumento fanno sì che la latenza tra data center sia soggetta ad un'elevata variabilità, a causa del cambiamento del percorso dei dati, nel processo di ridistribuzione delle risorse.

Il progetto proposto permette di creare un primo modello affidabile dei ritardi di comunicazione dell'ambiente distribuito dei vari fornitori, monitorando ed analizzando la latenza che caratterizza i diversi data center. Questi risultati verranno successivamente restituiti all'utente finale.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Load\\_balancing\\_\(computing\)](https://en.wikipedia.org/wiki/Load_balancing_(computing))

# Lo stack MERN

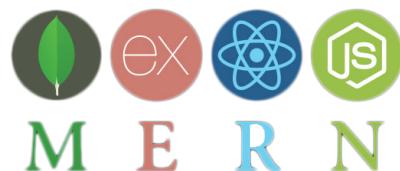


Figura 2.1: Lo stack MERN

Sorgente: <http://adsvento.in/mern-stack-development.php>

## Scelta delle tecnologie

È necessario un set di tecnologie che permetta di memorizzare, analizzare e presentare i dati riguardanti i ritardi di comunicazione tra diversi data center: lo stack MERN (MongoDB, Express, React e Node.js) permette di realizzare queste funzionalità.

L'utilizzo di questo stack standard permette di condividere un obiettivo comune tra le varie tecnologie: eliminare le corse critiche e ripetitive e permettere allo sviluppatore di lavorare dove può davvero aggiungere valore. Ulteriore punto a favore è l'utilizzo di un unico linguaggio per tutta l'infrastruttura: JavaScript.

MongoDB si occuperà di salvare i dati raccolti dal sistema di monitoraggio installato sulle Virtual Machines, Express e Node.js collaboreranno per realizzare il back-end, per analizzare i dati ed infine React implementerà il front-end, con cui l'utente interagirà.

## Introduzione

Gli utenti richiedono in maniera frequente un'esperienza sempre più attiva dai siti Web, aspettandosi lo stesso livello di prestazioni e interattività che ottengono con le app native per sistemi desktop e dispositivi mobili. Contemporaneamente, viene indotta una forte pressione sugli sviluppatori per fornire nuove applicazioni sempre più veloci, garantendo allo stesso tempo l'alta disponibilità e scalabilità del prodotto realizzato. Fortunatamente, esiste un set di tecnologie che rende possibile tutto ciò.

Il momento in cui il ruolo di JavaScript era limitato all'aggiunta di effetti visivi o finestre popup è passato. Gli sviluppatori ora utilizzano questo linguaggio per implementare lo strato di front-end, la logica di business e accedere al database. Esistono due stack di app Web JavaScript dominanti: MEAN (MongoDB, Express, Angular, Node.js) e MERN (MongoDB, Express, React, Node.js). In questo progetto verrà utilizzata quest'ultima proposta, che ha permesso di garantire una notevole reattività e scalabilità all'intera infrastruttura.

MERN è un insieme di componenti Open Source che, insieme, provvedono a fornire un framework end-to-end per la creazione di applicazioni web dinamiche, partendo dall'alto (codice in esecuzione nel browser) verso il basso (database) [7]. Lo stack è composto da:

- MongoDB: database utilizzato dal back-end per archiviare i propri dati in documenti JSON (JavaScript Object Notation).
- Express: framework di applicazioni Web back-end in esecuzione su Node.js
- React: framework di applicazioni Web front-end; esegue il codice JavaScript nel browser del cliente, garantendo dinamicità e scalabilità all'interfaccia utente
- Node.js: ambiente runtime JavaScript, che permette di implementare il back-end dell'applicazione

L'elemento comune dell'ambiente è JavaScript: ogni linea di codice è scritta infatti con questo linguaggio di programmazione.

## Suddivisione dei compiti

È presente una sovrapposizione tra le funzionalità disponibili nelle tecnologie che costituiscono lo stack MERN ed è necessario decidere "chi fa cosa".

Sicuramente la decisione più importante è quella di determinare chi debba eseguire le operazioni complesse. Sia Express che React includono funzionalità per il routing delle pagine, l'esecuzione del codice dell'applicazione e possono essere utilizzati per implementare la business logic di applicazioni sofisticate. L'approccio più tradizionale sarebbe quello di implementare lo sviluppo delle funzionalità nel back-end in Express. Tutto ciò presenta diversi vantaggi:

- Il back-end è più vicino al database e ad altre risorse, quindi può ridurre al minimo la latenza se vengono effettuate molte chiamate al sistema di persistenza
- I dati sensibili possono essere conservati in questo livello che è più sicuro
- Il codice dell'applicazione è nascosto all'utente, proteggendo la proprietà intellettuale di questo
- È possibile utilizzare server potenti, con conseguente aumento delle prestazioni

Tuttavia, c'è una tendenza crescente che indica l'utilizzo di React come luogo computazionale, in esecuzione nel browser dell'utente. Questo permette di:

- Utilizzare la potenza di elaborazione delle macchine degli utenti, riducendo la necessità di risorse elevate per alimentare il back-end. Questo fornisce un'architettura più scalabile, in cui ogni nuovo utente porta con sé le proprie risorse di calcolo
- Migliori tempi di risposta (presupponendo che non ci siano troppe richieste verso il back-end per accedere al database o ad altre risorse)
- Applicazioni progressive: si continua a fornire un servizio quando l'applicazione client non può contattare il back-end (ad es. quando l'utente non ha una connessione Internet). I browser moderni permettono all'applicazione di archiviare i dati localmente e di sincronizzarsi con il sistema quando viene ripristinata la connettività.

MongoDB ha un sofisticato framework di aggregazione che può eseguire molte analisi, spesso in modo più efficiente rispetto ad Express o React poiché tutti i dati richiesti sono locali. È qui che il progetto in questione ha deciso di assegnare la maggior parte della complessità di calcolo.

Un'altra decisione è decidere lo strato in cui convalidare tutti i dati che l'utilizzatore fornisce.

Idealmente questo processo dovrebbe essere eseguito il più vicino possibile all'utente, utilizzando React per verificare che una password fornita soddisfi le regole di sicurezza e consenta il feedback istantaneo al cliente. È inoltre possibile far controllare i documenti a MongoDB, permettendo di evitare la scrittura di dati errati da parte di software maliziosi.

## Importanza dello stack

Avere uno stack di applicazioni standard ed uniforme rende molto più facile e veloce l'introduzione di nuovi sviluppatori e la loro rapida implementazione, in quanto vi sono buone probabilità che abbiano utilizzato la stessa tecnologia altrove.

Da MongoDB verso l'alto, queste tecnologie condividono un obiettivo comune: prendersi cura delle aspetti critici e ripetitivi, per liberare gli sviluppatori e lavorare dove questi possono davvero aggiungere valore.

Queste sono le tecnologie che stanno rivoluzionando il web, donando l'aspetto, la sensazione e le prestazioni tipiche delle applicazioni native per desktop o mobile.

La separazione dei livelli, ed in particolare le API REST, hanno segnato la fine delle applicazioni monolitiche. Un'applicazione anziché essere un'entità isolata, ora può interagire con più servizi tramite API pubbliche.

## Evoluzione e varietà in continua crescita

Anche limitandosi all'ecosistema JavaScript, la gamma in continua espansione di framework, librerie, strumenti e linguaggi è impressionante. Ad esempio, se si è alla ricerca di un middleware per realizzare una particolare funzionalità, è probabile che questo sia reperibile online, già realizzato da uno sviluppatore.

## Evoluzione di JavaScript

Il linguaggio JavaScript stesso non è stato immune ai cambiamenti.

Ecma International<sup>1</sup> è stata creata per standardizzare le specifiche del linguaggio JavaScript e per aumentarne la sua portabilità: l'idea è che qualsiasi codice scritto in questo linguaggio possa essere eseguito in qualsiasi browser o altro ambiente di runtime JavaScript.

Una delle versioni più recenti e ampiamente supportate è ECMAScript 6, normalmente denominata ES6; questa è supportata dalle versioni recenti di Chrome, Opera, Safari e Node.js. Le seguenti sono alcune delle funzionalità chiave aggiunte in ES6:

- Classi e moduli
- Promises: un modo più conveniente per gestire il completamento o il fallimento delle chiamate di funzioni asincrone (rispetto alle callback)
- Arrows function: sintassi concisa per scrivere espressioni di funzione
- Generators: funzioni che possono dare la precedenza ad altri di eseguire
- Iteratori
- Matrici tipizzate

Poiché ES6 e Typescript non sono supportati in tutti gli ambienti, è comune trasporre il codice in una versione precedente di JavaScript per renderlo più portabile. Alla scrittura, React utilizza Babel (via react-script) per effettuare il transpiling del nostro codice.

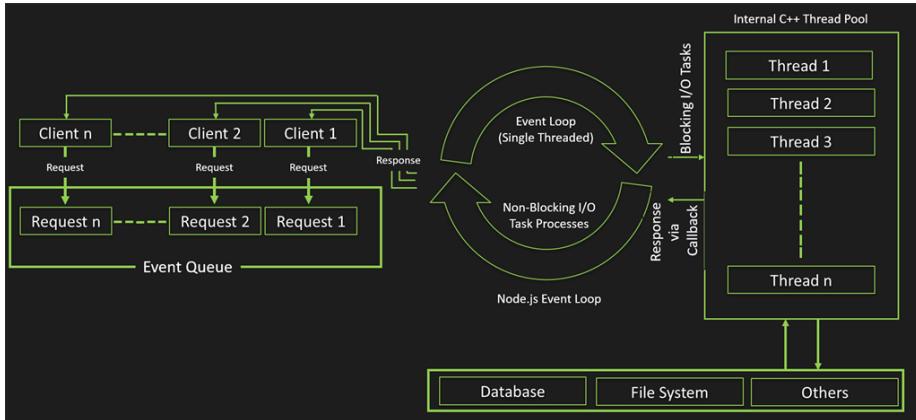


Figura 2.2: Funzionamento di Node.js

Sorgente: <https://www.c-sharpcorner.com/article/node-js-event-loop/>

## Node.js

Node.js è un ambiente runtime Javascript che esegue l'applicazione back-end; è basato sul motore JavaScript V8 di Google che viene utilizzato nel browser Chrome. Esso include anche una serie di moduli che forniscono funzionalità essenziali per l'implementazione di applicazioni Web, inclusi protocoli di rete come HTTP. I moduli aggiuntivi, incluso il driver di MongoDB, possono essere installati utilizzando lo strumento *npm*.

Node.js è un motore asincrono, basato su eventi, in cui l'applicazione effettua una richiesta e successivamente continua a lavorare su altre attività utili invece di bloccarsi nell'attesa di una risposta. Al termine della mansione richiesta, l'applicazione viene notificata dei risultati. Questo permette di eseguire un numero elevato di operazioni, necessario quando si ridimensionano le applicazioni.

## Vantaggi di Node.js

- Facile da imparare: secondo un'indagine svolta da Node.js<sup>2</sup>, JavaScript è uno dei linguaggi di programmazione più diffusi per lo sviluppo front-end. Questo permette al linguaggio suddetto di essere implementato anche nel back-end, richiedendo meno sforzo e meno tempo per imparare e lavorare sulla tecnologia
- Libertà nella costruzione di app: mentre Ruby on Rails è un framework che impone regole e linee guida per lo sviluppo di software tramite determinate direttive, Node.js offre una maggiore libertà di programmazione
- Full-stack JS: prima di Node.js, JavaScript è stato utilizzato solo per lo sviluppo lato client. Era dunque necessario utilizzare un diverso linguaggio di programmazione lato server, vincolando l'azienda ad assumere due

<sup>1</sup><https://www.ecma-international.org/>

<sup>2</sup><https://nodejs.org/static/documents/2016-survey-report.pdf>

diversi sviluppatori per il front-end e il back-end; Node.js risolve queste lacuna. Oggigiorno è possibile scrivere entrambe le parti delle applicazioni Web in JavaScript, rendendo la distribuzione di queste molto più semplice ed efficiente

- Comunità attiva: la comunità di sviluppatori Node.js è un gruppo molto attivo e vivace, che contribuisce al costante miglioramento della struttura. Grazie alla collaborazione dei programmatorei e al loro contributo alla community, è possibile avere accesso ad una vasta fonte di soluzioni pronte all'uso
- Gestione simultanea delle richieste: Node.js offre un sistema dove le operazioni I/O (Input/Output) risultano non bloccanti, permettendo l'elaborazione di numerose richieste in maniera contemporanea, gestendole mediante una coda, consentendo una notevole rapidità. I vantaggi sono visibili in termini di prestazioni e scalabilità

## Svantaggi di Node.js

- Instabilità delle API: uno dei maggiori svantaggi di Node.js è la mancanza di coerenza. Le API mutano frequentemente e le modifiche sono spesso incompatibili con le versioni precedenti. A causa di ciò, i programmatorei sono costretti ad apportare modifiche al codice esistente, per renderlo compatibile con l'ultima versione rilasciata
- Inadatto per applicazioni di calcolo pesante: Node.js non supporta ancora la programmazione multi-thread. È in grado di servire numerose chiamate diverse, ma non è adatto per eseguire calcoli di lunga durata; questi bloccano le richieste in entrata, il che può portare ad una riduzione delle prestazioni
- Immaturità degli strumenti: nonostante il core di Node.js sia stabile, molti pacchetti nel registro npm (il gestore di pacchetti preinstallato che organizza l'installazione e la gestione di moduli di terze parti) sono ancora di scarsa qualità o non sono stati documentati correttamente. Numerosi strumenti non sono stati supervisionati a causa della natura per lo più open source dell'ecosistema, diminuendo la qualità del sistema [8]

## Compagnie che utilizzano la tecnologia

Node.js è utilizzato da diverse compagnie di particolare rilievo come Netflix, Trello, PayPal, LinkedIn, Ebay, NASA.

## Express

Express è il livello dell'applicazione web che esegue il codice dell'applicazione (JavaScript) di back-end. Questo framework viene eseguito come modulo all'interno dell'ambiente Node.js.

Una delle funzionalità offerte è quella di gestire il routing delle richieste alle

parti corrette dell'applicazione o a diverse app in esecuzione nello stesso ambiente.

È possibile eseguire la logica di business completa dell'app all'interno di Express e persino generare l'HTML finale che deve essere visualizzato dal browser dell'utente. Si può inoltre fornire un punto di accesso, un'API REST, che consente al front-end di dialogare con le risorse necessarie, come il database.

## React

### Introduzione e breve storia

Prima di sviluppare React.js, Facebook si è trovata di fronte ad un'importante dilemma: creare un'interfaccia utente dinamica con prestazioni elevate.

Per riuscirci, la compagnia ha dovuto migliorare il processo di sviluppo e si è deciso di farlo con JavaScript. È stato proposto di inserire XHP, la sintassi del markup di Facebook, nel sistema di coordinate JS. L'idea sembrava impossibile, ma nel 2011 il team di Jordan Walke ha pubblicato la libreria React.js, basata sulla simbiosi JavaScript e XHP. Successivamente, si è realizzato che il prodotto realizzato funzionava più velocemente di qualsiasi altra implementazione del suo genere.

React.js è una libreria che sfrutta la velocità di JavaScript e utilizza un nuovo modo di effettuare il rendering di pagine Web, rendendole altamente dinamiche e reattive all'input dell'utente. Dopo che la libreria è stata rilasciata come strumento open source nel 2013, è diventata estremamente popolare grazie al suo approccio rivoluzionario alla programmazione delle interfacce utente.

### Vantaggi di React

#### DOM Virtuale

Il DOM virtuale di React.js rende l'esperienza utente migliore e il lavoro degli sviluppatori molto più veloce.

DOM (Document Object Model) è una struttura logica di documenti in formato HTML, XHTML o XML. In parole poche, si tratta di un accordo di visualizzazione sugli input e output dei dati, con una forma ad albero.

Il principale dubbio riguardante il tradizionale DOM è rappresentato dal modo in cui elabora le modifiche, ovvero gli input dell'utente, le query e così via. Un server controlla costantemente la differenza causata da questi cambiamenti e fornisce la risposta necessaria. Per rispondere correttamente, ha bisogno di aggiornare gli alberi DOM dell'intero documento, presentando a volte diversi problemi a causa delle grandi dimensioni della struttura.

Il team di React è riuscito ad aumentare la velocità degli aggiornamenti utilizzando il DOM virtuale, una copia astratta di quello reale. Grazie alla sua struttura e all'isolamento dei componenti, vengono ricaricate solo le parti che

devono modificarsi.

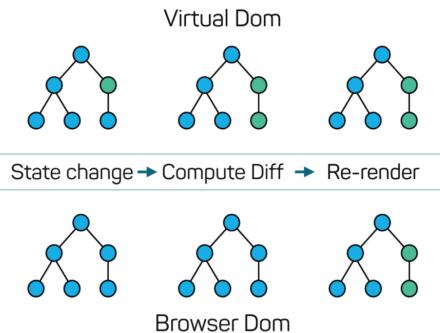


Figura 2.3: Differenza tra DOM reale e DOM virtuale

Sorgente: <https://blog.codecentric.de/en/2017/11/developing-modern-offline-apps-reactjs-redux-electron-part-2-reactjs-basics/>

Questo aumenta notevolmente la rapidità con cui vengono effettuati gli aggiornamenti, consentendo la realizzazione di un'interfaccia utente altamente dinamica. Grazie a ciò, questo approccio ha permesso agli sviluppatori di lavorare più rapidamente con gli oggetti dell'interfaccia utente, consentendo l'hot reloading e applicando le modifiche in tempo reale.

Non sono solo aumentate le prestazioni, ma è stata resa la programmazione più veloce.

### Riutilizzo dei componenti

Questa tecnologia consente il riutilizzo dei componenti, permettendo un notevole risparmio di tempo.

La gestione degli aggiornamenti è facile per gli sviluppatori in React perché tutti i componenti sono isolati e la modifica in uno non influisce sugli altri, permettendo il riutilizzo delle risorse e garantendo una migliore efficienza progettuale.

### Flusso degli aggiornamenti

Il flusso di dati unilaterale in React fornisce stabilità al codice.

React consente un approccio diretto con i componenti e utilizza l'associazione dei dati al ribasso (downward data binding) per far sì che i cambiamenti delle strutture figlie non influiscano sui genitori, rendendo il codice maggiormente stabile.

Per cambiare un oggetto, tutto ciò che uno sviluppatore deve fare è modificare lo stato di un componente e applicare gli aggiornamenti.

## **Libreria open source**

React è una libreria open source, in continua evoluzione e aperta ai contributi della comunità.

React attualmente ha oltre 100'000 stelle su GitHub e presenta oltre 1000 collaboratori che lavorano sulla sua struttura<sup>3</sup>.

## **Svantaggi di React**

### **Alto ritmo di sviluppo**

L'ambiente cambia continuamente ed è necessario imparare i nuovi metodi di approccio alla tecnologia. È un'evoluzione continua e alcuni developers non si sentono a proprio agio nel tenere il ritmo.

### **Documentazione scarsa**

Il problema della documentazione è dovuto ai rilasci costanti di nuovi strumenti. Diverse e nuove librerie come Redux e Reflux promettono di accelerare il lavoro o migliorare l'intero ecosistema di React.

## **Compagnie che utilizzano la tecnologia**

React è utilizzato da diverse compagnie di particolare rilievo come Facebook, Instagram, Netflix, AirBnB, Bloomberg.

## **MongoDB**

MongoDB è un database open source di documenti che permette la persistenza dei dati delle applicazioni ed è progettato per garantire scalabilità ed efficienza. Lo strumento colma il divario tra le coppie chiave-valore, che sono veloci e scalabili, e i database relazionali, che hanno una ricca funzionalità. Invece di memorizzare i dati in righe e colonne come si farebbe con un database relazionale, MongoDB memorizza i documenti con formato JSON, in collezioni con schemi dinamici.

Il modello a documenti agevola la memorizzazione e la combinazione dei dati di qualsiasi struttura, permettendo comunque l'utilizzo di regole di convalida sofisticate, l'accesso flessibile ai dati e le funzionalità di indicizzazione avanzata. È possibile modificare dinamicamente lo schema, requisito necessario per le applicazioni in rapida evoluzione.

MongoDB può essere ridimensionato all'interno e attraverso data center distribuiti geograficamente, garantendo alti livelli di disponibilità e scalabilità. All'aumentare della distribuzione, il database si adatta semplicemente senza tempi di inattività e senza modificare l'applicazione.

---

<sup>3</sup><https://github.com/facebook/react>

## Vantaggi di MongoDB

1. Flessibilità del modello dei dati: a differenza dei database relazionali, i database NoSQL memorizzano facilmente e combinano qualsiasi tipo di dato, strutturato o meno. È successivamente possibile aggiornare dinamicamente lo schema per evolvere con requisiti in continua mutazione
2. Alte prestazioni: i database NoSQL sono costruiti per grandi prestazioni, in termini di throughput e latenza

## Svantaggi di MongoDB

1. Ogni singolo documento è memorizzato insieme al nome dei campi, quindi le dimensioni dei dati potrebbero essere significativamente più elevate
2. Durante il processo di interrogazione, si ha una minore flessibilità, causata ad esempio dalla mancanza dell'operatore JOIN
3. Il modello a singolo documento può essere limitato per alcuni tipi di operazioni atomiche. L'aggregazione presenta delle performance discrete, ma non eccezionali [9]

## Databases relazionali (SQL) vs databases non relazionali (NoSQL)

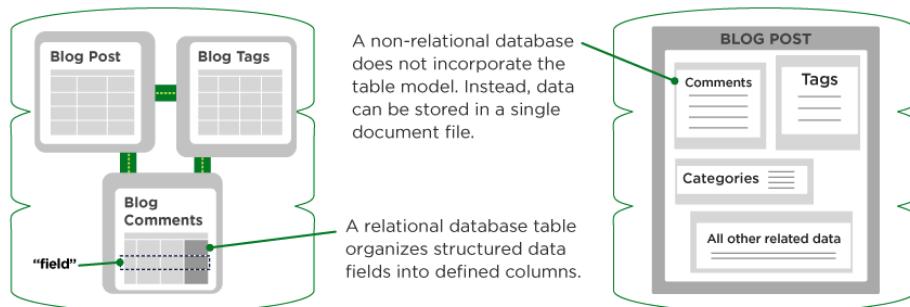


Figura 2.4: SQL vs NoSQL database

Sorgente: <https://www.upwork.com/>

I databases relazionali usano il linguaggio SQL per definire e manipolare i dati: questo è estremamente potente, grazie alla sua flessibilità e al grande utilizzo che ne viene fatto. Tuttavia, può risultare restrittivo, in quanto richiede l'utilizzo di schemi predefiniti per determinare la struttura (unica per tutto il database) dei dati prima di utilizzarli.

I database non relazionali, d'altra parte, hanno uno schema dinamico: grazie a tale caratteristica, è possibile creare documenti prima di conoscere la loro struttura [10].

Per quanto concerne la scalabilità, un database SQL è scalabile verticalmente: è possibile aumentare il carico su un singolo server aumentando le risorse di

sistema.

NoSQL, invece, è scalabile orizzontalmente: è possibile gestire un traffico maggiore attraverso lo sharding, in cui ogni nodo contiene una porzione del database, o aumentando il numero di server a disposizione. Questa soluzione è l'ideale per sistemi che trattano una grande mole di dati o in continua evoluzione.

### **Compagnie che utilizzano la tecnologia**

MongoDB è utilizzato da diverse compagnie di particolare rilievo come Expedia, Bosch, Cisco, Forbes, The Guardian

# Progetto

In questo capitolo verranno trattati gli aspetti salienti riguardanti le funzionalità messe a disposizione dal progetto e la sua architettura.

## Funzionalità

Per soddisfare i requisiti, il progetto mette a disposizione una serie di funzionalità:

- Creazione di algoritmi per l'analisi continua ed efficiente dei dati provenienti dai Cloud Provider interessati
- Esposizione di RESTful API per permettere il salvataggio dei dati in ingresso e l'esposizione di servizi verso l'esterno; gli endpoint verranno sfruttati soprattutto dalla logica di presentazione
- Presentazione immediata ed intuitiva delle tendenze, mediante l'utilizzo di grafici per massimizzare la comprensione delle differenze tra i vari fornitori di servizi cloud
- Supporto alla memorizzazione dei dati
- Fornitura di strumenti pronti all'uso, tramite i quali espandere con il minimo sforzo quello che è il bacino d'interesse su cui operare l'indagine
- Funzionalità di interrogazione del database: le query

## Architettura

Nell'introduzione si è discussa l'importanza di suddividere le responsabilità per ogni area dello stack e questo elaborato tiene in considerazione tali valutazioni sin dall'architettura del progetto.

La Figura 3.1 rappresenta la distribuzione dell'intero sistema:

- Le nuvole azzurre rappresentano i data center attivi: qui sono in funzione le Virtual Machines che raccolgono i dati
- Le nuvole grigie rappresentano alcuni dei data center futuri: in questi verrà iniziata una campagna di raccolta dati
- Il cerchio verde rappresenta il back-end, che elabora e riceve i dati
- Il cerchio rosso rappresenta il front-end, utilizzato dall'utente finale

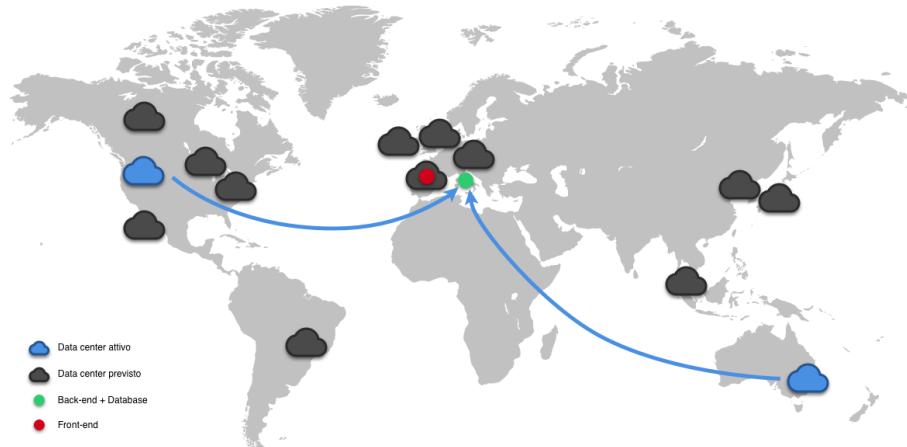


Figura 3.1: Deployment del progetto e distribuzione del sistema. Data center attivi: Oregon e Sydney; front-end situato in Francia, back-end e database situati in Italia

È possibile individuare dunque 5 diversi settori, ognuno dei quali con una specifica competenza:

- Cloud Provider: hosting delle Virtual Machines
- Probing: algoritmo per la raccolta dati
- Persistenza: salvataggio dati su database
- Back-end: logica di business dell'applicazione
- Front-end: presentazione dati

Si aggiunge un'immagine (Figura 3.2) rappresentante l'insieme delle iterazioni che coinvolgono il sistema:

- Le stazioni<sup>1</sup> raccolgono i dati e li inviano al back-end

---

<sup>1</sup>Le stazioni rappresentano le Virtual Machines appartenenti ad un determinato data center, per un particolare Cloud Provider. Esse sono responsabili del probing, ovvero della raccolta dati.

- Il back-end comunica con il database e viceversa; insieme, realizzano le funzionalità di memorizzazione ed analisi dei dati
- Il front-end comunica con il back-end, mediante le API, per utilizzare i servizi esposti; il back-end risponde alla richiesta
- L'utente interagisce con il front-end e riceve riscontro ai suoi input

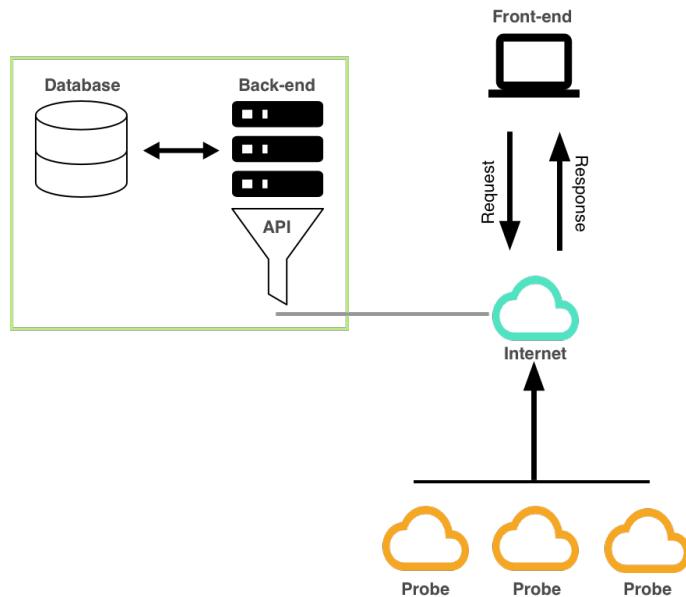


Figura 3.2: Iterazioni del sistema

Infine, la Figura 3.3 rappresenta le varie funzionalità dei vari livelli dell'applicazione.

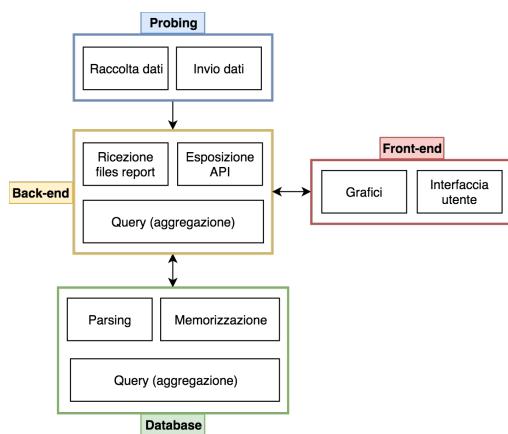


Figura 3.3: Funzionalità del sistema, per ogni livello dell'applicativo

## Probing

In questa sezione verranno approfondite quelle che sono le caratteristiche riguardanti la raccolta dei dati.

Per un'iniziale stima della latenza si è scelto di utilizzare lo strumento più semplice e concreto messo a disposizione da ogni sistema operativo: il ping<sup>2</sup>.

Questo contiene infatti tutte le informazioni atte a svolgere il sondaggio proposto: numero di pacchetti inviati e ricevuti, la loro dimensione, il tempo totale trascorso tra l'invio di ogni pacchetto e la ricezione della risposta corrispondente, la media dei tempi e la percentuale di risposte ottenute.

Successivamente, un algoritmo, preleva da standard output le informazioni sopra descritte e crea un file di formato CSV che identifica le stime sulla latenza di una determinata Virtual Machine, per uno specifico provider, in un preciso giorno dell'anno.

Ogni giornata gli algoritmi raccoglieranno, mediante lo strumento "ping", i dati riguardanti la latenza che caratterizza i diversi data center. Successivamente, alla mezzanotte, i files di report giornalieri verranno inviati ad un server, che li salva localmente e li memorizza in maniera persistente su un database.

Questi dati verranno successivamente elaborati dal back-end.

## Persistenza

Il sistema di persistenza è una delle aree più rilevanti del progetto: il suo compito è quello di salvare, in seguito ad un'operazione di parsing<sup>3</sup>, ogni singolo rapporto proveniente da ogni stazione di raccolta informazioni.

Il notevole volume di dati in entrata con periodicità giornaliera, ha richiesto l'utilizzo di una tecnologia di qualità e la scelta, anche grazie alle caratteristiche dello stack MERN, è ricaduta su MongoDB.

Questo DBMS mette a disposizione una funzionalità di rilevante interesse: l'aggregazione. Essa permette di avere una visione globale dei dati, analizzandoli nel loro insieme e permettendo di ricavare statistiche.

L'aggregazione è oggi una funzionalità essenziale in ambiti come Business Intelligence e Big Data.

MongoDB mette a disposizione tre meccanismi per effettuare l'aggregazione:

- Pipeline di aggregazione
- MapReduce

---

<sup>2</sup>Ping (Packet internet groper) è un'utilità di amministrazione per reti di computer usata per misurare il tempo, espresso in millisecondi, impiegato da uno o più pacchetti ICMP a raggiungere un dispositivo di rete (attraverso una qualsiasi rete informatica basata su IP) e a ritornare indietro all'origine. È prettamente utilizzato per verificare la presenza e la raggiungibilità di un altro computer connesso in rete e per misurare le latenze di trasmissione di rete.

<sup>3</sup>In informatica, il parsing, analisi sintattica o parsificazione, è un processo che analizza un flusso continuo di dati in ingresso (input, letti per esempio da un file o una tastiera) in modo da determinare la sua struttura grazie ad una data grammatica formale

- Metodi di aggregazione per scopi singoli

Il progetto sfrutta la prima modalità, consentendo di riassumere milioni di dati con pochi valori risultanti, per esprimere in maniera sintattica ed efficace quelle che sono le caratteristiche di performance, in termini di latenza, da parte dei vari Cloud Provider.

Tramite l'aggregazione è stato possibile realizzare funzioni specifiche che hanno permesso di sintetizzare le peculiarità di ogni osservato<sup>4</sup>:

- Latenza media in un arco temporale, con confronto tra fornitori: il risultato è il confronto, in termini di latenza media, tra i vari fornitori registrati nel sistema. La ricerca è svolta all'interno degli estremi di date indicate
- Latenza media per un mese e provider selezionato: il risultato è un singolo dato che rappresenta la latenza media che caratterizza il provider, nel mese selezionato, nell'anno corrente

In queste due funzionalità è possibile esprimere se la ricerca debba appartenere allo stesso data center o meno<sup>5</sup>.

Un'ulteriore funzionalità esposta è quella delle query: interrogando il database, è possibile ottenere la lista dei dati che soddisfano un determinato criterio di ricerca.

Le query vengono utilizzate per ottenere un sottoinsieme dei valori iniziali, dando la possibilità all'applicazione di filtrare in base a determinati parametri:

- Soglia su RTT<sup>6</sup>: è possibile selezionare i ping che rientrano in una soglia specificata per il campo indicato
- Intervallo temporale: precisando due date, si possono ottenere tutte le rilevazioni incluse negli estremi
- Provider: si può ottenere la lista dei ping appartenenti ad un determinato fornitore cloud
- Data center: si può ottenere la lista dei ping appartenenti ad un determinato data center di un fornitore cloud

Con il termine query a risultato singolo si indicherà il concetto di aggregazione, le query a risultato multiplo rappresentano invece le tipiche query descritte.

---

<sup>4</sup>Cloud Provider

<sup>5</sup>Con stesso data center si indica il fatto che la stazione mittente e quella destinataria appartengono alla stessa regione; non appartiene allo stesso data center, ad esempio, una rilevazione proveniente dal data center situato in Oregon e diretta al data center situato a Sydney.

<sup>6</sup>Il Round Trip Time o Round Trip Delay (acronimo RTT) è una misura del tempo impiegato da un pacchetto di dimensione trascurabile per viaggiare da un nodo della rete ad un altro e tornare indietro (tipicamente, un'andata client-server ed il ritorno server-client).

## Back-end

Node.js in sinergia con il framework Express esprime tutte le sue potenzialità nel progetto.

Il back-end svolge un ruolo centrale e i principali compiti sono:

- Esposizione di RESTful API<sup>7</sup>: tramite architettura REST vengono esposti servizi invocabili dal front-end oppure dai vari client su cui viene eseguito l'algoritmo di probing
- Comunicazione con database: il back-end comunica con il database e le operazioni di query e di memorizzazione passeranno prima da questo strato
- Ricezione e salvataggio in locale dei files di report
- Manipolazione dati: le query

Il back-end è il fulcro delle funzionalità esposte dall'applicazione: qui verranno scritte in linguaggio JavaScript le principali query che interrogheranno il database.

A causa del notevole volume di dati e in previsione di sviluppi futuri, si è deciso di ricorrere alla tecnica della paginazione delle API.

Grazie ad essa è possibile gestire il risultato da ritornare dato un grande volume di dati e ridurre notevolmente i tempi di risposta, aumentando conseguentemente la qualità dell'esperienza utente.

È possibile inserire un parametro nella richiesta HTTP per indicare il numero di risultati desiderati e tramite un set di contatori, poter specificare l'intervallo dei dati da visualizzare.

## Front-end

Il front-end è lo strato responsabile della presentazione dei dati.

Questo strato si occupa di:

- Definire un'interfaccia utente semplice e minimale
- Creare grafici intuitivi: l'utilizzo di grafici dona immediatezza e maggiore comprensibilità al risultato
- Effettuare richieste asincrone al back-end: è possibile reperire efficacemente le informazioni contattando gli endpoint esposti dal back-end
- Creare una Web Application basata sul concetto di Single Page Application: la riscrittura dinamica della pagina elimina le interruzioni nella navigazione, migliorando l'esperienza finale

---

<sup>7</sup>In informatica, entro un programma, con Application Programming Interface (API) si indica un insieme di procedure (in genere raggruppate per strumenti specifici) attive all'espletamento di un dato compito.

- Creare una Web Application basata sul concetto di Responsive Web Design: i contenuti vengono visualizzati correttamente, indipendentemente dal dispositivo utilizzato

Per sua natura, React permette di scomporre semplicemente l'applicazione in componenti: ogni elemento visualizzato è riutilizzabile ed è dotato di determinate caratteristiche e funzionalità. Così facendo, si verte verso una progettazione bottom-up che ha permesso di semplificare il problema, riducendo gli sforzi necessari per raggiungere il risultato finale.

### L'idea prima dei grafici

Nelle prime fasi del progetto, l'idea era quella di visualizzare la lista dei dati filtrati secondo determinati criteri; tuttavia, dopo una prima implementazione, a cui sono succedute una serie di rifiniture di ottimizzazione, si è scoperto che il valore informazionale offerto non era all'altezza delle aspettative.

Solo in un secondo momento l'idea si è evoluta, diventando l'implementazione attuale: i grafici.

Grazie a questi, mediante l'utilizzo di forme e colori, è possibile cogliere immediatamente quelle che sono le informazioni salienti del risultato ottenuto.

# Implementazione progetto

In questo capitolo verranno mostrate e spiegate le principali strategie di implementazione degli algoritmi dell'applicazione.

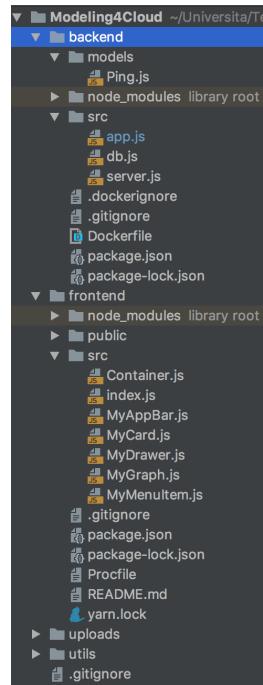


Figura 4.1: Struttura dell'implementazione

## Cloud Provider

I vari fornitori di servizi sono gli interessati dell'osservazione; pertanto, è stato necessario predisporre una molteplicità di Virtual Machines su cui eseguire l'algoritmo per la raccolta dei dati.

Durante l'intera fase di deployment si sono potuti apprezzare quelli che sono i pregi e le peculiarità dei vari provider.

Da notare come sia stato necessario configurare correttamente i gruppi di sicurezza, per permettere il traffico in entrata ed in uscita riguardante le connessioni SSH, le porte utilizzate dall'applicazione e la predisposizione per gli strumenti di monitoraggio.

## Amazon Web Services

Amazon Web Services ha permesso di effettuare un deployment delle Virtual Machines in maniera semplice e concreta.

Le seguenti immagini (Figura 4.2, Figura 4.3, Figura 4.4) rappresentano alcuni dei passi principali necessari per la creazione di un'istanza.

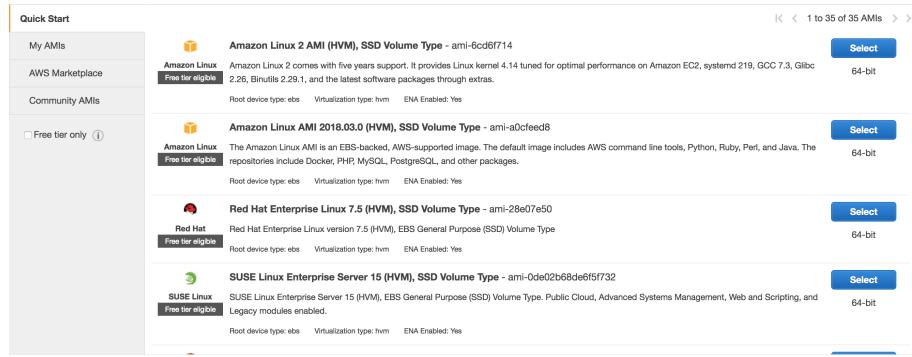


Figura 4.2: Scelta dell'immagine del sistema operativo

| Currently selected: t2.micro (Variable ECUs, 1 vCPU, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only) |                 |            |       |              |                       |                         |                     |              |
|--|-----------------|------------|-------|--------------|-----------------------|-------------------------|---------------------|--------------|
|  | Family          | Type       | vCPUs | Memory (GiB) | Instance Storage (GB) | EBS-Optimized Available | Network Performance | IPv6 Support |
| <input type="checkbox"/>   | General purpose | t2.nano    | 1     | 0.5          | EBS only              | -                       | Low to Moderate     | Yes          |
| <input checked="" type="checkbox"/>  | General purpose | t2.micro   | 1     | 1            | EBS only              | -                       | Low to Moderate     | Yes          |
| <input type="checkbox"/>   | General purpose | t2.small   | 1     | 2            | EBS only              | -                       | Low to Moderate     | Yes          |
| <input type="checkbox"/>   | General purpose | t2.medium  | 2     | 4            | EBS only              | -                       | Low to Moderate     | Yes          |
| <input type="checkbox"/>   | General purpose | t2.large   | 2     | 8            | EBS only              | -                       | Low to Moderate     | Yes          |
| <input type="checkbox"/>   | General purpose | t2.xlarge  | 4     | 16           | EBS only              | -                       | Moderate            | Yes          |
| <input type="checkbox"/>   | General purpose | t2.2xlarge | 8     | 32           | EBS only              | -                       | Moderate            | Yes          |
| <input type="checkbox"/>   | General purpose | t3.nano    | 2     | 0.5          | EBS only              | Yes                     | Up to 5 Gigabit     | Yes          |
| <input type="checkbox"/>   | General purpose | t3.micro   | 2     | 1            | EBS only              | Yes                     | Up to 5 Gigabit     | Yes          |

Figura 4.3: Scelta delle caratteristiche dell'istanza

The screenshot shows the 'Assign a security group' section of the AWS Management Console. It includes fields for 'Security group name' (set to 'launch-wizard') and 'Description' (set to 'launch-wizard'). Below these are five tabs for defining security rules: 'Type' (set to 'SSH'), 'Protocol' (set to 'TCP'), 'Port Range' (set to '22'), 'Source' (set to 'Custom 0.0.0.0/0'), and 'Description' (set to 'e.g. SSH for Admin Desktop'). An 'Add Rule' button is at the bottom left.

Figura 4.4: Definizione dei gruppi di sicurezza

L'installazione dell'algoritmo di raccolta dati è avvenuta con successo: allo stato attuale sono attive 2 Virtual Machines nel data center di Oregon e 1 Virtual Machine nel data center di Sydney.

Non si segnala alcun limite da parte di questo fornitore.

## Google Cloud Platform

Google Cloud Platform si è rilevato altrettanto semplice e concreto nel deployment di Virtual Machines.

Le seguenti immagini (Figura 4.5, Figura 4.6) rappresentano alcuni dei passi principali necessari per la creazione di un'istanza.

The screenshot shows the 'Create instance' wizard on the Google Cloud Platform. The first step is 'Configurazione'. The form fields include:

- Nome:** instance-1
- Area geografica:** us-east1 (Carolina del Sud)
- Zona:** us-east1-b
- Costo stimato:** 24,67 \$ al mese (tariffa oraria effettiva 0,034 \$ / 730 ore al mese)
- Tipo di macchina:** Personalizza (1 vCPU, 3,75 GB di memoria)
- Container:** Implementa l'immagine di un contenitore su questa istanza VM. Ulteriori informazioni
- Disco di avvio:** Nuovo disco permanente standard di 10 GB (Immagine: Debian GNU/Linux 9 (stretch))
- Identità e accesso API:** Account di servizio: Compute Engine default service account
- Ambiti di accesso:** Consent l'accesso predefinito (selected)
- Firewall:** Aggiungi tag e regole firewall per consentire traffico di rete specifico da Internet

Figura 4.5: Scelta dell'immagine e delle caratteristiche dell'istanza. Interessante la funzionalità di stima dei costi sin dai primi passi

The screenshot shows the 'Regole firewall' (Firewall Rules) section in the Google Cloud Platform. At the top, there are buttons for 'CREA REGOLA FIREWALL' (Create Firewall Rule), 'AGGIORNA' (Update), and 'ELIMINA' (Delete). Below this, a note states: 'Le regole del firewall controllano il traffico in entrata o in uscita in un'istanza. Per impostazione predefinita, il traffico in entrata dall'esterno della rete è bloccato.' (Firewall rules control incoming or outgoing traffic in an instance. By default, external network incoming traffic is blocked.) There is also a link to 'Ulteriori informazioni' (More information) and a note about App Engine firewalls. A search bar labeled 'Filtrta risorse' (Filter resources) and a 'Colonne' (Columns) dropdown are present. The main table has columns: Nome (Name), Tipo (Type), Destinazioni (Destinations), Filtri (Filters), Protocolli/porte (Protocols/Ports), Azione (Action), Priorità (Priority), and Rete (Network). A single row is shown for the rule 'default-allow-ssh':

| Nome              | Tipo       | Destinazioni    | Filtri                   | Protocolli/porte | Azione   | Priorità | Rete    |
|-------------------|------------|-----------------|--------------------------|------------------|----------|----------|---------|
| default-allow-ssh | In entrata | Applica a tutte | Intervallo IP: 0.0.0.0/0 | tcp:22           | Consenti | 65534    | default |

Figura 4.6: Definizione dei gruppi di sicurezza

L'installazione dell'algoritmo di raccolta dati è avvenuta con successo: allo stato attuale sono attive 2 Virtual Machines nel data center di Oregon e 1 Virtual Machine nel data center di Sydney.

Non si segnala alcun limite da parte di questo fornitore.

## Microsoft Azure

Il processo di deployment da seguire è simile a quello rappresentato per Amazon Web Services e Google Cloud Platform; tuttavia, la procedura di creazione di una Virtual Machine si è rilevata più prolissa rispetto alla concorrenza.

Il problema principale di questo Cloud Provider è la gestione dei gruppi di sicurezza: non è possibile definire regole nel firewall per i pacchetti ICMP<sup>1</sup>. La regola di default deny non lascia passare alcun pacchetto, nemmeno quelli necessari per effettuare il sondaggio.

```
albertobagnacani@client1:~$ ping 52.
PING 52.                                56(84) bytes of data.
```

Figura 4.7: Test non funzionante del ping

---

<sup>1</sup>In telecomunicazioni e informatica l'Internet Control Message Protocol (ICMP) è un protocollo di servizio per reti a pacchetto che si occupa di trasmettere informazioni riguardanti malfunzionamenti (causati dai primi 8 byte del datagramma IP), informazioni di controllo o messaggi tra i vari componenti di una rete di calcolatori.

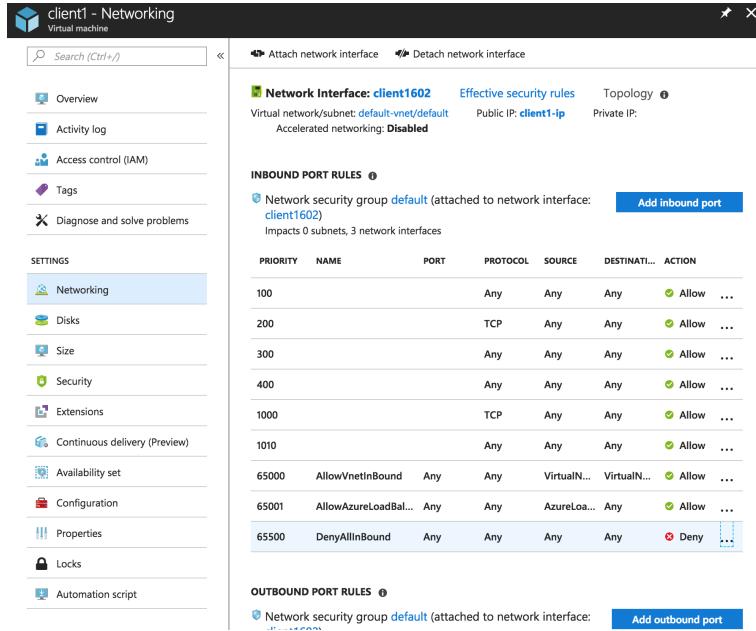


Figura 4.8: Configurazione gruppi di sicurezza, regola di default deny

## IBM Cloud

IBM Cloud non ha dato la possibilità di installare l'algoritmo di raccolta dati: le promozioni riservate agli studenti non si sono rilevate sufficienti per affrontare il costo necessario per la creazione di una Virtual Machine.

IBM, tuttavia, mette a disposizione un servizio gratuito che dà la possibilità di eseguire container.

### I container

I container rappresentano degli spazi-utente isolati [11], in esecuzione sul sistema operativo di un server. Sono funzionalità del S.O. Linux che permettono di virtualizzare alcune risorse di sistema per eseguire applicazioni, in ambienti operativi diversi, all'interno dello stesso server fisico. Invece che ricreare virtualmente tutte le caratteristiche di un computer (come processore, memoria, risorse di rete), come avviene con l'approccio basato sulle Virtual Machines, viene astratto solo l'ambiente di esecuzione delle applicazioni.

I container eseguono direttamente sulla CPU, evitando così apparati di emulazione, permettendo di risparmiare risorse attuando una virtualizzazione a livello del solo sistema operativo.

Non dovendo inglobare tutte le risorse di un server, i container sono più "leggeri" delle macchine virtuali e possono essere attivati tempestivamente. Questa tecnologia può essere sfruttata utilmente nelle situazioni in cui il carico di lavoro da sostenere è variabile nel tempo, con picchi difficilmente prevedibili.

Questa scelta si rivela particolarmente interessante negli ambienti cloud ad alta densità applicativa, con microservizi atti ad avere una vita utile piuttosto breve,

grazie alle buone caratteristiche di automatizzazione dei processi di creazione e riorganizzazione delle risorse.

Per la realizzazione di questa tipologia di deployment sono stati studiati gli strumenti Docker<sup>2</sup> e Kubernetes<sup>3</sup>: è stata creata l'immagine rappresentante l'intero progetto ed è stato eseguito il relativo container.

Tuttavia, l'intero processo stava richiedendo una quantità di tempo non ritenuta opportuna e si è deciso di abbandonare questa strada.

## Probing

Il principio filosofico del Rasoio di Occam afferma: "A parità di fattori, la spiegazione più semplice è da preferire".

Gli algoritmi che si occupano della raccolta e dell'invio dei dati sono scritti in *Bash* e cercano di essere i più concreti e semplici possibili.

Nella Figura 4.9 si mostra il flusso di chiamate che coinvolgono l'area di sondaggio: in blu sono rappresentate le Virtual Machines che operano la raccolta dati, in verde la Web App responsabile di ricevere la lista dei files giornalieri e in rosso i server che ospitano il database MongoDB. La Web App comunicherà con il sistema di persistenza e viceversa.

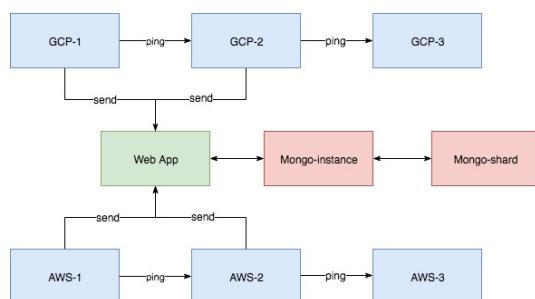


Figura 4.9: Flusso di chiamate che coinvolgono l'area di sondaggio

## registerPing.sh

Questo script prende in input l'indirizzo IP della Virtual Machine da contattare (all'interno dello stesso data center o meno) ed inizia un ping verso tale destinazione; successivamente lo standard output viene analizzato e viene effettuata una prima analisi dei dati: sono prelevati i campi interessanti e scritti con convenzione CSV in un file. Questo presenta un nome in un formato simile ad un timestamp.

```

1 #...
2 if ! [ -d ~/csv ] ; then
3   mkdir ~/csv
  
```

---

<sup>2</sup><https://www.docker.com/>  
<sup>3</sup><https://kubernetes.io/>

```

4 fi
5
6 ping $TOHOST |
7 #...
8 while read LINE; do
9   #...
10  printf "$PROVIDER,$FROMZONE,$TOZONE,$FROMHOST,$TOHOST,$ICMPSEQ,
11    $TTL,$TIME,$TIMESTAMP\n" >> $FILE
12  #...
13 done

```

registerPing.sh

## curlCsv.sh

Lo script in questione sfrutta il tool cURL<sup>4</sup> per inviare i dati al server centrale.

```

1 TODAY=$(date +%Y-%m-%d)
2 #...
3 FILE=~/csv/$PROVIDER-$NUMBER-$YESTERDAY.csv
4
5 curl -F "data=@$FILE" $SERVER

```

curlCsv.sh

## Invio dati giornalieri

Il sistema operativo di tutte le Virtual Machines è Ubuntu; per questa motivazione, è stato utilizzato lo strumento *cron*<sup>5</sup> per l'invio periodico del file giornaliero. Alla mezzanotte, *cron* di ogni stazione di raccolta dati, richiama curlCsv.sh e il file descritto viene inviato correttamente.

Dopo un'attenta analisi delle performance, si è riscontrato che il server centrale, che riceve i files, può sostenere diverse chiamate, quindi non si è dovuto ricorrere a particolari meccanismi (semplicemente si poteva effettuare l'invio in orari diversi) per sopportare il carico di lavoro.

```

ubuntu@ip-172-      :~$ crontab -l
0 0 * * * ~/Modeling4Cloud/utils/curlCsv.sh AWS 1 http://137.2          /api/
upload

```

Figura 4.10: Crontab per il richiamo giornaliero di curlCsv.sh

## Persistenza

Nella fase di implementazione, la persistenza si è rilevata l'area più delicata dell'intero progetto.

BSON (JSON Binario) è il formato dati utilizzato da MongoDB e permette di rappresentare strutture dati semplici e array associativi. BSON può essere

---

<sup>4</sup>cURL è un tool a linea di comando per ottenere o inviare files usando la sintassi URL.

<sup>5</sup>Job scheduler basato sul tempo.

considerato un superset di JSON.

I documenti sono rappresentati da liste ordinate di elementi, in cui ciascuno consiste in un nome di campo (stringa), un tipo e un valore. I tipi principali sono:

- stringhe
- interi (32 o 64 bit)
- double
- date
- booleani
- array BSON
- codice JavaScript

Per quanto riguarda l'efficienza, comparato a JSON, BSON è progettato per essere più efficiente in termini di spazio richiesto dai dati e velocità di ricerca.

Le funzionalità principali che sono state realizzate sono:

- Memorizzazione di tutti i documenti giornalieri
- Manipolazione dei dati attraverso le query

## Memorizzazione dati

Il flusso di iterazioni è il seguente:

1. Il back-end riceve i files dalle varie stazioni di monitoraggio e li salva localmente
2. Il back-end opera per ogni file ricevuto un'azione di parsing e conversione: la necessità è quella di passare dalla notazione CSV a quella JSON (BSON) di MongoDB. Il risultato è un insieme di documenti, dove ognuno rappresenta il ping di ogni singola Virtual Machine
3. Memorizzazione dei documenti da parte del database

## Memorizzazione singolare e multipla

MongoDB mette a disposizione API per l'inserimento singolo o multiplo di documenti.

```
1 db.products.insert({id: 10, item: "box", qty: 20})
```

Esempio inserimento singolo

```

1 db.products.insertMany([
2     {item: "card", qty: 15},
3     {item: "envelope", qty: 20},
4     {item: "stamps", qty: 30}
5 ]);
```

Esempio inserimento multiplo

Dopo una prima implementazione di entrambi i metodi, si è scoperto che le aspettative in termini di performance non venivano soddisfatte: l'inserimento era troppo lento e poteva causare overflow di memoria.

Si è deciso così di ricorrere ad uno strumento più efficiente, messo a disposizione da MongoDB: il mongoimport.

### Parsing e memorizzazione mediante Mongoimport

Mongoimport è uno strumento che importa contenuti da un file Extended JSON, CSV o TSV.

Questo strumento ha permesso di ottenere interessanti performance nelle operazioni di parsing e salvataggio dei documenti entranti ogni giornata.

Il seguente codice rappresenta l'implementazione dell'operazione descritta: è necessario passare il file desiderato, indicando l'host, il database, la collezione, lo username, la password, il tipo di file e i tipi delle colonne.

```

1 mongoimport -h IP -d database -c collection -u albertobagnacani
2 -p password --type csv --columnsHaveTypes
3 --fields "provider.string\(\),from_zone.string\(\),
4 to_zone.string\(\),from_host.string\(\),to_host.string\(\),
5 icmp_seq.int32\(\),ttl.int32\(\),time.double\(\),
6 timestamp.date\(`2006-01-02T15:04:05-00:00`)"
```

Esempio Mongoimport

### Ottimizzazione delle performances

Il livello di persistenza si è rilevato il più critico dell'intero progetto, a causa dell'elevata mole di dati.

L'utilizzo di indici ha permesso di migliorare i tempi di elaborazione del database: sono strutture dati speciali che memorizzano una piccola parte dell'insieme dei valori della raccolta.

Questi [12], permettono di supportare in maniera efficiente l'esecuzione delle queries da parte di MongoDB, altrimenti, il DBMS, dovrebbe effettuare una scansione di tutta la collezione, per verificare che ci sia un match con gli statements indicati. Se un indice appropriato esiste per una determinata query, il database può ridurre il numero di documenti da ispezionare.

L'ordinamento delle voci dell'indice supporta efficaci corrispondenze di uguaglianza e operazioni di query basate su intervalli.

Nonostante l'implementazione di questa funzionalità, l'aggregazione impiega ancora troppo tempo, che non permette di garantire un'esperienza utente soddisfacente.

## Back-end

Il back-end è il cuore delle funzionalità dell'applicazione.

Notevole rilevanza è data ai moduli di Node.js che hanno permesso di semplificare e velocizzare il processo di sviluppo; qui un esempio di alcuni dei più importanti:

- Mongoose: Object Document Mapping (analogo degli ORM<sup>6</sup> per i database non relazionali) per MongoDB, che offre funzionalità che permettono di semplificare la validazione, il casting e la logica di business "boilerplate"
- Body-parser: Node.js HTTP Request body parsing middleware
- Multer: Middleware per la gestione dell'upload dei files
- Express-paginate, Mongoose-paginate: librerie necessarie alla paginazione delle API

### Installazione del Web Server

Node.js, in sinergia con il framework Express, permette con poche linee di codice di lanciare un Web Server, in ascolto su una porta specificata.

```
1 var server = app.listen(port, function(){
2   console.log("Server started at port " + port);
3 });
```

Esempio installazione del Web Server sulla porta desiderata

### Ricezione dei files giornalieri, comunicazione con il database

L'idea è quella di possedere un luogo centralizzato dove raccogliere tutti i files provenienti dalle Virtual Machines.

Questi files, oltre che venire salvati localmente, devono essere analizzati e scomposti nei singoli dati che verranno memorizzati sul database.

### Ricezione dei files

Per realizzare questa funzionalità sono stati utilizzati i moduli *FileSystem* di Node.js e *Multer*.

```
1 const UPLOAD_PATH = '../uploads/';
2 const upload = multer({dest: UPLOAD_PATH});
```

Scelta del percorso dove salvare i files

---

<sup>6</sup>In informatica l'Object-Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS. Un prodotto ORM fornisce tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.

Il seguente codice definirà l'endpoint a cui poter inviare il file interessato; successivamente esso verrà rinominato con una specifica convenzione e verrà restituita la risposta al cliente.

```
1 router.route('/upload').post(upload.single('data'), function (req,
2   res) {
3     fs.rename(UPLOAD_PATH+req.file.filename, UPLOAD_PATH+req.file.
4       originalname, function(err){
5         if (err) return res.status(500).send("Problem in POST\n");
6         res.status(200).send("File registered\n");
7         #...
8       });
9   });
10 }
```

### Comunicazione con il database

Il modulo *Mongoose* permette, tra le varie funzionalità, di connettersi semplicemente ad un database specificato.

```
1 const url = 'mongodb://...';
2 var mongoose = require('mongoose');
3
4 mongoose.connect(url, function(err, db){
5   if (err) console.log('Unable to connect to DB. Error: ' + err);
6   else console.log('Connection established to DB');
7 });


```

### Modellazione del dominio

Il modello del dominio è responsabile di rappresentare il comportamento e i dati di un sistema. Suo compito è quello di raffigurare concetti del mondo reale, spiegando le entità in gioco e le loro relazioni.

Il modello del dominio del progetto è rappresentato principalmente dal ping: il seguente codice permetterà dunque all'ODM *Mongoose* di interfacciarsi con il database:

```
1 var PingSchema = new Schema({
2   provider: String,
3   from_zone: String,
4   to_zone: String,
5   from_host: String,
6   to_host: String,
7   icmp_seq: Number,
8   ttl: Number,
9   time: Number,
10  timestamp: { type : Date, default: Date.now }
11 }).plugin(mongoosePaginate);
```

## Esposizione di endpoint e paginazione delle API

### Esposizione di endpoint

Obiettivo fondamentale del back-end è quello di esporre servizi mediante endpoint che potranno essere contattati dal front-end, per ricevere le informazioni necessarie da visualizzare.

Il framework Express mette a disposizione API per creare un routing, un "in-stradamento" [13].

Con il routing è possibile determinare come un'applicazione risponde ad una richiesta da parte di un client ad un endpoint specifico; questo è composto da un URI (percorso) e un metodo di richiesta HTTP specifico (GET, POST, ecc.).

Ciascuna route può disporre di una o più funzioni dell'handler, le quali sono eseguite nel momento in cui viene trovata una corrispondenza.

La struttura è la seguente:

```
1 app.METHOD(PATH, HANDLER);
```

Dove:

- app è un'istanza di express
- METHOD è il metodo di una richiesta HTTP
- PATH è un percorso sul server
- HANDLER è la funzione da eseguire una volta trovata la corrispondenza per la route

Il seguente codice mostra come sia facilmente possibile creare una route: eseguendo una richiesta GET al server all'indirizzo "http://IP/api/pings", l'handler invocherà la funzione implementata; il front-end potrà, ad esempio, contattare l'endpoint descritto per ottenere la lista JSON di tutti i documenti salvati.

```
1 router.route('/pings').get(async function(req, res, next) {  
2   // Esecuzione codice desiderato  
3 }
```

## Paginazione delle API

La paginazione delle API è una tecnica molto diffusa nei sistemi moderni che lavorano con un grande volume di dati.

L'idea iniziale era quella di presentare tutta la lista dei valori, filtrata secondo determinati criteri. Nonostante venga restituito un sottoinsieme delle informazioni iniziali, non era sensato visualizzarlo unicamente in una pagina.

Si è deciso così di ricorrere ad una tecnica più sofisticata, che permette di manipolare semplicemente ed efficacemente la suddivisione in pagine dei risultati ritornati: la paginazione delle API.

Il seguente codice descrive come è possibile ottenere la lista paginata dei documenti, a seguito di una richiesta HTTP GET al back-end.

La prima linea di codice permette di specificare il limite di default e il limite massimo di documenti che il cliente può richiedere. Successivamente avviene la vera e propria paginazione, grazie ai moduli *Express-paginate* e *Mongoose-paginate*; viene ritornata la lista JSON con informazioni aggiuntive sulle pagine desiderate.

```

1 app.use(paginate.middleware(100, 200));
2
3 router.route('/pings').get(async function(req, res, next) {
4     try {
5         const [ results, itemCount ] = await Promise.all([
6             Ping.find().limit(req.query.limit).skip(req.skip).lean()
7                 ().exec(),
8             Ping.find().count({})
9         ]);
10
11         const pageCount = Math.ceil(itemCount / req.query.limit);
12
13         if (req.accepts('json')) {
14             res.json({
15                 object: 'list',
16                 number_of_items: results.length,
17                 total_number_of_items: itemCount,
18                 total_number_of_pages: pageCount,
19                 has_more_pages: paginate.hasNextPages(req)(pageCount)
20
21             });
22         } catch (err) {
23             next(err);
24         }
25     });

```

Le seguenti immagini (Figura 4.11, Figura 4.12, Figura 4.13) rappresentano concretamente il concetto descritto.

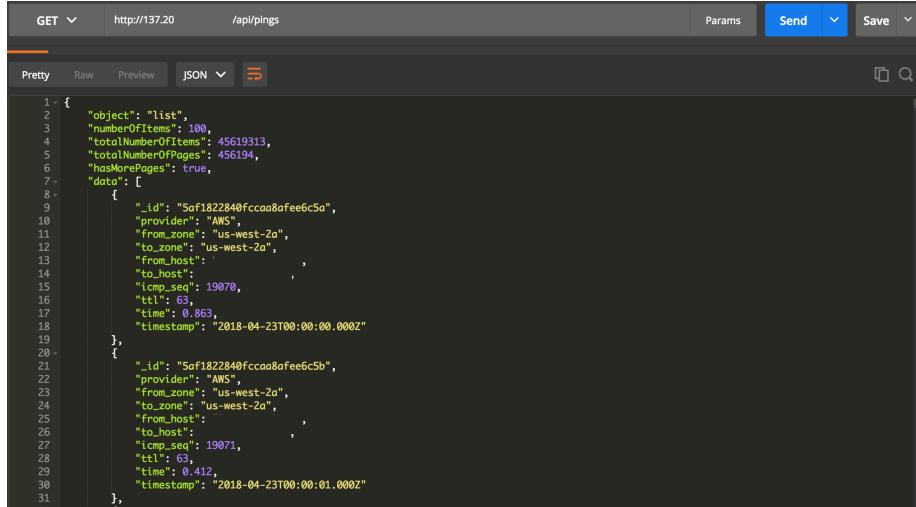


Figura 4.11: Ottenimento della prima pagina dei dati

```

1 {
2   "object": "list",
3   "numberOfItems": 100,
4   "totalNumberOfItems": 43262074,
5   "totalNumberOfPages": 432621,
6   "hasMorePages": true,
7   "data": [
8     {
9       "_id": "5af1825d40fccaa8afeff343",
10      "provider": "AWS",
11      "from_zone": "us-west-2a",
12      "to_zone": "us-west-2a",
13      "from_host": "...",
14      "to_host": "...",
15      "icmp_seq": 53435,
16      "ttl": 63,
17      "time": 0.847,
18      "timestamp": "2018-04-24T03:45:01.000Z"
19    },
20    {
21      "_id": "5af1825d40fccaa8afeff344",
22      "provider": "AWS",
23      "from_zone": "us-west-2a",
24      "to_zone": "us-west-2a",
25      "from_host": "...",
26      "to_host": "...",
27      "icmp_seq": 53438,
28      "ttl": 63,
29      "time": 0.715,
30      "timestamp": "2018-04-24T03:45:04.000Z"
31    }
32  ]
33}

```

Figura 4.12: Ottenimento di una precisa pagina dei dati (pagina 1000)

```

1 {
2   "object": "list",
3   "numberOfItems": 74,
4   "totalNumberOfItems": 43262074,
5   "totalNumberOfPages": 432621,
6   "hasMorePages": false,
7   "data": [
8     {
9       "_id": "5b8322ebeae85a3fcf494403",
10      "provider": "GCP",
11      "from_zone": "us-west1-a",
12      "to_zone": "australia-southeast1-a",
13      "from_host": "...",
14      "to_host": "...",
15      "icmp_seq": 15147,
16      "ttl": 64,
17      "time": 161,
18      "timestamp": "2018-08-26T23:58:46.000Z"
19    },
20    {
21      "_id": "5b8322ebeae85a3fcf494404",
22      "provider": "GCP",
23      "from_zone": "us-west1-a",
24      "to_zone": "australia-southeast1-a",
25      "from_host": "...",
26      "to_host": "...",
27      "icmp_seq": 15148,
28      "ttl": 64,
29      "time": 161,
30      "timestamp": "2018-08-26T23:58:47.000Z"
31    }
32  ]
33}

```

Figura 4.13: Ottenimento dell'ultima pagina dei dati

## Manipolazione dati: query

Con il termine query si indica l'interrogazione da parte di un utente di un database, per compiere determinate operazioni sui dati (selezione, inserimento, cancellazione, aggiornamento, ecc.).

Le query realizzate sono principalmente di due tipologie: quelle a risultati multipli e quelle a risultato singolo (aggregazione). Essendo lo strato di back-end molto legato a quello di persistenza, si inserisce l'implementazione di questa parte in questa sezione.

### Query a risultato multiplo

Queste query permettono di ritornare una lista ordinata di documenti, filtrati sulla base di determinati parametri forniti in input.

Le funzionalità realizzate sono quelle descritte nel capitolo di progetto; si riporta un esempio di codice che implementa la feature che permette di ritornare tutti i valori appartenenti al provider indicato; è possibile indicare il numero di risultati che si desidera ottenere (limit) e saltare (skip).

```
1 router.route('/pings/query/provider').get(async (req, res, next) =>
2   {
3     try {
4       var provider = (req.query.provider) ? req.query.provider :
5         'AWS';
6
7       // Esecuzione della query
8       const [ results, itemCount ] = await Promise.all([
9         Ping.find({ provider: provider }).limit(req.query.limit)
10        .skip(req.skip).lean().exec(),
11         Ping.find({ provider: provider }).count({})
12       ]);
13
14       // Inizio della paginazione
15       const pageCount = Math.ceil(itemCount / req.query.limit);
16
17       if (req.accepts('json')) {
18         res.json({
19           object: 'list',
20           number_of_items: results.length,
21           total_number_of_items: itemCount,
22           total_number_of_pages: pageCount,
23           has_more_pages: paginate.hasNextPages(req)(pageCount)
24             ,
25             data: results
26           });
27       }
28     } catch (err) {
29       next(err);
30     }
31   }
32 );
```

### Query a risultato singolo (aggregazione)

Le query di aggregazione permettono di ritornare, dato un grande volume di dati, un singolo risultato; questa tipologia è quella che il front-end utilizzerà

maggiormente.

La seguente query permetterà di ottenere, dato un mese nell'anno corrente e un provider, la media di tutti i ping (latenza) per ogni giorno del mese indicato, per il provider selezionato. Si specifica inoltre la volontà che i risultati appartengano o meno allo stesso data center.

```
1 router.route('/pings/query/monthAggregation').get(async (req, res,
2   next) => {
3     var month, provider, sameRegion;
4
5     month = parseInt(req.query.month);
6     provider = req.query.provider;
7     sameRegion = parseInt(req.query.sameRegion);
8
9     Ping.aggregate()
10       .allowDiskUse(true)
11       .match({provider: myProvider})
12       .project({month: {$month: "$timestamp"}, sameRegion: {$cmp:
13         ['$from_zone', '$to_zone']}, timestamp: "$timestamp",
14         time: "$time"})
15       .match({$and: [{month: myMonth}, {sameRegion: mySameRegion
16         }]})
17       .group({_id: {"$dayOfYear": "$timestamp"}, avg: {$avg: "
18         $time"}})
19       .sort({_id: 1})
20       .exec(function (err, resp) {
21         if (err) {
22           // ...
23           console.log(err);
24         } else {
25           res.json(resp);
26         }
27       })
28     });
29   });
```

Da notare l'utilizzo obbligatorio dell'istruzione `.allowDiskUse(true)`: MongoDB ha un limite di 100MB per le la memoria RAM utilizzata da una fase della sua pipeline [14]; infatti, per permettere la gestione di un notevole volume di dati è necessario includere la linea di codice indicata. L'operazione descritta permette al database di scrivere file temporanei.

## Front-end

Il front-end rappresenta lo stadio finale dell'intera applicazione, quello che verrà presentato all'utente e con cui esso interagirà.

React, con il suo modo di pensare, si è dimostrato uno strumento particolarmente potente nell'intero processo di implementazione.

In questa sezione verranno trattati due aspetti principali: interfaccia utente e grafici.

## Interfaccia utente

L'obiettivo principale era quello di realizzare un'interfaccia utente semplice, reattiva ed intuitiva.

*Material-UI* è un framework compatibile con React per la creazione di interfacce utente in stile Material Design<sup>7</sup>.

Le seguenti immagini (Figura 4.14, Figura 4.15, Figura 4.16, Figura 4.17) mostrano un esempio della grafica finale dell'interfaccia utente dell'applicazione.

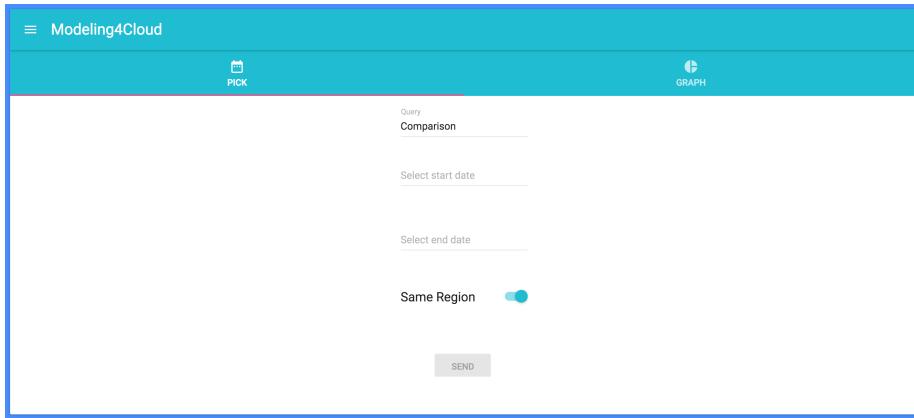


Figura 4.14: Pagina principale del sito

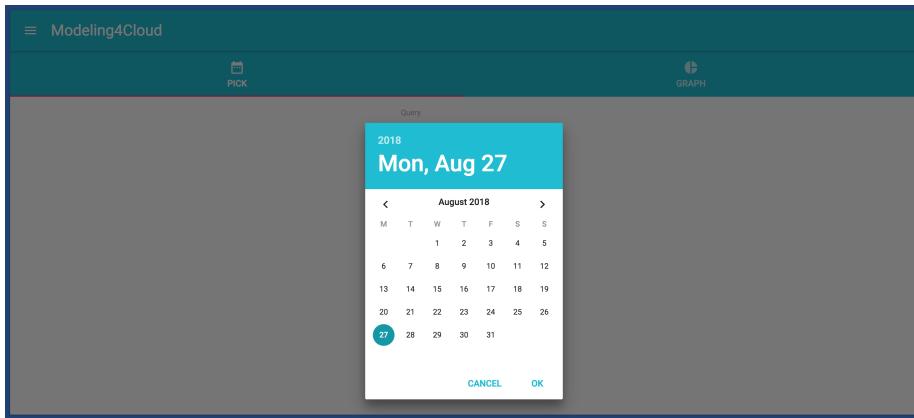


Figura 4.15: Widget per la scelta della data

---

<sup>7</sup>Il Material Design è un design sviluppato da Google che si concentra su uso di layout basati su una griglia, animazioni e transizioni ed effetti di profondità come illuminazione ed ombre.

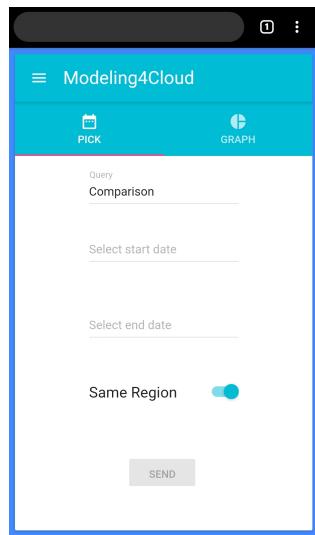


Figura 4.16: Pagina principale del sito da mobile

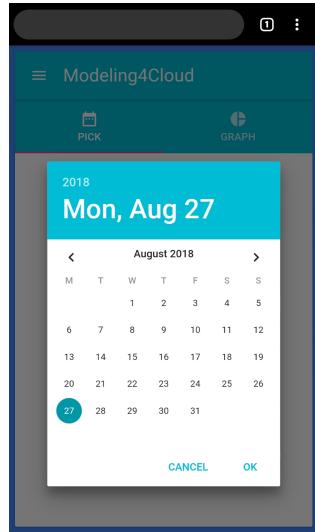


Figura 4.17: Widget per la scelta della data da mobile

## Implementazione dell'interfaccia utente

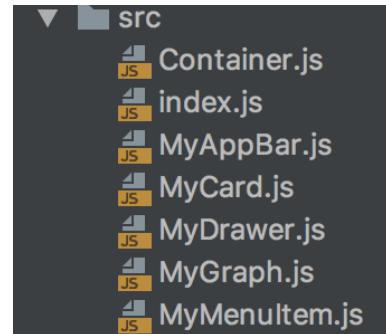


Figura 4.18: Struttura e componenti del front-end

React, per sua natura, favorisce il pensiero della logica a componenti, rendendoli riutilizzabili per altri progetti.

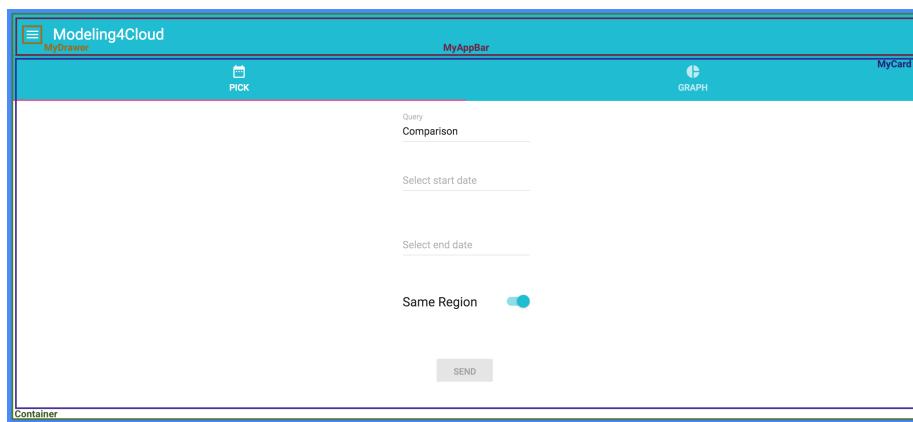


Figura 4.19: Evidenziazione dei componenti principali della struttura

La seguente immagine specifica il flusso di chiamate all’invocazione della pagina principale index.js.

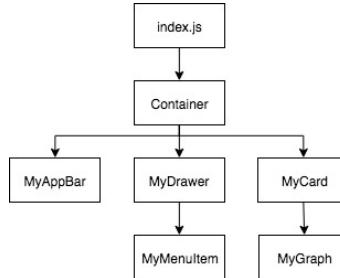


Figura 4.20: Flusso delle chiamate del front-end

Il container è responsabile di visualizzare il contenuto della pagina: esso contiene lo stato di apertura/chiusura del menù e visualizza la barra superiore, il menù e il corpo centrale della pagina; ad ognuno di essi vengono passate determinate proprietà e i gestori di funzione. Quest’ultimi determinano un’azione in seguito ad un evento.

```

1 export default class Container extends React.Component {
2   //...
3   render() {
4     return (
5       <div>
6         <MyAppBar
7           title="Modeling4Cloud"
8           iconClassNameRight="muidocs-icon-navigation-
9             expand-more"
10          onLeftIconButtonClick={this.handleToggle}
11        />
12        <MyDrawer
13          docked={false}
14          width={200}
15          open={this.state.drawerOpened}
16          onRequestChange={(drawerOpened) => this.
17            setState({drawerOpened})}
18          onMenuItemClick={this.handleClose}
19        >
20          <MyCard/>
21        </div>
22      );
23    }
  
```

## Grafici

I grafici arricchiscono l’esperienza utente, aggiungendo espressività e immediatezza al risultato.

La loro realizzazione ha richiesto l’ausilio di *react-chartjs-2*, un contenitore della nota libreria *chartjs*.

Il metodo *render* di ogni componente di React ha il compito di visualizzare le

informazioni grafiche; questo, nella classe *MyCard*, avrà la responsabilità principale di creare un’istanza di un nuovo grafico, passandogli come proprietà i dati che questo dovrà visualizzare.

```

1 //...
2 render(){
3   return(
4     //...
5     <div>
6       {this.state.buttonClicked ? <MyGraph data={this.state.
7         graphData} graphType={this.state.graphType}/> : null}
8     </div>
9   );
}

```

Metodo *render* della classe *MyCard*: creazione dell’istanza di *MyGraph*

```

1 //...
2 render(){
3   return(
4     //...
5     <div>
6       <HorizontalBar data={this.props.data} width={spec.width}
7         height={spec.height} options={spec.options} legend={spec.legend} redraw/> : null
8     </div>
9   );
}

```

Metodo *render* della classe *MyGraph*: creazione e visualizzazione del grafico

## Comunicazione mediante gli endpoint con il back-end

Il front-end contatta gli endpoint messi a disposizione da parte del back-end.

Una volta scelta la funzionalità, vengono riempiti i campi e viene abilitata la possibilità di eseguire l’operazione: al click del bottone, un gestore delle funzioni contatterà il back-end e riceverà i dati elaborati da visualizzare:

```

1 <RaisedButton label="Send" secondary={true} disabled={this.state.
  buttonDisabled} onClick={this.handleClick}/>

```

Agganciamento del gestore delle funzioni all’azione di ”click” sul bottone

```

1 handleClick = (event) =>{
2   this.callApi()
3     .then(res => {
4       //...
5       // Imposto lo stato del componente con i dati
6       // ricevuti dal back-end
7       this.setState({graphData: graphDataModified,
8         graphType: graphType, buttonClicked: true,
9         redraw: true});
10      //...
11    })
12  }

```

Gestore delle funzioni

```

1  async callApi(){
2      var query = (this.state.queryNumber == 1)
3          ? 'http://IP/api/pings/query/dateQuery?start=' + moment(
4              this.state.start).format('YYYY-MM-DD') + '&end=' +
5                  moment(this.state.end).format('YYYY-MM-DD') + '&
6                      sameRegion=' + ((this.state.sameRegion === true) ? 0
7                          : 1)
8          : 'http://IP/api/pings/query/monthAggregation?month=' +
9              this.state.month + '&sameRegion=' + ((this.state.
10                 sameRegion === true) ? 0 : 1) + '&provider=' + this.
11                     state.provider;
12
13     // Chiamata al back-end e ricezione della risposta
14     const response = await fetch(query);
15     const body = await response.json();
16
17     return body;
18 }
```

Chiamata al back-end passando i dati inseriti ed opportunamente formattati.  
Ottenimento della risposta

# Risultati sperimentali

In questo capitolo vengono effettuate considerazioni sul software realizzato: sono analizzate le performance, la capacità di scalare del sistema ed i suoi limiti. Per verificare l'efficacia del nostro progetto sono state effettuate alcune sessioni di test, che riuscissero ad evidenziare le reali prestazioni del prototipo realizzato.

Viene indicato l'hardware di cui si dispone: CPU AMD Opteron 2,3 GHz dual core, 4 GB memoria RAM, 40 GB HDD, sistema operativo Centos.

## Analisi back-end

### Salvataggio, parsing ed importazione dei dati sul database

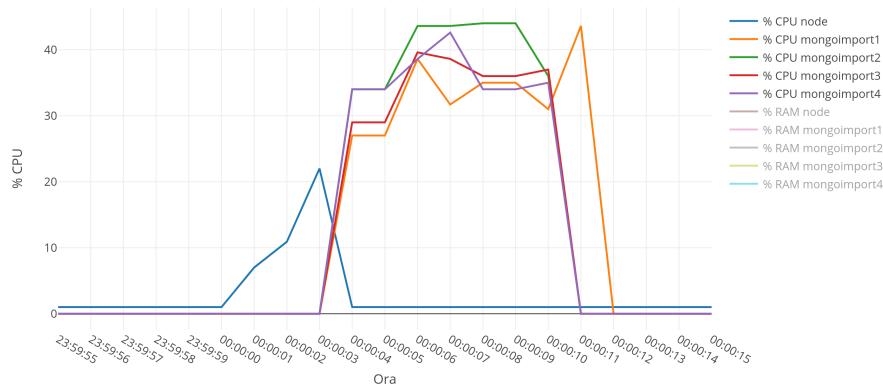


Figura 5.1: Utilizzo della CPU da parte dei processi "node" e "mongoimport"

La Figura 5.1 rappresenta la percentuale di utilizzo della CPU da parte dei processi di:

1. salvataggio dei file, rappresentato in legenda da "% CPU node"
2. parsing e memorizzazione dei dati di ogni file giornaliero su MongoDB, rappresentato in legenda da "% CPU mongoimport"

Alla mezzanotte di ogni giorno, le stazioni inviano il report giornaliero al server centrale, che ospita il back-end; le considerazioni riguardano il caso attualmente presente, con 4 probe (2 su Amazon Web Services e 2 su Google Cloud Platform) che inviano i dati raccolti. All'arrivo dei files, Node.js inizia l'opera di salvataggio sul disco fisso, che si conclude completamente nell'arco di 4 secondi, utilizzando al massimo il 20% della CPU.

Successivamente viene svolta l'opera di parsing e di memorizzazione dei singoli dati sul database: viene dedicato un processo ad ogni file ricevuto. L'intera operazione, eseguita in parallelo, perdura per un totale di circa 8 secondi, con un utilizzo massimo del 45% della CPU da parte di ogni processo.

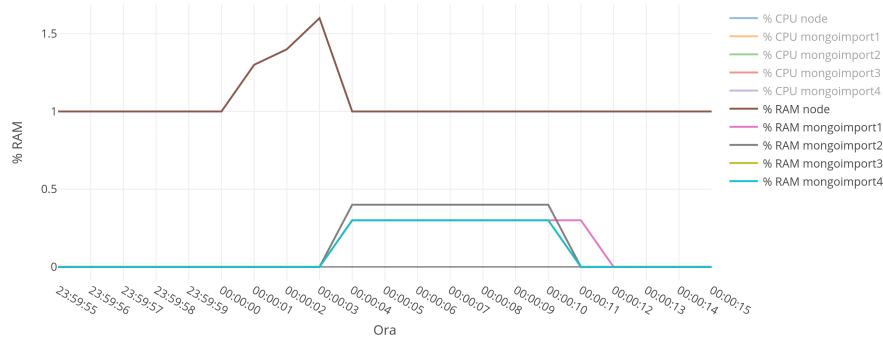


Figura 5.2: Utilizzo della RAM da parte dei processi "node" e "mongoimport"

La Figura 5.2 rappresenta analogamente la percentuale di utilizzo della memoria RAM. È possibile notare l'uso irrisorio di questa risorsa da parte dei processi.

Grazie a queste analisi è possibile comprendere come l'aggiunta di ulteriori Virtual Machines impatti sull'intero applicativo; in particolare, tramite semplici grafici, sono state evidenziate la capacità di scalare di questa parte del sistema, le sue potenzialità ed i suoi limiti.

Nel caso si ritenga opportuno effettuare delle ottimizzazioni, può essere sufficiente distribuire l'invio dei files in diversi orari della giornata.

### Richieste operate agli endpoint da parte del front-end

Node.js è molto efficiente nella gestione delle richieste da parte dei clienti: operando diverse chiamate (più di 10) parallele da parte del front-end, l'utilizzo delle risorse di sistema si è rivelato totalmente irrisorio.

## Analisi persistenza

A causa del numero elevato di documenti che il database ospita (47,5 milioni allo stato attuale), destinato ad aumentare, le query rappresentano un importante problema in termini di performance.

## Impiego delle risorse di sistema

In Figura 5.3 è possibile notare l'utilizzo delle risorse da parte del processo di MongoDB, per eseguire l'aggregazione sulla mole di dati.

1. Il punto rosso rappresenta la prima richiesta di esecuzione da parte di un cliente: viene utilizzato al massimo il 100% della CPU. L'utilizzo della memoria RAM rimane invariato
2. Il punto verde rappresenta invece 2 o più chiamate parallele di esecuzione da parte di più clienti: viene utilizzato al massimo il 200% della CPU (come intuibile, in quanto si dispone di una CPU dual core). L'utilizzo della memoria RAM rimane invariato
3. Il punto giallo rappresenta il termine dell'intera elaborazione

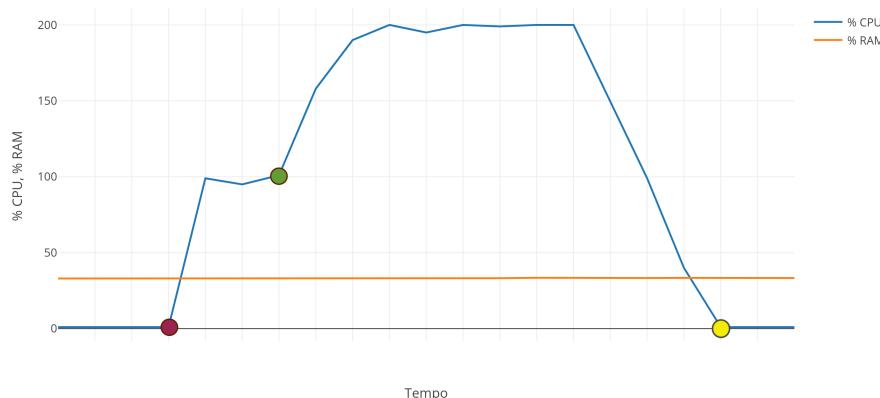


Figura 5.3: Utilizzo delle risorse da parte di MongoDB

È possibile dunque individuare che ad un numero di richieste contemporanee al sistema di persistenza, vengono utilizzate completamente le risorse riguardanti la CPU; conseguentemente, aumenta il tempo di risposta impiegato dal sistema di persistenza.

## Tempo impiegato dall'aggregazione

Il tempo impiegato dall'aggregazione è il limite principale dell'intero progetto: a causa di questo problema, non si riesce a garantire all'utilizzatore finale un'esperienza utente soddisfacente. La maggior parte delle query, infatti, impiega almeno 3 minuti e mezzo per concludersi.

L'implementazione degli indici ha consentito di ridurre notevolmente i tempi di risposta, permettendo al back-end di restituire il risultato. Senza questa tecnica, le politiche di MongoDB farebbero sì che al cliente non sia presentato nulla: esiste un time-out (valutato in 6 minuti), dopo il quale il driver back-end di MongoDB chiude la comunicazione con il sistema di persistenza.

## Sharding

Lo sharding [15] è un metodo per distribuire i dati su più macchine. MongoDB utilizza questa tecnica per supportare la ripartizione di set di valori molto grandi e operazioni ad elevato throughput.

I sistemi di database che presentano volumi di dati o applicazioni ad elevato throughput, possono mettere alla prova la capacità di un singolo server. Ad esempio, alte percentuali di query possono esaurire la CPU del server, mentre dimensioni di working set superiori alla RAM del sistema, stressano la capacità di I/O dei dischi rigidi.

Esistono due metodi per affrontare la crescita del sistema: ridimensionamento verticale e orizzontale.

Lo scaling verticale comporta l'aumento delle risorse di un singolo server.

Lo scaling orizzontale comporta la divisione e il caricamento dei documenti del sistema su più server. Sebbene la velocità complessiva di una singola macchina non sia elevata, ognuna gestisce un sottoinsieme del carico di lavoro totale, offrendo potenzialmente una migliore efficienza rispetto ad un singolo server con elevate prestazioni. L'espansione delle capacità di distribuzione richiede solamente l'aggiunta di server secondo necessità, con un costo inferiore rispetto all'hardware di fascia alta per una singola macchina. Il compromesso è una maggiore complessità nell'infrastruttura e nella manutenzione.

Questa tecnica è sicuramente la soluzione al problema presentato: sarebbe possibile ridurre i tempi di risposta del database ed analizzare il comportamento del sistema all'aumentare del numero di dati, in proporzione al numero di shard.

## Probing

La raccolta dati è rapida e l'invio si svolge nell'arco di un paio di secondi. Un ulteriore pregio è rappresentato dalla **facilità** con cui è possibile iniziare una nuova campagna di raccolta dati per un diverso provider o data center, ampliando il bacino di analisi: è sufficiente creare una Virtual Machine, impostare i gruppi di sicurezza con le porte che il sistema utilizza ed installare il programma di sondaggio.

## Front-end

L'interfaccia grafica è reattiva, fluida e gradevole e le funzionalità esposte sono interessanti.

Le applicazioni Web progressive sono un ulteriore punto a favore, permettendo di vivere l'esperienza anche da dispositivo mobile.

## Dati raccolti e futuri

È impressionante il volume di documenti con cui il sistema lavora: la raccolta dati iniziata nella 3<sup>a</sup> settimana di Aprile fino al momento attuale della scrittura, ha apportato un totale di 47,5 milioni di documenti.

È facile predire e calcolare la crescita del database: ogni stazione registra 1 ping al secondo, per un totale di 86'400 valori al giorno.

Con 4 sistemi di sondaggio attivi (2 per Amazon Web Services e 2 per Google Cloud Platform), si raggiungono le 345'600 entries giornaliere.

Per ogni Cloud Provider è prevista la raccolta dati all'interno dello stesso data center e all'esterno di esso.

Per quanto concerne le dimensioni dei singoli file, riguardante la lista dei ping giornalieri di una stazione, le unità di grandezza sono le seguenti: 68 bytes la riga d'intestazione e 101 bytes per le altre righe del file CSV. È possibile dunque calcolare un peso di circa 6,2KB al minuto, 8,93MB al giorno, 267,8MB al mese e 3,21GB all'anno.

Si inseriscono alcune immagini (Figura 5.4, Figura 5.5, Figura 5.6) che permettono di intuire la crescita del database in relazione a diversi fattori.

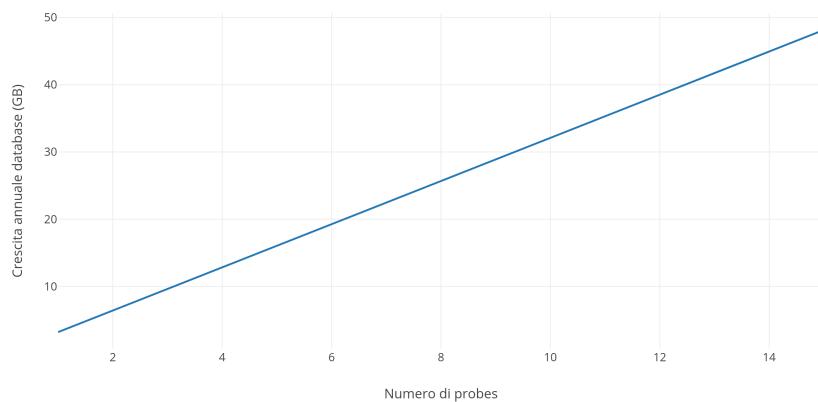


Figura 5.4: Grafico che mostra la crescita delle dimensioni del database in relazione al numero di probes installate

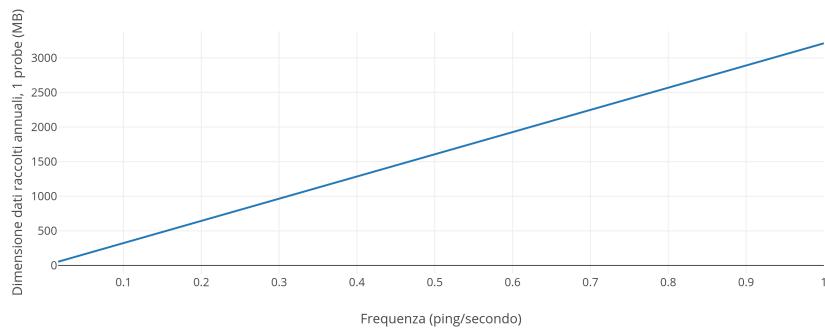


Figura 5.5: Grafico che mostra la crescita delle dimensioni annuali del database, con una probe attiva, in relazione alla frequenza di ping

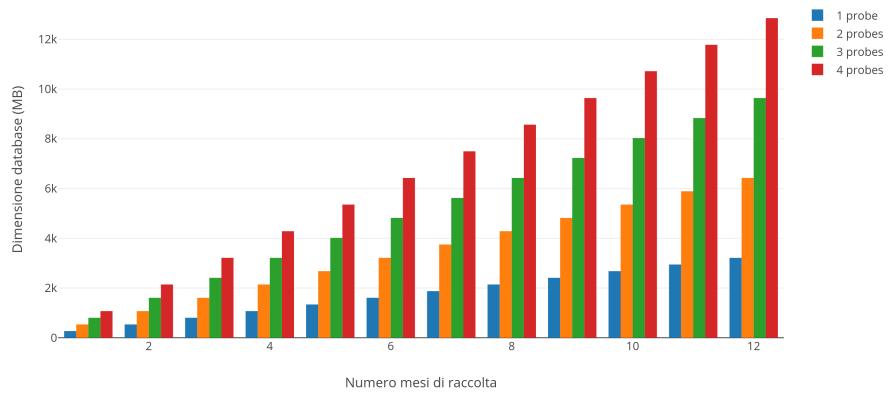


Figura 5.6: Grafico che mostra la crescita delle dimensioni del database, con una o più probes attive, in relazione al numero di mesi di raccolta dati

## Analisi dei risultati

Si riportano i risultati ottenuti dai grafici, in seguito alla raccolta e all'elaborazione dei dati. Sono qui osservati i valori riguardanti la latenza media che coinvolge la coppia di data center Oregon-Sydney, per i fornitori di servizio Amazon Web Services e Google Cloud Platform. Sulle ascisse si trovano i giorni, numerati a partire dall'inizio dell'anno e sulle ordinate i valori medi della latenza (in ms). Si lasciano le considerazioni al lettore.

Maggio 2018

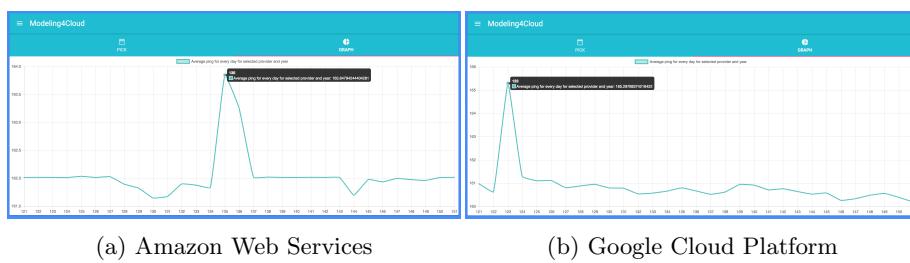


Figura 5.7: Confronto tra i fornitori per il mese di Maggio

Giugno 2018



Figura 5.8: Confronto tra i fornitori per il mese di Giugno

## Luglio 2018



Figura 5.9: Confronto tra i fornitori per il mese di Luglio

## Agosto 2018

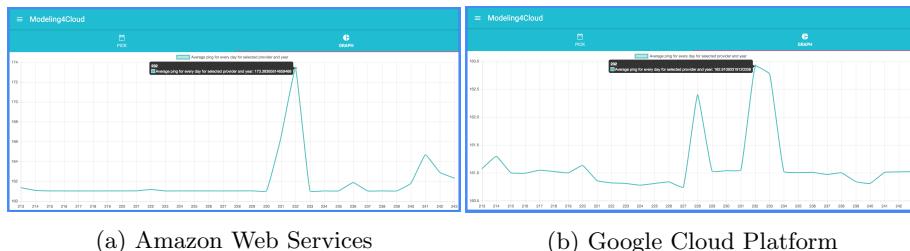


Figura 5.10: Confronto tra i fornitori per il mese di Agosto

## Risultato finale del progetto

Il front-end è lo strato finale con cui l'utente viene a contatto ed è spontaneo che diventi il principale metro di paragone di un progetto: si inseriscono quindi diverse immagini, complete di grafici e spiegazione, che riassumono un esempio di utilizzo tipico dell'applicativo (Figura 5.11, Figura 5.12, Figura 5.13, Figura 5.14, Figura 5.15, Figura 5.16).

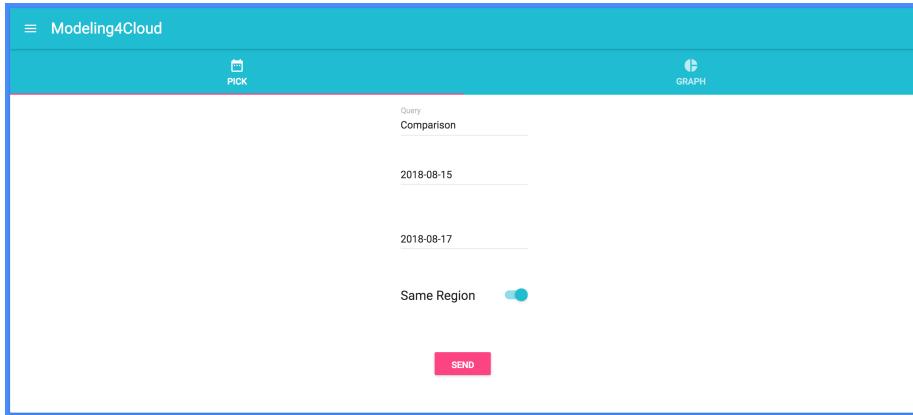


Figura 5.11: Inserimento dati per la funzionalità di comparazione della latenza tra provider: sono immesse le date di inizio e di fine dell’intervallo di tempo interessato e si indica la necessità che il risultato appartenga solo a stazioni che risiedono nello stesso data center



Figura 5.12: Risultato dell’analisi con evidente confronto tra i fornitori, mostrato mediante grafici. Sulle ascisse la latenza media (in ms) e sulle ordinate la lista dei provider

The screenshot shows the 'Modeling4Cloud' search interface. At the top, there are three tabs: 'PICK' (selected), 'GRAPH', and 'SEARCH'. Below the tabs, there are four input fields: 'Query' (empty), 'Year' (empty), 'Day' (containing '7'), and 'Provider' (containing 'AWS'). A 'Same Region' toggle switch is turned off. At the bottom right is a red 'SEND' button.

Figura 5.13: Inserimento dati per la funzionalità di ricerca in un determinato mese dell'anno in corso, per un determinato provider. Si indica la necessità che il risultato appartenga solo a stazioni che non risiedono nello stesso data center

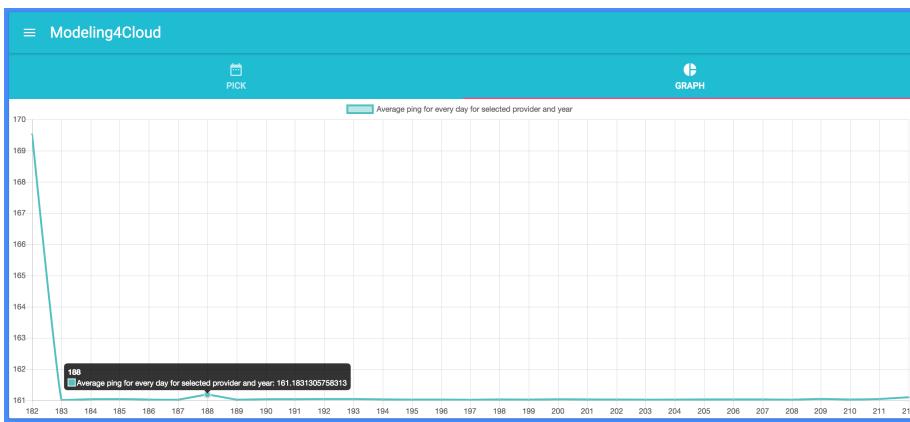
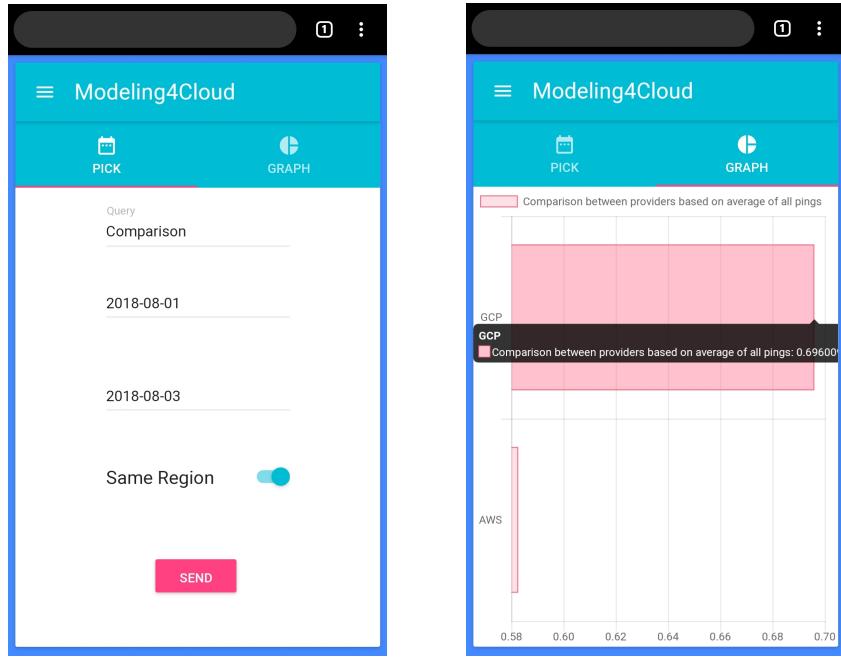


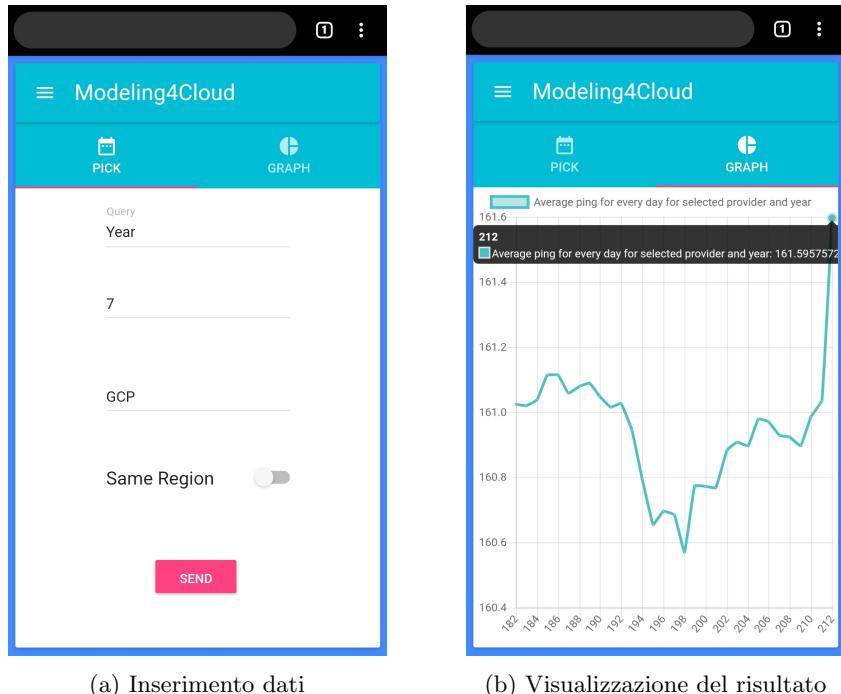
Figura 5.14: Risultato dell'analisi con evidente valutazione sulla stabilità del fornitore, mostrato mediante grafici. Sulle ascisse il numero del giorno a partire dall'inizio dell'anno e sulle ordinate la latenza media (in ms)



(a) Inserimento dati

(b) Visualizzazione del risultato

Figura 5.15: Funzionalità di comparazione tra provider: esperienza da mobile



(a) Inserimento dati

(b) Visualizzazione del risultato

Figura 5.16: Funzionalità di visualizzazione della stabilità per un determinato provider: esperienza da mobile

# Conclusioni

Lo scopo del progetto è quello di raccogliere, memorizzare, elaborare e presentare i ritardi di rete che caratterizzano i diversi data center, distribuiti in tutto il mondo, per ogni Cloud Provider.

Il risultato è un sistema full-stack funzionante, che sfrutta tecnologie allo stato dell'arte, innovative, che hanno permesso di raggiungere l'obiettivo prefissato. Le sonde collocate nei vari data center raccolgono ogni giorno centinaia di migliaia di valori, migliorando il modello di analisi della latenza di comunicazione e perfezionando l'accuratezza e l'affidabilità dei risultati e delle previsioni future. Viene dunque creata un'unica fonte di verità per i ritardi di rete del modello ad inter-data center, per tutti i fornitori di servizi Cloud. Questa permetterà di monitorare a runtime e non solo, quelle che sono le latenze, e consentirà, per esempio, di analizzare e migliorare il deployment dei vari componenti di un'infrastruttura.

L'introduzione specificava come l'opera di monitoraggio dovesse coinvolgere 4 Cloud Provider: Amazon Web Services, Google Cloud Platform, Microsoft Azure e IBM Cloud.

Come è però possibile notare, le analisi finali sono state operate solo sui primi 2 fornitori indicati.

Gli sviluppi futuri si concentreranno nel capire come installare l'algoritmo di raccolta dati per il provider Microsoft Azure, che non ha permesso di analizzare i ritardi di rete che caratterizzano i suoi data center, a causa delle politiche riguardanti il firewall. Strumenti come *psping*<sup>1</sup>, *netperf*<sup>2</sup> e *qperf*<sup>3</sup> permettono di ottenere informazioni aggiuntive oltre a quelle specificate nel progetto; inoltre, alcuni di essi, consentono di utilizzare il protocollo TCP invece che ICMP per contattare le altre stazioni: grazie a tale soluzione il problema potrebbe essere risolto.

Per quanto concerne IBM Cloud, si cercherà di implementare l'utilizzo dei container.

È necessario inoltre inserire nel front-end un menù a tendina, in cui poter selezionare il provider e i data center interessati per l'analisi: nel caso si volesse aggiungere un'ulteriore stazione di raccolta dati, per un diverso data center, è necessario implementare questa funzionalità che permette di filtrare i risultati.

---

<sup>1</sup><https://docs.microsoft.com/en-us/sysinternals/downloads/psping>

<sup>2</sup><https://github.com/HewlettPackard/netperf>

<sup>3</sup><https://linux.die.net/man/1/qperf>

È infine stato spiegato come il tempo impiegato da MongoDB, per l'esecuzione delle query sulla notevole mole di dati, non sia accettabile.  
L'utilizzo della tecnica di sharding è probabilmente la soluzione a questo problema e verrà implementata nelle versioni successive dell'applicazione.

Lo scienziato descrive ciò che esiste; l'ingegnere crea ciò che non era mai stato.  
~ *Theodore von Kármán*

# Elenco delle figure

|      |   |    |
|------|---|----|
| 1.1  | Il Cloud Computing . . . . .  | 6  |
| 1.2  | La storia del Cloud Computing . . . . .   | 7  |
| 1.3  | Tipi di servizio del Cloud Computing . . . . .  | 9  |
| 1.4  | Funzionamento del load balancer . . . . .   | 12 |
| 2.1  | Lo stack MERN . . . . .   | 13 |
| 2.2  | Funzionamento di Node.js . . . . .  | 17 |
| 2.3  | Differenza tra DOM reale e DOM virtuale . . . . .   | 20 |
| 2.4  | SQL vs NoSQL database . . . . .   | 22 |
| 3.1  | Deployment del progetto e distribuzione del sistema. Data center attivi: Oregon e Sydney; front-end situato in Francia, back-end e database situati in Italia . . . . . | 25 |
| 3.2  | Iterazioni del sistema . . . . .  | 26 |
| 3.3  | Funzionalità del sistema, per ogni livello dell'applicativo . . . . .   | 26 |
| 4.1  | Struttura dell'implementazione . . . . .  | 31 |
| 4.2  | Scelta dell'immagine del sistema operativo . . . . .  | 32 |
| 4.3  | Scelta delle caratteristiche dell'istanza . . . . .   | 32 |
| 4.4  | Definizione dei gruppi di sicurezza . . . . .   | 32 |
| 4.5  | Scelta dell'immagine e delle caratteristiche dell'istanza. Interessante la funzionalità di stima dei costi sin dai primi passi . . . . .                                | 33 |
| 4.6  | Definizione dei gruppi di sicurezza . . . . .   | 34 |
| 4.7  | Test non funzionante del ping . . . . .   | 34 |
| 4.8  | Configurazione gruppi di sicurezza, regola di default deny . . . . .  | 35 |
| 4.9  | Flusso di chiamate che coinvolgono l'area di sondaggio . . . . .  | 36 |
| 4.10 | Crontab per il richiamo giornaliero di curlCsv.sh . . . . .   | 37 |
| 4.11 | Ottenimento della prima pagina dei dati . . . . .   | 43 |
| 4.12 | Ottenimento di una precisa pagina dei dati (pagina 1000) . . . . .  | 44 |
| 4.13 | Ottenimento dell'ultima pagina dei dati . . . . .   | 44 |
| 4.14 | Pagina principale del sito . . . . .  | 47 |
| 4.15 | Widget per la scelta della data . . . . .   | 47 |
| 4.16 | Pagina principale del sito da mobile . . . . .  | 48 |
| 4.17 | Widget per la scelta della data da mobile . . . . .   | 48 |
| 4.18 | Struttura e componenti del front-end . . . . .  | 49 |
| 4.19 | Evidenziazione dei componenti principali della struttura . . . . .  | 49 |
| 4.20 | Flusso delle chiamate del front-end . . . . .   | 50 |

|      |  |    |
|------|--|----|
| 5.1  | Utilizzo della CPU da parte dei processi "node" e "mongoimport"  | 53 |
| 5.2  | Utilizzo della RAM da parte dei processi "node" e "mongoimport"  | 54 |
| 5.3  | Utilizzo delle risorse da parte di MongoDB . . . . .   | 55 |
| 5.4  | Grafico che mostra la crescita delle dimensioni del database in relazione al numero di probes installate . . . . .   | 57 |
| 5.5  | Grafico che mostra la crescita delle dimensioni annuali del database, con una probe attiva, in relazione alla frequenza di ping . .  | 58 |
| 5.6  | Grafico che mostra la crescita delle dimensioni del database, con una o più probes attive, in relazione al numero di mesi di raccolta dati . . . . .   | 58 |
| 5.7  | Confronto tra i fornitori per il mese di Maggio . . . . .  | 59 |
| 5.8  | Confronto tra i fornitori per il mese di Giugno . . . . .  | 59 |
| 5.9  | Confronto tra i fornitori per il mese di Luglio . . . . .  | 60 |
| 5.10 | Confronto tra i fornitori per il mese di Agosto . . . . .  | 60 |
| 5.11 | Inserimento dati per la funzionalità di comparazione della latenza tra provider: sono immesse le date di inizio e di fine dell'intervallo di tempo interessato e si indica la necessità che il risultato appartenga solo a stazioni che risiedono nello stesso data center . | 61 |
| 5.12 | Risultato dell'analisi con evidente confronto tra i fornitori, mostrato mediante grafici. Sulle ascisse la latenza media (in ms) e sulle ordinate la lista dei provider . . . . .  | 61 |
| 5.13 | Inserimento dati per la funzionalità di ricerca in un determinato mese dell'anno in corso, per un determinato provider. Si indica la necessità che il risultato appartenga solo a stazioni che non risiedono nello stesso data center . . . . .                              | 62 |
| 5.14 | Risultato dell'analisi con evidente valutazione sulla stabilità del fornitore, mostrato mediante grafici. Sulle ascisse il numero del giorno a partire dall'inizio dell'anno e sulle ordinate la latenza media (in ms) . . . . .   | 62 |
| 5.15 | Funzionalità di comparazione tra provider: esperienza da mobile  | 63 |
| 5.16 | Funzionalità di visualizzazione della stabilità per un determinato provider: esperienza da mobile . . . . .  | 63 |

# Bibliografia

- [1] W. Cerroni, L. Foschini, G. Y. Grabarnik, L. Shwartz, and M. Tortonesi, “Estimating delay times between cloud datacenters: A pragmatic modeling approach.” <https://ieeexplore.ieee.org/document/8187641>, 2018.
- [2] Wikipedia, “Cloud computing — wikipedia, l’enciclopedia libera.” [https://it.wikipedia.org/wiki/Cloud\\_computing](https://it.wikipedia.org/wiki/Cloud_computing), 2018. [Online; controllata il 1-sett-2018].
- [3] amazon.com, “Cos’è il cloud computing?.” <https://aws.amazon.com/it/what-is-cloud-computing/>, n.d.
- [4] isplus.it, “La storia del cloud computing in un’infografica.” <http://www.isplus.it/cloud-blog/75-la-storia-del-cloud-computing-in-un-infografica>, 2014.
- [5] antonioangelino.com, “Cloud computing: cos’è e quali sono vantaggi e svantaggi.” <http://antonioangelino.com/cloud-computing-cosa-e-vantaggi-svantaggi/>, n.d.
- [6] microsoft.com, “Cos’è il cloud computing?.” <https://azure.microsoft.com/it-it/overview/what-is-cloud-computing/>, n.d.
- [7] A. Morgan, “The modern application stack – part 1: Introducing the mean stack.” <https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack>, 2017.
- [8] N. Chrzanowska, “Why to use node.js: Pros and cons of choosing node.js for back-end development.” <https://www.netguru.co/blog/pros-cons-use-node.js-backend>, 2017.
- [9] mongodb.com, “Nosql databases pros and cons.” <https://www.mongodb.com/scale/nosql-databases-pros-and-cons>, n.d.
- [10] Xplenty, “The sql vs nosql difference: Mysql vs mongodb.” <https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2>, 2017.
- [11] fastweb.it, “Cloud container: cosa sono e come funzionano.” <https://www.fastweb.it/grandi-aziende/news/articolo/cloud-container-cosa-sono-e-come-funzionano/>, 2017.
- [12] mongodb.com, “Indexes strategies.” <https://docs.mongodb.com/manual/indexes/>, n.d.

- [13] expressjs.com, “Routing di base.” <http://expressjs.com/it/starter/basic-routing.html>, n.d.
- [14] mongodb.com, “Aggregation pipeline limits.” <https://docs.mongodb.com/manual/core/aggregation-pipeline-limits/#agg-memory-restrictions>, n.d.
- [15] mongodb.com, “Sharding.” <https://docs.mongodb.com/manual/sharding/>, n.d.
- [16] altexsoft.com, “The good and the bad of reactjs and react native.” <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/>, 2017.
- [17] gerardnico, “React - shared state (lifting state up).” [https://gerardnico.com/lang/javascript/react/shared\\_state](https://gerardnico.com/lang/javascript/react/shared_state), 2018.