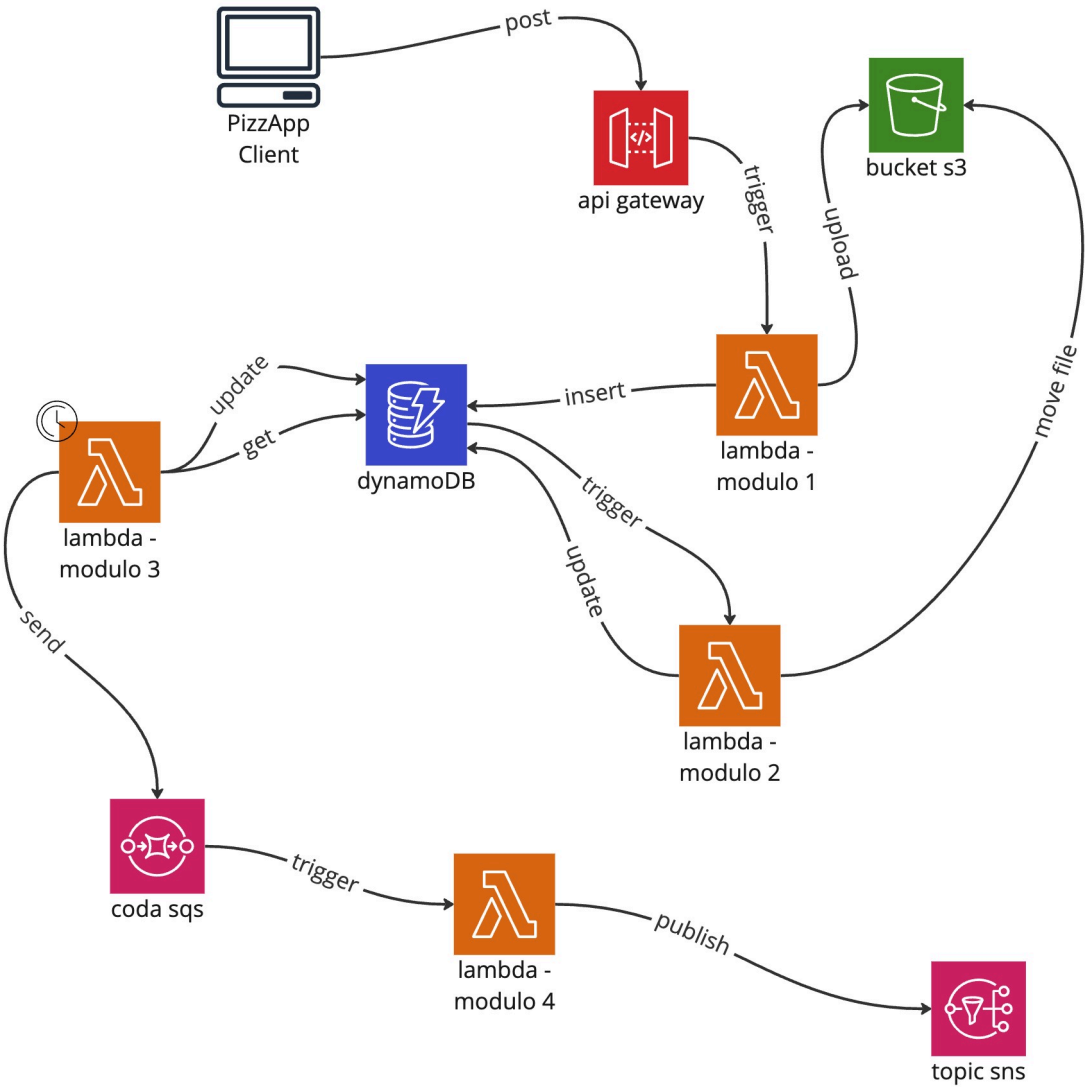
Project - Variante Delta

PizzApp

Pizza Printing è un'azienda innovativa che prepare pizze e stampa anche cartoni con grafica personalizzata!

Ci ha commissionato lo sviluppo del software, chiamato PizzApp, che prende in carico gli ordini, associa le pizze, produce e consegna al cliente.

Di seguito il disegno dell'architettura:



L'applicazione deve essere così strutturata:

Modulo 1 - ricezione ordine cliente

Attraverso un client http (es. Postman) un cliente deve essere in grado di fare una chiamata POST per inserire un nuovo ordine. Il payload della chiamata è il seguente:

```
{
  "file": "https://.....",
  "pizza": [margherita|capricciosa|4 stagioni],
  "qty": [1-5]
}
```

La lambda deve:

- validare l'ordine con le seguenti regole:
  - "`file`" non deve essere vuoto
  - "`pizza`" deve essere un valore tra quelli consentiti (`margherita`, `capricciosa` o `4 stagioni`)
  - "`qty`" deve essere compreso tra 1 e 5
- caricare il file su un bucket s3 dentro la cartella "`upload/`"
- inserire un record su dynamo aggiungendo, al payload dell'utente, i seguenti campi:
  - "`id`" con valore a piacere (ovviamente univoco)
  - "`status`" con valore `NEW`

Se la validazione fallisce, la lambda restituisce un errore all'utente.

Modulo 2 - preparazione ordine

La scrittura di un record con stato "NEW" triggera una seconda lambda che sposta il file dal path "`/upload/`" al path "`/ready/`" rinominandolo come "`<id>_<pizza>_<qty>`" e aggiorna lo stato su dynamo come "`READY`".

Modulo 3 - produzione ordini

Una lambda schedulata ogni 5 minuti recupera tutti i record con stato "`READY`" e li raggruppa per tipologia di pizza. Per ogni gruppo di almeno 3 elementi viene inviato un messaggio ad una coda SQS strutturato come segue:

```
{
  "items": [
    {
      "id": "..",
      "file": "...",
      "pizza": "...",
      "qty": ...
    },
    {
      "id": "..",
      "file": "...",
      "pizza": "...",
      "qty": ...
    },
    ...
  ]
}
```

I soli record inviati alla coda vengono aggiornati su dynamo con stato "`DONE`".

Modulo 4 - notifica ordine pronto

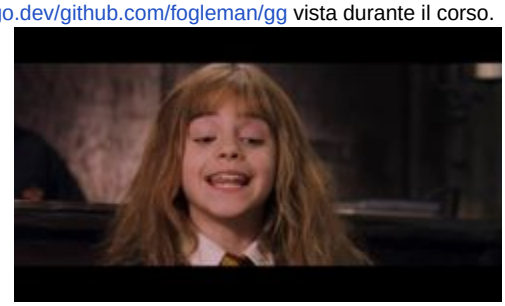
La coda SQS triggera una lambda che invia un messaggio su un topic SNS strutturato come segue:

```
{
  "team": <nome_del_team>,
  "items": [
    {
      "id": "..",
      "file": "...",
      "pizza": "...",
      "qty": ...
    },
    {
      "id": "..",
      "file": "...",
      "pizza": "...",
      "qty": ...
    },
    ...
  ]
}
```

Il topic ha una sottoscrizione con la mail lesson@pixartprinting.com.

Bonus:

1. I cartoni della pizza possono avere lato da 33 cm (pizze classiche) o 46 cm (pizze famiglia). A causa di un problema su una delle macchine, possiamo produrre solo i cartoni da 33, per cui gli ordini da 46 cm vanno scartati.
   Aggiungere alla struttura dell'ordine un campo "`size`" con valore 33 o 46.
   La prima lambda aggiunge, tra i criteri da validare, il campo "`size`" con i valori ammessi.
   La seconda lambda, prima di rinominare il file, verifica se la "`size`" è diversa da 33 e in tal caso setta lo stato come "`DISCARDED`".
2. Le nuove macchine di taglio hanno necessità di avere un layer con un cerchio rosso semi-trasparente esattamente al centro dell' immagine. (vedi esempio)
   Applicare il layer sul file salvato nel path "`/ready/`".
   Si consiglia l'uso della libreria https://pkg.go.dev/github.com/fogleman/gg vista durante il corso.



3. Anziché chiedere “`size`" all’utente, vogliamo basarci sulla grafica caricata, per cui modificare la prima lambda assegnando size = 33 se base e altezza del file sono inferiori o uguali a 33 cm, size = 46 se base e altezza sono maggiori di 33 cm e inferiori o uguali a 46 cm. In tutti gli altri casi salvare l'ordine con stato "`DISCARDED`".
   Considerare una risoluzione di 300 dpi.
4. Controllare gli ordini dalla console di AWS risulta scomodo, prevedere un api che fa il listing di tutti i record presenti su dynamo, con eventuale possibilità di filtrare per stato. E’ richiesta solo l’api, non il frontend.

Istruzioni

1. I progetti saranno sviluppati da team di 2/3 persone.
2. Le tecnologie da utilizzare sono:
   a. serverless per il deploy delle risorse
   b. go come linguaggio di programmazione
3. Il nome del team deve corrispondere al nome del progetto o comunque comparire come prefisso/suffisso nel nome delle risorse, in modo da evitare interferenze con il lavoro degli altri.
4. Per ciascuna lambda si avranno 90 minuti a disposizione (ovviamente è possibile chiedere aiuto ai docenti).
5. I punti bonus possono essere sviluppati in qualsiasi ordine solo dopo aver concluso il progetto base.