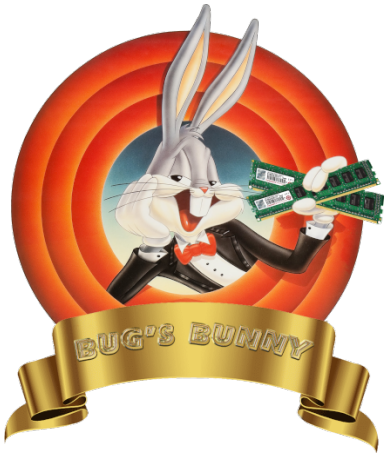
Università degli Studi di Padova
Ingegneria del Software
Anno Accademico: 2021/2022



Bug's Bunny

Verbale esterno

2022/04/21

Redazione: Giulio Zanatta
Verifica: Ruth Genevieve Bousapnamene
Approvazione: Angela Arena

Generalità

- **Ora inizio:** 14.00
- **Ora fine:** 15.45

Presenze

- Angela Arena
- Marco Bellò
- Tommaso Di Fant
- Ruth Genevieve Bousapnamene
- Matteo Tossuto
- Marco Volpato
- Giulio Zanatta

Resoconto

Incontro con il proponente per l'introduzione al concetto di serverless, Lambda, API Gateway, DB e servizi altri AWS. L'incontro si è composto di due sezioni distinte: una teorica e una pratica.

Teoria

RDS: componente serverless (MySQL) a gestione totalmente esterna, nessun lavoro di manutenzione e "montaggio" da parte nostra.

Serverless: modello di programmazione a microservizi, cioè ogni Lambda fa una cosa per conto suo (infrastruttura gestita interamente da AWS).

AWS fa uno scaling automatico: alloca lui le risorse hardware necessarie in base alle richieste.

Tools:

- *EC2*: server gestito da Amazon, ma non propriamente elemento serverless, ci si può accedere via SSH.
- *S3*: file system, è un DB che si mostra all'utente come se fosse una sorta di g-drive
- *CloudFront*: CDN (content delivery network), solitamente posto davanti API Gateway, permette di servire contenuti statici, fare caching, proteggere da attacchi DDOS.

- *VPC*: sistema di gestione del networking fra i vari servizi, dove è possibile creare reti virtuali private o pubbliche. Necessita di un internet gateway per poter comunicare con l'esterno (esterno di AWS quindi Internet).
- *API Gateway*: punto dove parte frontend invia richieste alle Lambda functions.
- *Cognito*: Database per credenziali utente.
- *Serverless framework*: astrae l'infrastruttura attraverso template in formato YAML, si occupa per esempio di gestire in automatico la creazione di funzioni Lambda e l'upload su AWS di eventuali dipendenze del codice che verrà eseguito (durante la demo non ha funzionato proprio bene...).

Lambda

Prendono linguaggi come JavaScript, Typescript, Python, Go. È motore di esecuzione, funziona ad eventi, richieste dirette, oppure tramite schedule.

API Gateway -¿ chiama Lambda -¿ che svolge le sue istruzioni (ad esempio chiamare altre Lambda, servizi AWS o servizi esterni). Osservazione: Lambda limitate da massimo 3GB e da massimo 15 minuti di tempo in esecuzione (per aggirare queste limitazioni si deve, in caso, dividere il carico su più Lambda). Possono funzionare in modo sincrono (a chiamata) o asincrono (schedulato).

DataBase

RDS: Database relazionale (MySQL, PostgreSQL e simili) In alternativa si può usare Amazon Aurora Serverless (basato su MySQL) al posto di RDS che è avvantaggiato dal fatto di essere scalabile, più veloce, più sicuro e più affidabile (grazie alla sua caratteristica di essere serverless).

Altro vantaggio di Aurora è il Multi AZ (Availability Zone): DB si trova su diverse zone indipendenti, una che fa da Master, per aumentare l'affidabilità e la latenza del DB (le zone non master possono, all'occorrenza, essere promosse a master).

API Gateway

Pensato per uso sincrono, gestisce richiesta che permea per un massimo di 30 secondi (aggirabile con Lambda che chiamano altre Lambda, cioè con catena di richieste), comunica direttamente con Amazon Cloudfront che fa da tramite con l'internet esterno alla mia web app (immaginabile come una collezione di servizi AWS gestiti dal Gateway).

Pratica

VPC (Virtual Private Cloud)

: È il contenitore nel quale inserisco i miei servizi aws (es: Lambda va deployata in una vpc, dove andrò a mettere poi dei gateway). La posso vedere come una "rete locale" in cloud (è una porzione di rete chiusa dall'esterno).
Configurazione:
IPv4 CIDR Standard: 10.0.0.0/16

Poi vado a fare due subnet (partizioni ulteriori della vpc), in genere due pubbliche e due private:
private standard:
10.0.0.0/24
10.0.1.0/24
pubbliche standard:
10.0.10.0/24
10.0.11.0/24
Poi creo un gateway e lo attacco ("attach") alla vpc. Poi creo un NAT gateway che dialoga con il gateway e serve a dialogare con l'esterno (importante: allocare elastic ip). Infine associare subnet.

Ora posso creare RDS (o, nel nostro caso, Aurora), per prima cosa devo creare dei subnet groups. Succesivamente attacco vpc e security group (fatti prima) e poi si abilita "data API".

Facoltativo (ma consigliato): tramite il secret manager creare un secret che contenga credenziali di accesso al DB, così che poi basti passarlo alle Lambda per far loro accedere in sicurezza al database.

Il referente di zero12 ha mandato sul canale Slack il codice usato durante la dimostrazione, quindi verrà qui omesso. Il seguente comando, eseguito da terminale, esegue il deploy su AWS dell'infrastruttura.

```
AWS_SDK_LOAD_CONFIG=1 serverless deploy
```

Osservazione: il primo deploy è spesso traumatico, ed è consigliato avere una CI/CD ben fatta (con pipeline di deploy) per evitare dei problemi tipici di un deploy manuale (che infatti è fallito durante la dimostrazione di zero12, obbligandoli da chiudere anticipatamente la lezione).