Cloud Computing

AWSS - DNAs substring matching

A.A. 2021/2022

Docenti:
Simone Merlini
Nicolò Marchesi

Membri del team:
Alessandro Capici
Guglielmo Cassini
Danilo Modica
Domenico Ragusa

Scopo del prodotto

Realizzazione di un sistema per il calcolo della sottostringa di DNA più lunga in comune tra due stringhe di DNA.
Tale sistema sarà basato su un'architettura cloud e deve poter prevedere il suo utilizzo da parte di più utenti contemporaneamente.

In particolare, tra gli altri requisiti, lo sviluppo della web application deve essere fatto in modo tale che sia:

- Distribuita, resiliente e scalabile
- Testata
- Versionata
- con CI/CD

In aggiunta (opzionali):

- costruita secondo il paradigma dell'Infrastructure as Code (IaC)
- facente uso di pattern architetturali avanzati

Responsabilità

Assegnazione responsabilità di progetto:

- **Alessandro Capici** => Docker, ECS/ECR, runECSTask lambda, C-Test
- **Guglielmo Cassini** => Docker, ECS/ECR, runECSTask lambda, LCS
- **Danilo Modica** => API Gateway, Lambda, Python Tests, Web Interface (JavaScript)
- **Domenico Ragusa** => Route53, Cloudfront, OpenSearch, SQS, CI/CD, Web Interface (HTML/CSS)

Le restanti sezioni (documentazione, presentazione, studio di architettura e requisiti…) sono state svolte in sessioni di lavoro comune.

Modello di sviluppo

Il modello di sviluppo utilizzato è stato quello **incrementale**

Gli sprint seguiti erano così organizzati:

1. Prima analisi requisiti, architettura embrionale con EC2 on demand

2. Interfaccia web, Cloudfront e Route53 , container Docker

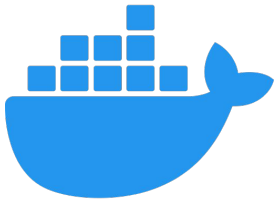
3. Code, SQS, notifiche con Lambda

4. Logging dell'infrastruttura (Cloudwatch/OpenSearch)

5. Privacy e security tramite policy di accesso e WAF

6. CI/CD

Tecnologie



freenom

Documentazione

1. Requisiti
2. Casi d’uso
3. Modello di dominio

Requisiti

Principali requisiti da implementare:

- **RF01**: New computation
- **RF03**: Storage of results
- **RNF04**: Automatic deletion of results

ID	**RF01**
Name	**New computation**
Definition	Customers launching the web application will have the ability to perform the calculation, given two strings of DNA, of a substring with the longest common base sequence
Affected by	Contents of the web application homepage
Actors	Customer

ID	**RF03**
Name	**Storage of results**
Definition	The results that are produced should be stored for future reference as well and must be identified by a unique code
Affected by	Type of the storage chosen and unique identifier generation system
Actors	System

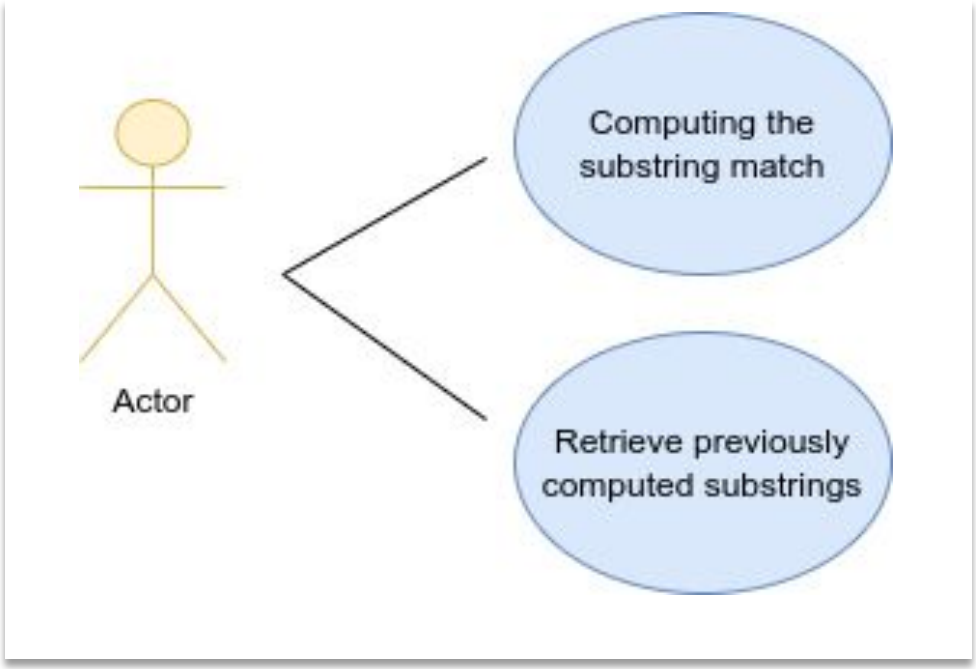
ID	**RNF04**
Name	**Automatic deletion of results**
Definition	A daily check will allow input and/or output files to be automatically deleted when they may be considered obsolete
Motivation	To avoid wasting resources

Lista completa nella documentazione GitHub

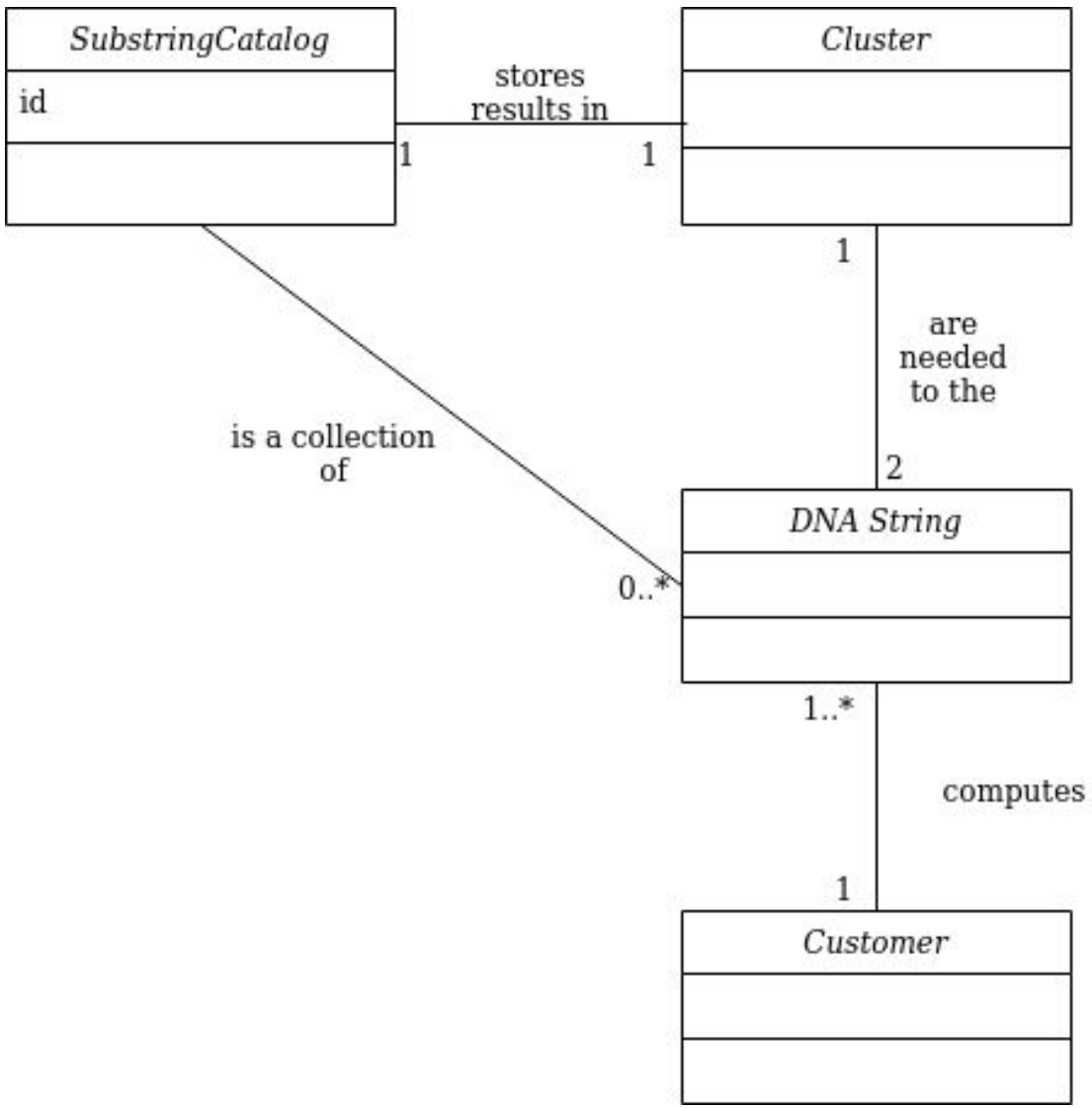
Casi d’uso

Lista e diagramma dei casi d’uso

- **UC1D**: Computing the substring
- **UC1B**: On-demand download of previously calculated substrings



Modello di dominio



Progettazione

1. Algoritmo LCS
2. Architettura del sistema
3. Interfaccia Web
4. CI/CD

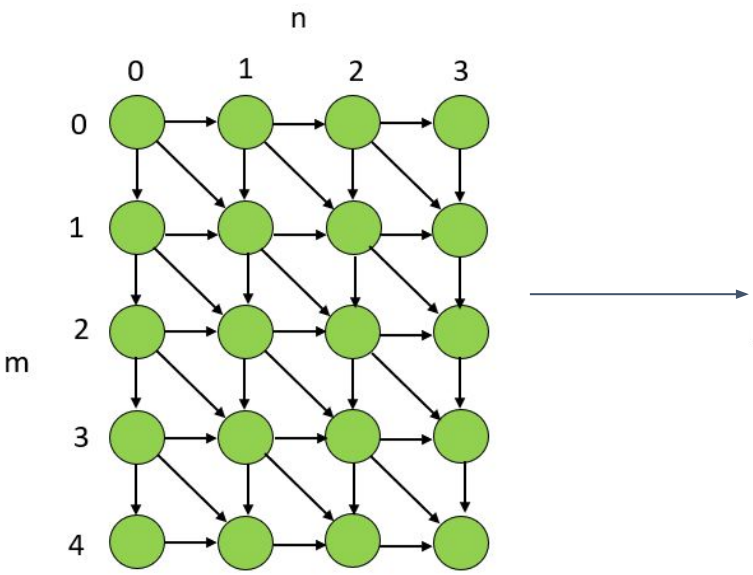
Algoritmo LCS

Longest common subsequence: una stringa w è definita come la più lunga e comune sottosequenza tra N stringhe S_1,..., S_N se è una sottosequenza di tutte le stringhe S_1,..,S_N ed è la più lunga tra le possibili stringhe di sottosequenza.

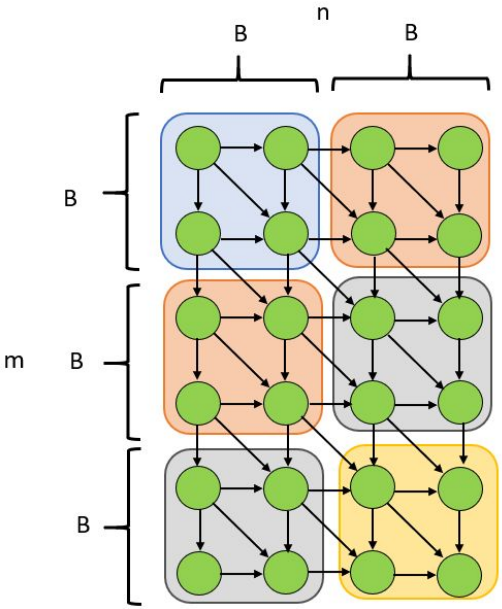
$$lcs(x_1 \ldots x_i, y_1 \ldots y_j) = \begin{cases} 1 + lcs(x_1 \ldots x_{i-1}, y_1 \ldots y_{j-1}) & \text{se } x_i = y_j \\ max(lcs(x_1 \ldots x_i, y_1 \ldots y_{j-1}), \\ \quad lcs(x_1 \ldots x_{i-1}, y_1 \ldots y_j)) & \text{se } x_i \neq y_j \end{cases}$$

→ $C_{ij} = lcs(i,j)$

Dependency graph



Parallelization



Architettura del sistema

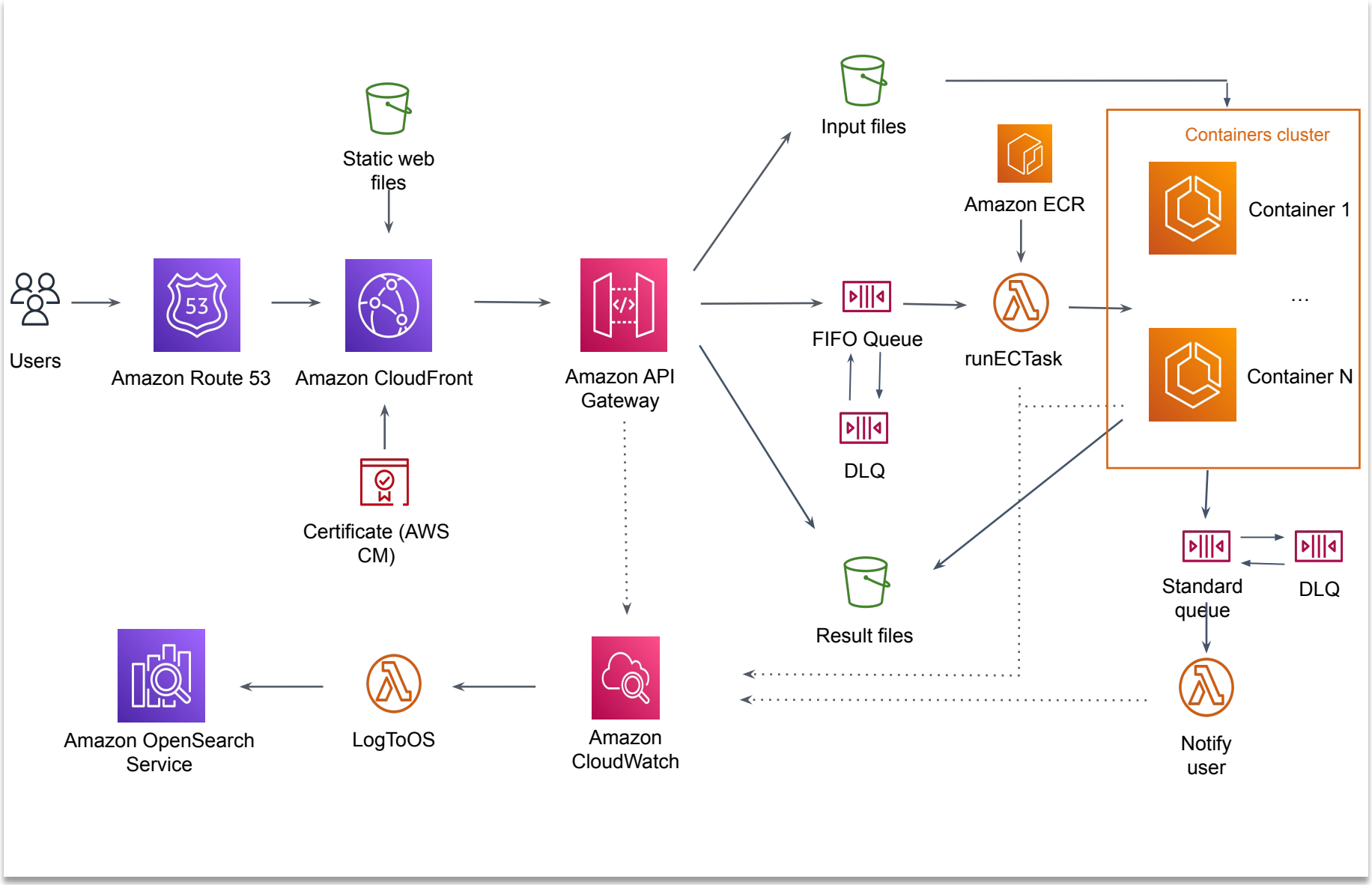
Il sistema è basato sul modello **Infrastructure as a Service**.

Nell'architettura del sistema, in particolare, si distinguono due strati:

1. **Strato di presentazione**
2. **Strato di elaborazione e gestione dati**

Alla base dell'architettura vi è un'**infrastruttura cloud AWS**, la quale può scalare facilmente sia verticalmente che orizzontalmente.

Architettura AWS



Front-end

Amazon Route 53

- Hosted zone creata manualmente
- Record cloudfront A/AAAA e alias CNAME www creati come IaC
- Healthcheck collegato ad allarme Cloudwatch e SNS Topic

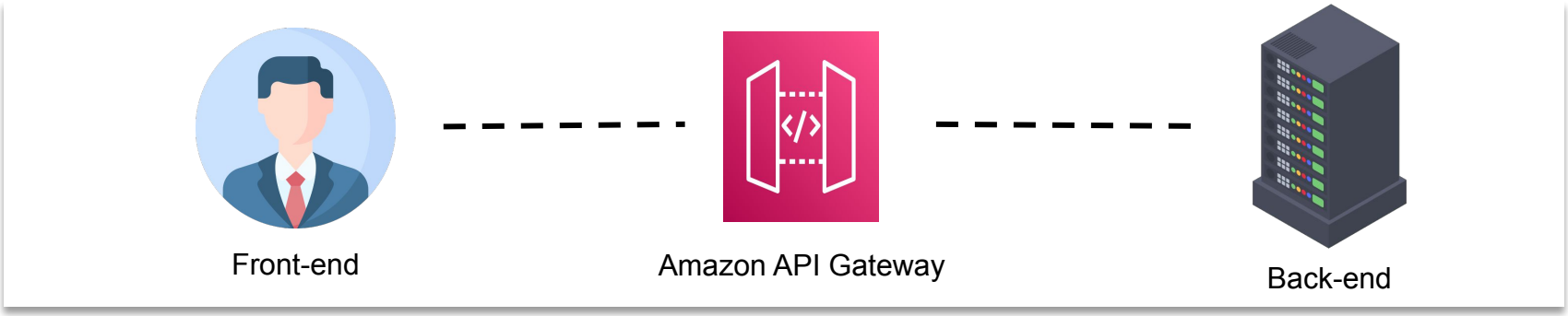
- Uso di tutte le edge location per migliorare il ping (opzione *best performance*)
- Certificato SSL/TLS per abilitare l'uso del protocollo https
- Cache di 1 ora per i contenuti
- *Origin Access Identity* e *Origin Shield* per l'isolamento del bucket con i file statici

Amazon CloudFront

www.awss

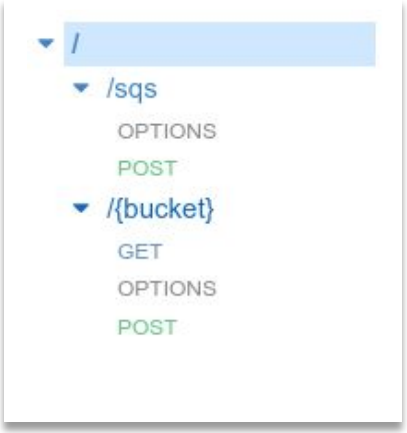
- Static website hosting
- Accessibile soltanto da Cloudfront per il deploy dei contenuti sulle edge locations
- Policy CORS che consente solo richieste GET

API Gateway



Due funzioni principali:

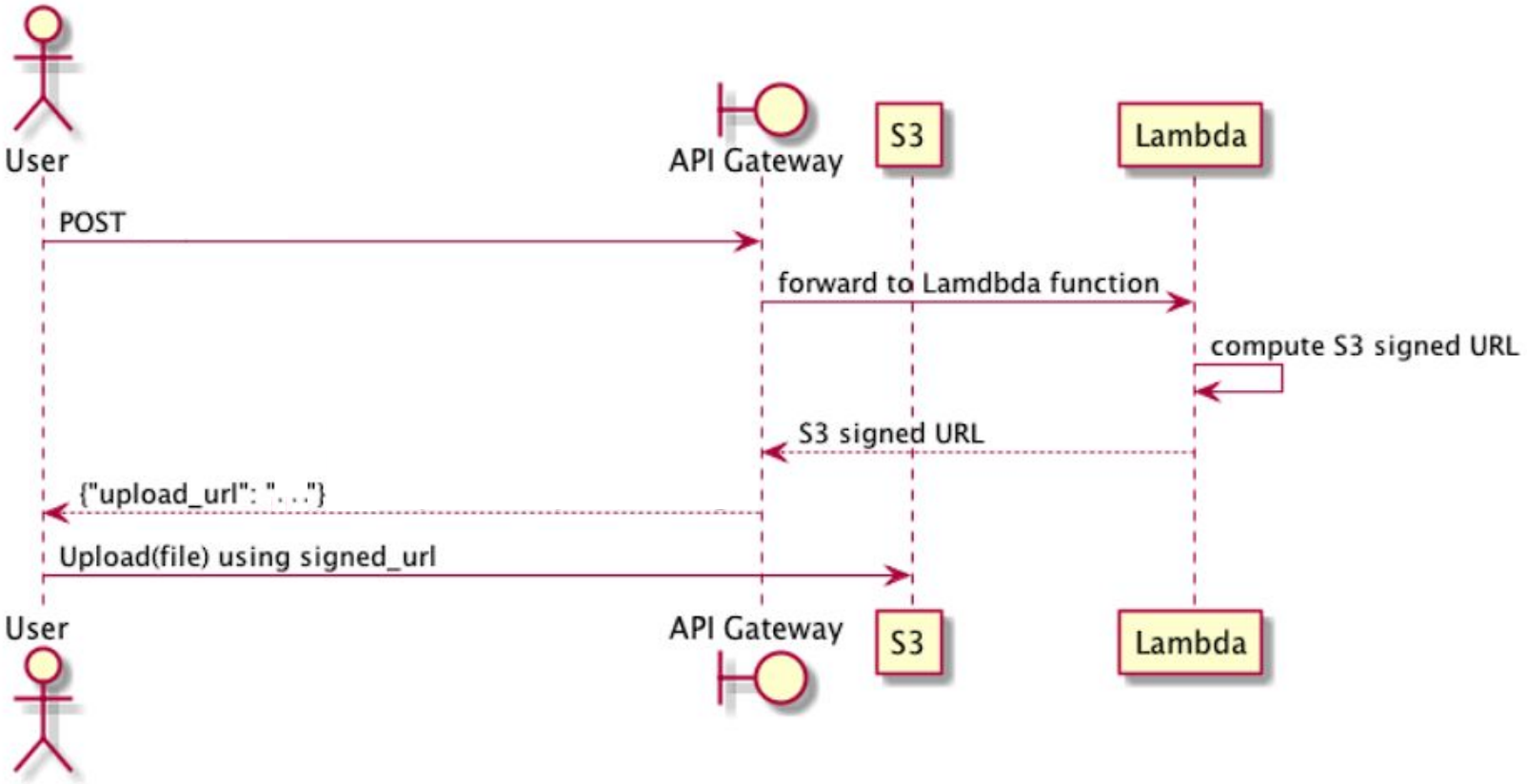
1. Download/Upload di file da un bucket *S3*
2. Aggiungere messaggi in una coda *SQS*



Caratteristiche principali:

- Throttling burst: *500 reqs*
- Throttling rate limit: *50 req/s*
- WAF: quattro Web ACLs per bloccare/limitare certe richieste

API Gateway - signed URL



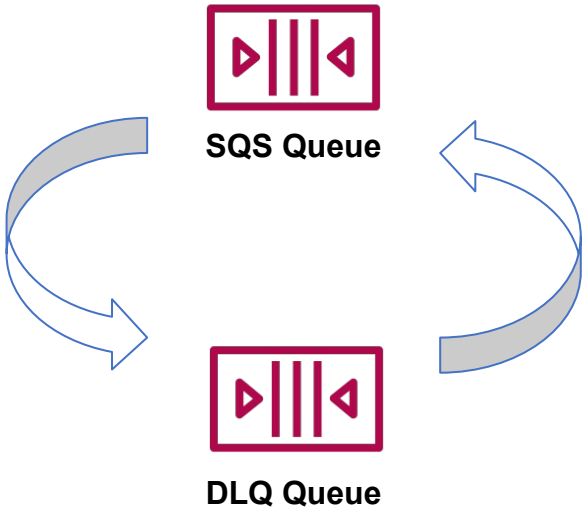
S3 I/O Buckets e Code SQS

awss-input-files

awss-result-files

- Transfer Acceleration
- Oggetti non pubblici
- Inputs lifecycle: “*expiration*”
- Results lifecycle: “*north-pole*”

- Code cifrate con SSE-SQS
- FIFO Queue: *InputMsg*
- Standard Queue: *sendMail*
- Code DLQ per l’affidabilità (numero max di tentativi: 2)



AWS Lambdas

runECSTask

- Python 3.9
- Triggerato dalla coda di input
- Istanzia un nuovo container
- Orchestrator

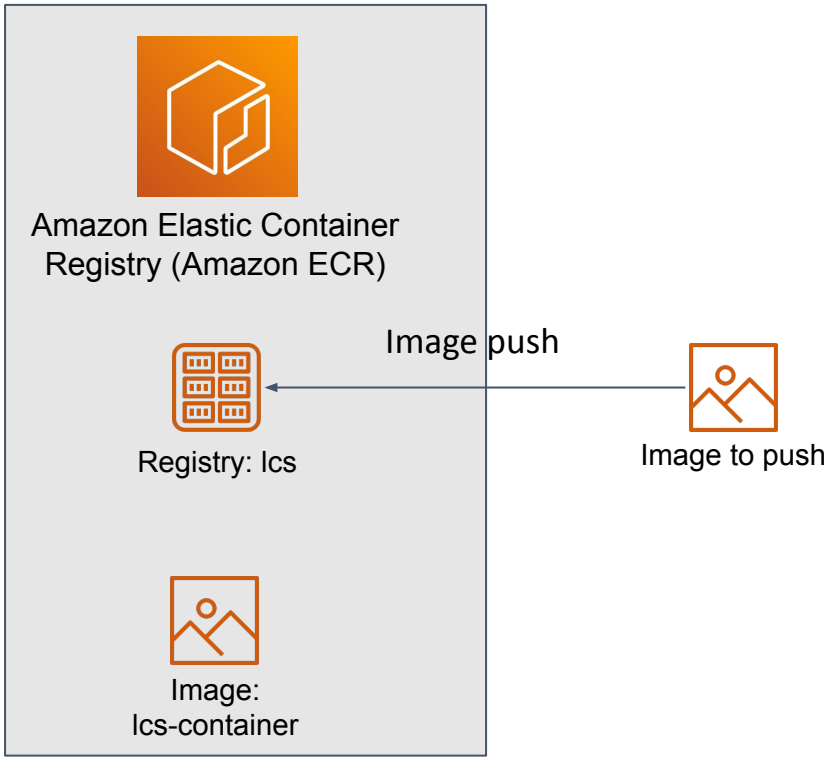
sendMail

- Python 3.9
- Triggerato dalla coda dei risultati
- Invia mail all'utente con risultato

LogsToOS

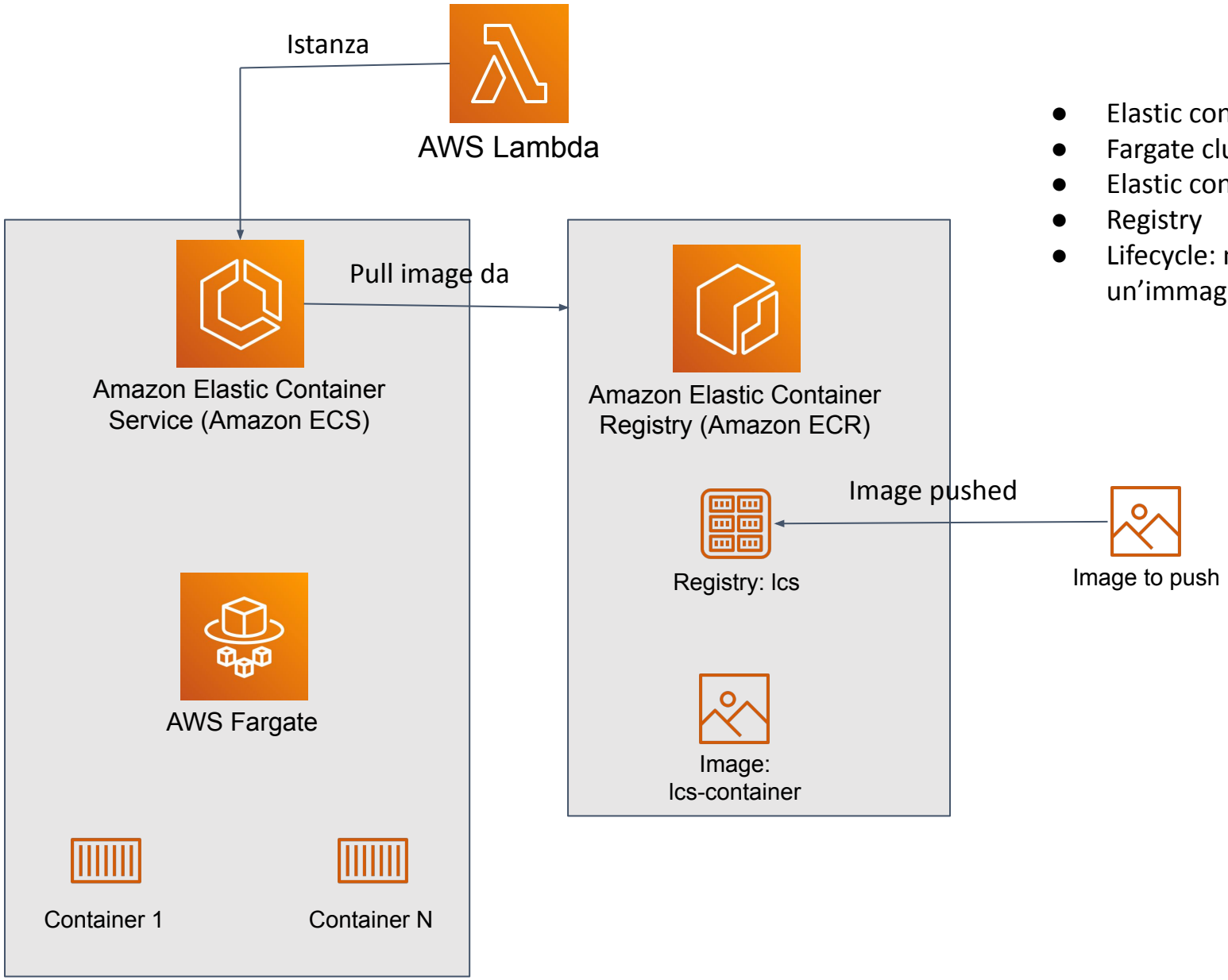
- NodeJS 14.x
- Legge i log di cloudwatch e li carica su opensearch

Docker



- Ubuntu 22.04
- Python + librerie
- Avvio: esegue run.py
- Minore boot time rispetto EC2

ECS / ECR

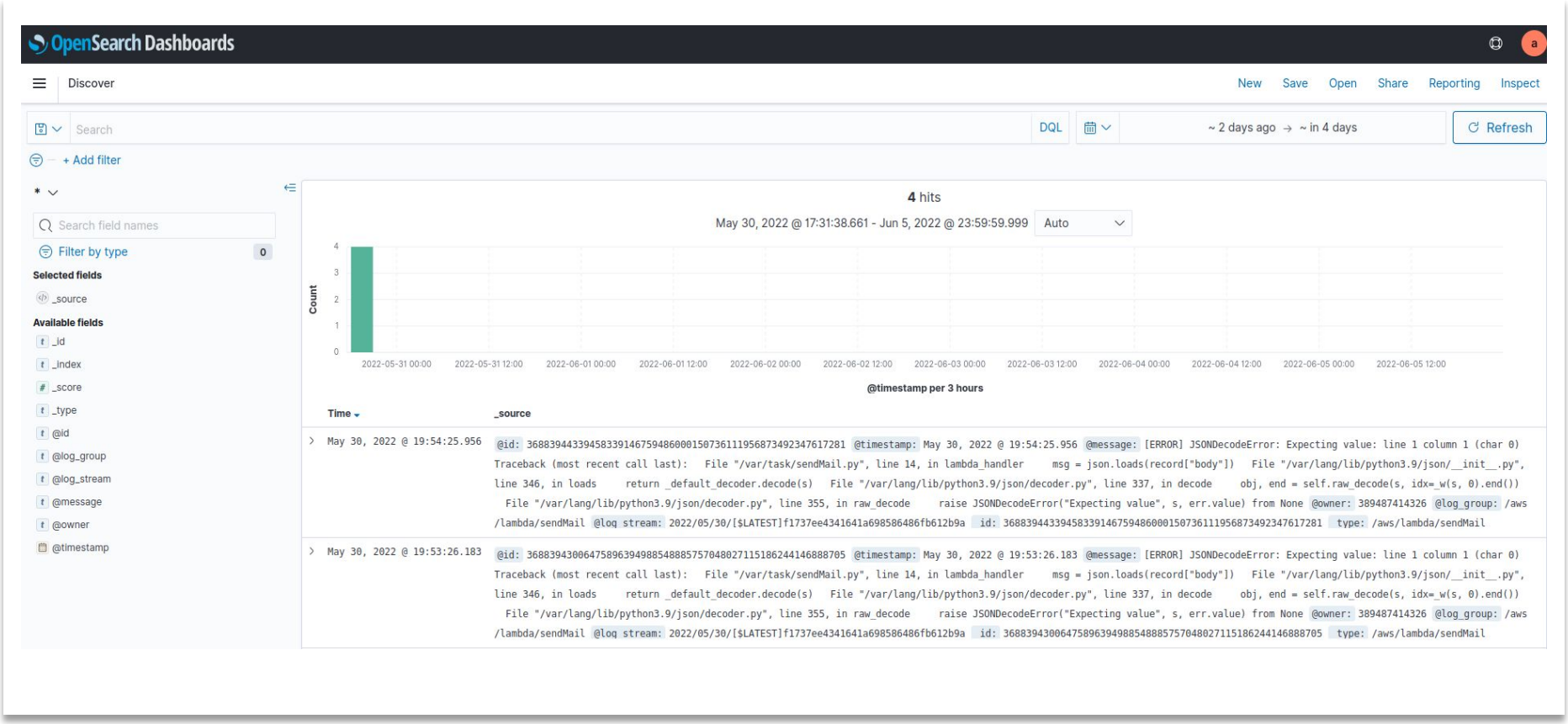


- Elastic container service
- Fargate cluster
- Elastic container registry
- Registry
- Lifecycle: mantiene fino a un'immagine non taggata

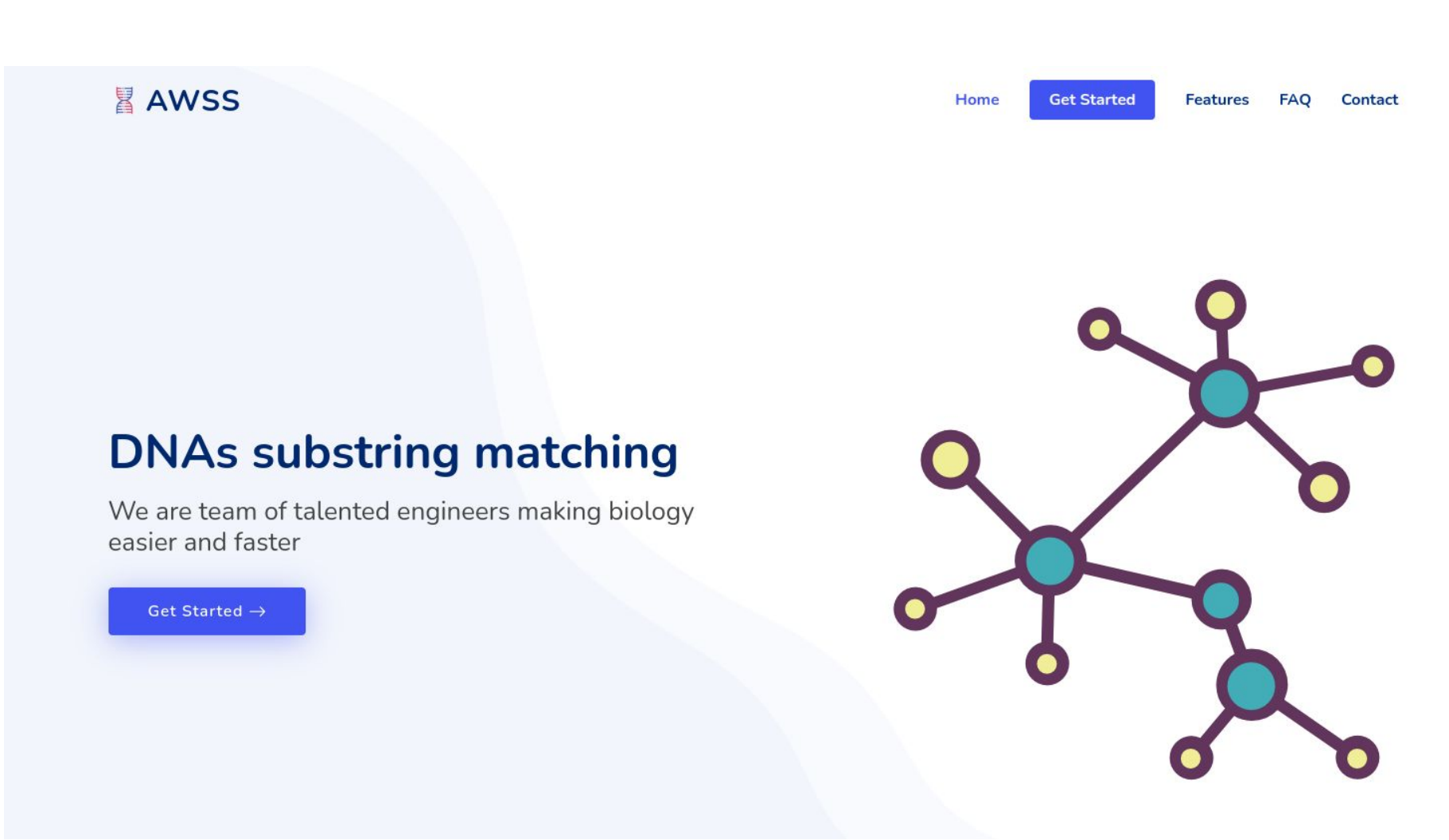
Cloudwatch + OpenSearch

- Aggregazione soltanto dei log di errori mediante filter pattern “ERROR”
- 2 zone di disponibilità
- 2 nodi di tipo t3.small.search con 10 GB SSD ognuno
- Auto-tune con finestra di manutezione di 3 ore ogni domenica dalle 9
- OpenSearch 1.2
- Fine grained access control con dashboard pubblica

OpenSearch Dashboard

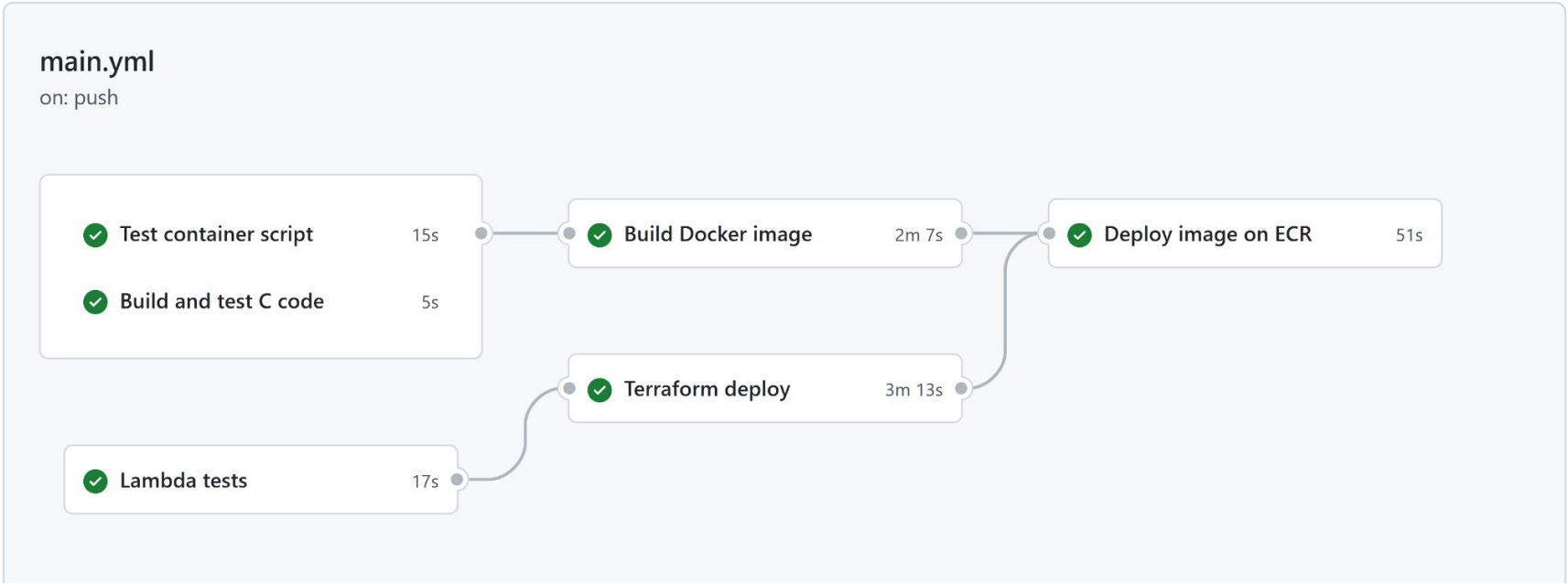


Interfaccia Web



CI/CD

GitHub Actions Workflow



- run.py (script container)
- SendMail test
- Test codice C (LCS)

Sviluppi futuri

- Sistema di autenticazione, area riservata elaborazioni e risultati
- Dashboard risorse utilizzate e billing
- “File-multipart”: download/upload dei file di DNA, problemi di rete e parallelizzazione
- Problemi di allocazione memoria e singolo workflow distribuito

Grazie per l’attenzione

Repository: https://github.com/domenico-rgs/AWSS

Working demo: https://awss-cloud.ga/