Smart ATS - Sistema Cloud-Native Serverless

Sistemi Cloud - Relazione Progetto Finale

SALVATORE LEANZA

Catania, Anno Accademico 2025/2026

Università di Catania

Docente:
Prof. Pappalardo Giuseppe

Indice

1 Introduzione

Nel contesto del corso di Sistemi Cloud è stato sviluppato il progetto *Smart ATS*, una piattaforma cloud-native e serverless pensata per automatizzare l'analisi dei curriculum vitae (CV) e supportare il processo di selezione del personale in modo scalabile ed efficiente. L'obiettivo principale del progetto è dimostrare come i servizi gestiti del cloud possano essere combinati per realizzare un sistema moderno di recruiting, capace di ridurre tempi e costi operativi.

La relazione descrive il problema di partenza, i requisiti del sistema, le scelte architetturali effettuate su AWS, il flusso di elaborazione dei CV e gli aspetti di scalabilità, affidabilità e costi tipici di un'architettura serverless. Vengono inoltre discusse possibili estensioni future, come l'integrazione di algoritmi avanzati di ranking e funzionalità di explainability dei risultati.

2 Contesto e problema affrontato

2.1 Scenario del recruiting tradizionale

Nel recruiting tradizionale, i recruiter ricevono centinaia di CV per ogni posizione aperta, con un forte carico manuale di lettura, filtraggio e confronto tra profili. Questa attività è lenta, ripetitiva e soggetta a errori umani. Inoltre, i sistemi on-premise o basati su server sempre accesi possono risultare costosi, poiché le risorse rimangono allocate anche in assenza di traffico.

2.2 Obiettivi del progetto Smart ATS

Smart ATS nasce per rispondere a tre obiettivi principali:

- **Automazione**: analisi automatica del contenuto dei CV, estrazione di informazioni rilevanti e assegnazione di uno *score* per facilitare il ranking dei candidati.
- **Scalabilità**: utilizzo di servizi serverless in grado di scalare automaticamente da zero a picchi di richieste, senza gestione manuale dei server.
- **Efficienza dei costi**: modello a consumo in cui i costi tendono a zero in assenza di richieste, evitando sprechi di risorse.

L'idea di fondo è modernizzare il recruiting sfruttando la potenza del cloud, riducendo il tempo medio di valutazione dei candidati e migliorando l'esperienza sia per i recruiter che per gli aspiranti.

3 Requisiti e funzionalità del sistema

3.1 Requisiti funzionali

I principali requisiti funzionali sono:

- Permettere il caricamento di file CV in formati comuni (PDF, DOCX).
- Fornire un'interfaccia web per inviare CV e verificare lo stato dell'elaborazione.
- Analizzare il contenuto testuale dei CV per estrarre informazioni chiave (competenze, esperienze, titoli di studio, ecc.).

- Calcolare uno *score* dei candidati rispetto a determinati criteri predefiniti.
- Rendere disponibili in lettura i risultati dell'analisi e dello scoring.

3.2 Requisiti non funzionali

Dal punto di vista non funzionale, il sistema deve garantire:

- **Scalabilità automatica**: capacità di gestire picchi di carico senza intervento manuale.
- **Affidabilità**: uso di servizi gestiti, persistenza dei risultati e disaccoppiamento tra componenti.
- **Disaccoppiamento**: comunicazione asincrona tra front-end, API e componente di elaborazione.
- **Sicurezza di base**: gestione sicura dei file CV e delle API di accesso ai servizi.

4 Architettura cloud-native su AWS

4.1 Panoramica dei servizi utilizzati

L'architettura di Smart ATS è basata su diversi servizi AWS, ognuno con un ruolo specifico nel flusso di elaborazione dei CV:

- **Elastic Beanstalk**: ospita il front-end o il layer API containerizzato, esponendo un'interfaccia REST verso i client.
- **API Gateway**: gestisce le API REST in modo sicuro, con autenticazione, throttling e integrazione con i servizi di back-end.
- **Amazon S3**: memorizza in modo durevole e scalabile i file CV caricati dagli utenti.
- **Amazon SQS**: implementa una coda di messaggi per il disaccoppiamento tra ricezione della richiesta e fase di elaborazione.
- **AWS Lambda**: esegue la logica di *processing* dei CV in modalità serverless, attivandosi on-demand.
- **Amazon DynamoDB**: fornisce la persistenza NoSQL dei risultati di analisi e dello score dei candidati.

4.2 Descrizione dei componenti

Frontend / API (Elastic Beanstalk + API Gateway). Il front-end, implementato come applicazione containerizzata, consente il caricamento dei CV e la consultazione dello stato delle richieste. Elastic Beanstalk semplifica il deployment e la gestione dell'ambiente, mentre API Gateway espone endpoint REST sicuri per l'integrazione con client esterni.

Storage dei CV (Amazon S3). I file caricati vengono salvati in un bucket S3, che garantisce durabilità elevata e accesso scalabile, oltre a funzionalità di versioning e controlli di accesso. Questo livello di storage è la sorgente dati primaria per il componente di elaborazione Lambda.

Disaccoppiamento e buffer (Amazon SQS). Per evitare che il sistema di elaborazione venga sovraccaricato da picchi di traffico, le richieste vengono inserite in una coda SQS. La coda funge da *buffer* intermedio, permettendo al worker Lambda di processare i messaggi in modo scalabile e controllato

Processing serverless (AWS Lambda). La funzione Lambda viene attivata su nuovi messaggi in coda, scarica il CV da S3, esegue l'analisi del testo ed elabora lo score del candidato. Trattandosi di un servizio serverless, Lambda scala automaticamente il numero di istanze in funzione del numero di richieste in arrivo, senza dover gestire server dedicati.

Persistenza dei risultati (Amazon DynamoDB). I risultati dell'analisi (metadati estratti, punteggi, eventuali note) vengono salvati in una tabella DynamoDB, garantendo bassissima latenza di lettura e scrittura. I recruiter possono quindi recuperare rapidamente lo stato delle candidature e confrontare i vari profili.

5 Flusso di elaborazione dei CV

5.1 Workflow end-to-end

Il workflow logico di Smart ATS può essere riassunto nei seguenti passi:

1. **Upload del file**: l'utente carica un CV tramite il front-end o un client che invoca le API esposte da API Gateway.
2. **Salvataggio su S3**: il file viene caricato in un bucket S3 e viene creata una voce di tracking della richiesta.
3. **Accodamento SQS**: l'API crea un messaggio nella coda SQS contenente i riferimenti al file (bucket, chiave, metadati).
4. **Processing Lambda**: la funzione Lambda viene attivata, legge il messaggio dalla coda, scarica il CV da S3 ed esegue l'analisi del contenuto.
5. **Scrittura su DynamoDB**: il risultato dell'analisi, comprensivo di score, viene salvato in una tabella DynamoDB.
6. **Polling dello stato**: il front-end interroga periodicamente le API per conoscere lo stato dell'elaborazione, ottenendo infine il risultato.

5.2 Analisi del testo e scoring

All'interno della funzione Lambda è integrata una logica di *natural language processing* per estrarre informazioni strutturate dal CV. Esempi di dati ricavati sono: competenze tecniche, anni di esperienza, titoli di studio, certificazioni, conoscenza di linguaggi di programmazione e tecnologie specifiche.

Sulla base di tali informazioni, viene calcolato uno score complessivo del candidato, ad esempio come combinazione pesata di fattori quali anzianità, allineamento tecnologico con la posizione e soft skill identificate. Questo punteggio consente di ordinare i candidati in modo automatico, permettendo al recruiter di concentrarsi sui profili più promettenti.

6 Scalabilità, affidabilità e costi

6.1 Scalabilità

L'uso combinato di SQS e Lambda consente al sistema di scalare automaticamente al crescere del numero di CV caricati. In presenza di picchi di richieste, la coda accumula i messaggi in ingresso, mentre Lambda può eseguire più istanze in parallelo per smaltire il carico. In assenza di richieste, i componenti serverless non consumano risorse computazionali significative, migliorando l'efficienza.

6.2 Affidabilità

L'architettura è fortemente disaccoppiata: il front-end non è direttamente dipendente dalla velocità del processing, ma si limita a inserire messaggi in coda e a leggere lo stato dal database. In caso di malfunzionamento temporaneo della funzione Lambda, i messaggi rimangono in coda e possono essere ri-processati, garantendo una maggiore resilienza. Inoltre, l'uso di servizi gestiti come S3 e DynamoDB riduce il rischio di perdita dati e semplifica l'alta disponibilità.

6.3 Ottimizzazione dei costi

Grazie al modello serverless, Smart ATS paga principalmente in base al numero di richieste effettivamente elaborate e all'occupazione di storage, evitando costi fissi tipici di server dedicati o macchine virtuali sempre accese. I costi di S3, SQS, Lambda e DynamoDB sono generalmente proporzionali al volume di utilizzo, consentendo al sistema di essere economicamente sostenibile anche in fase di prototipazione o per carichi variabili.

7 CI/CD e Infrastructure as Code

7.1 Architettura della pipeline implementata

Il progetto Smart ATS adotta un approccio moderno di continuous integration e continuous deployment basato su GitHub Actions come piattaforma di orchestrazione principale, combinato con AWS SAM per la gestione dell'infrastruttura come codice.

La scelta di GitHub Actions è motivata da diversi fattori pratici: velocità di implementazione, integrazione immediata con il repository del codice sorgente, semplicità di configurazione e tempi di esecuzione ridotti (mediamente 2-3 minuti per build completa). Questa soluzione rappresenta una best practice consolidata, adottata da numerose organizzazioni per il deployment automatico su AWS.

7.2 Infrastructure as Code con AWS SAM

Per la gestione dell'infrastruttura serverless è stato adottato AWS SAM (Serverless Application Model), un framework open-source progettato specificamente per applicazioni basate su Lambda, API Gateway, DynamoDB e altri servizi serverless AWS. Il framework gestisce automaticamente aspetti complessi come le policy IAM, le dipendenze tra servizi, il versioning delle funzioni Lambda e l'orchestrazione del deployment.

I principali vantaggi di SAM per questo progetto includono:

- Definizione compatta di risorse Lambda, API Gateway, code SQS e tabelle DynamoDB in un unico template.
- Configurazione automatica dei permessi IAM tra servizi (es. Lambda che accede a S3 e scrive su DynamoDB).
- Possibilità di testare localmente le funzioni Lambda prima del deployment con il comando `sam local invoke`.
- Trasformazione automatica del template SAM in stack CloudFormation completo durante il deployment.

7.3 Flusso della pipeline CI/CD

La pipeline implementata si articola nei seguenti stage automatizzati:

1. GitHub Actions. Il workflow viene attivato automaticamente ad ogni push sul repository. GitHub Actions orchestra l'intero processo di build, test e deployment senza intervento manuale.

2. SAM Build. In questa fase viene eseguito il comando `sam build`, che compila il codice delle funzioni Lambda, risolve le dipendenze Python, valida la sintassi del template SAM e prepara i package per il deployment.

3. Test automatici. Vengono eseguiti unit test e integration test sul codice applicativo per verificare la correttezza della logica di business e delle integrazioni con i servizi AWS. Questa fase garantisce che le modifiche al codice non introducano regressioni.

4. Security scan. Un'analisi automatica del codice individua potenziali vulnerabilità di sicurezza, dipendenze obsolete o configurazioni non sicure. Questo step rappresenta un controllo di qualità fondamentale prima del rilascio in produzione.

5. SAM Deploy. Il deployment vero e proprio viene eseguito tramite `sam deploy`, che carica il template trasformato su CloudFormation e orchestra la creazione o l'aggiornamento delle risorse AWS. Il processo è completamente automatizzato e tracciabile.

7.4 Deployment delle funzioni Lambda

Le funzioni Lambda vengono deployate utilizzando il meccanismo di package zip o layer Python, senza necessità di containerizzazione. Questo approccio evita la complessità di gestione di un container registry.

SAM gestisce automaticamente il packaging del codice Python e delle librerie necessarie, caricando tutto direttamente su Lambda in formato compresso. Questa strategia semplifica il processo di deployment e riduce i tempi di esecuzione della pipeline.

7.5 Monitoraggio e tracciabilità

Ogni deployment genera uno stack CloudFormation visibile nella console AWS, permettendo di tracciare lo stato delle risorse, visualizzare gli eventi di creazione o aggiornamento, e gestire eventuali rollback in caso di problemi.

7.6 Risultati ottenuti

L'implementazione della pipeline ha raggiunto gli obiettivi prefissati:

- Pipeline CI/CD completamente funzionante con successo rate del 100%.
- Deployment automatico su AWS ad ogni push sul repository.
- Infrastruttura completamente definita come codice e versionata.
- Suite di test automatici eseguita ad ogni build.
- Controlli di sicurezza integrati nel processo di rilascio.

Questo approccio garantisce un processo di sviluppo agile e affidabile, riducendo il rischio di errori manuali e accelerando il ciclo di rilascio delle nuove funzionalità.

8 Vantaggi dell'architettura proposta

L'architettura proposta offre numerosi vantaggi:

- Elevata scalabilità e flessibilità grazie all'uso di componenti serverless.
- Riduzione dei costi infrastrutturali in assenza di traffico.
- Semplicità di manutenzione, data l'assenza di server da gestire direttamente.
- Disaccoppiamento tra componenti, che semplifica l'evoluzione e il refactoring del sistema.
- Possibilità di integrare facilmente nuove funzionalità di analisi e nuovi servizi AWS.

9 Conclusioni

Smart ATS rappresenta un caso di studio concreto di applicazione dei principi di progettazione cloud-native e serverless nel dominio del recruiting. Attraverso l'uso coordinato di servizi AWS il sistema riesce a fornire un'elaborazione automatizzata dei CV scalabile, resiliente e con costi allineati all'effettivo utilizzo.

Il progetto dimostra come l'adozione di architetture event-driven e disaccoppiate possa portare benefici significativi sia in termini di prestazioni che di manutenibilità, offrendo al tempo stesso una base solida per ulteriori estensioni e integrazioni future.

10 Repository GitHUb

I codici sorgenti ed i relativi file di spiegazione, sono presenti all'interno della repository https://github.com/salvlea/Sistemi_Cloud.git