COMS E6998: **Microservices and Cloud Applications**

Lecture 11: Q&A, Final Project, All Project Summary

Dr. Donald F. Ferguson
dff9@columbia.edu

© Donald F. Ferguson, 2017. All rights reserved.

Comments
Questions

© Donald F. Ferguson, 2017. All rights reserved.

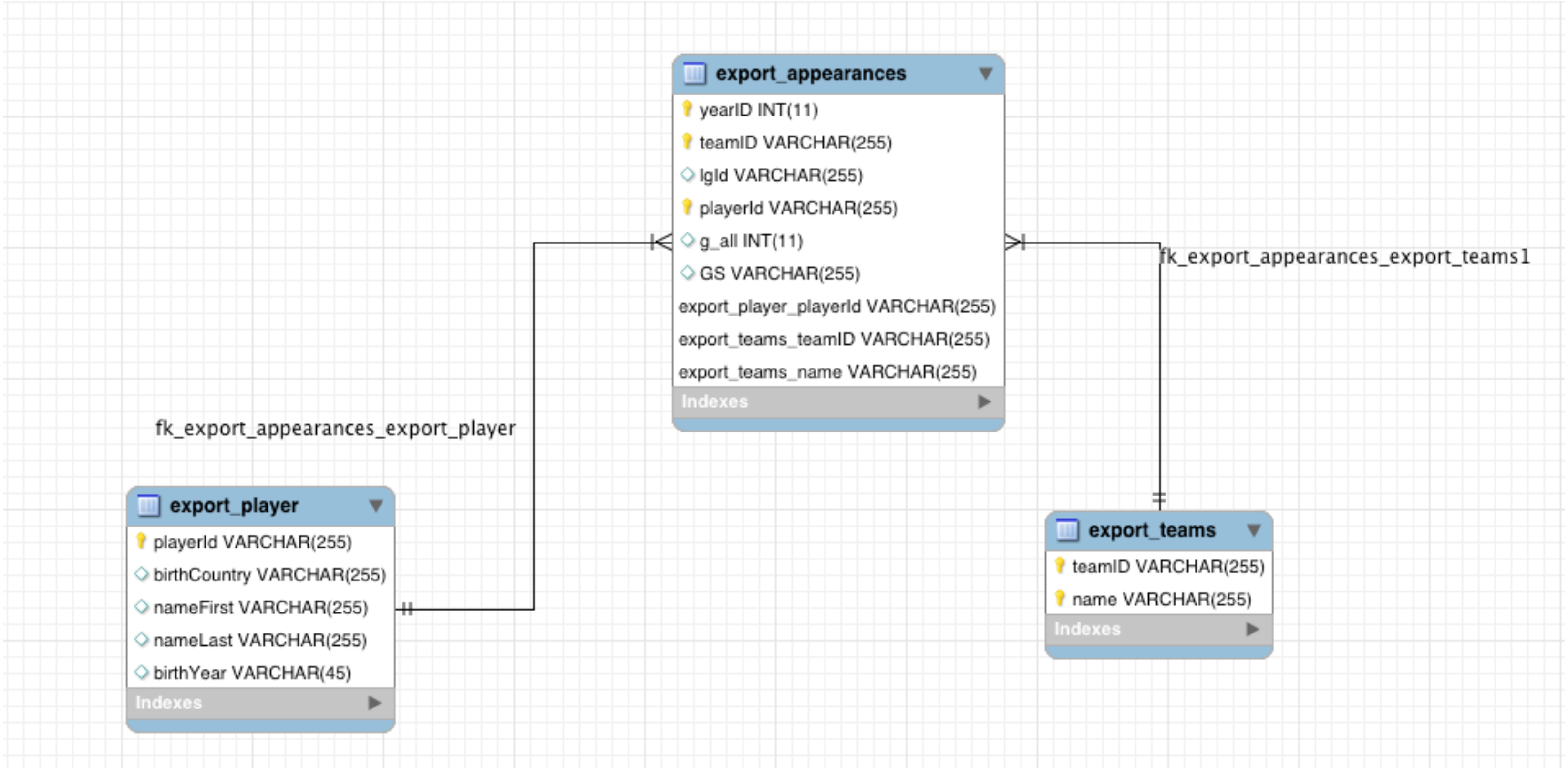
Projects

© Donald F. Ferguson, 2017. All rights reserved.

Final Project

© Donald F. Ferguson, 2017. All rights reserved.

Logical Data Model



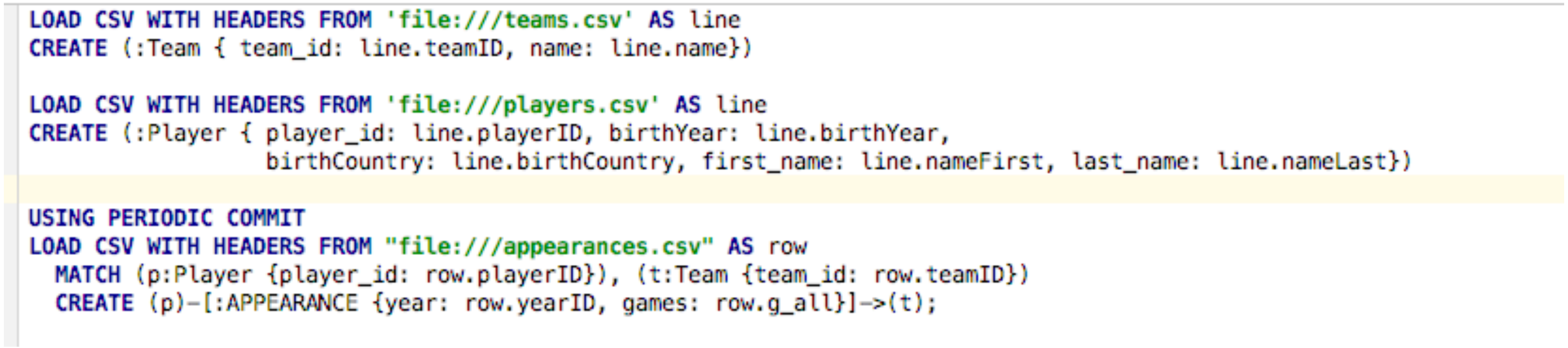
© Donald F. Ferguson, 2017. All rights reserved.

Standalone Neo4J Microservice

- Import data into Neo4J (local sever or in cloud)
  - Files are at https://courseworks2.columbia.edu/courses/47421/files/#
    - appearances.csv
    - teams.csv
    - players.csv
- Neo4J Import Process
  - Import from URL via HTTP or
  - From local directory (On Mac = /Users/donaldferguson/Documents/Neo4j/default.graphdb/import)
  - Using LOAD CSV commands

© Donald F. Ferguson, 2017. All rights reserved.

Neo4J LOAD CSV

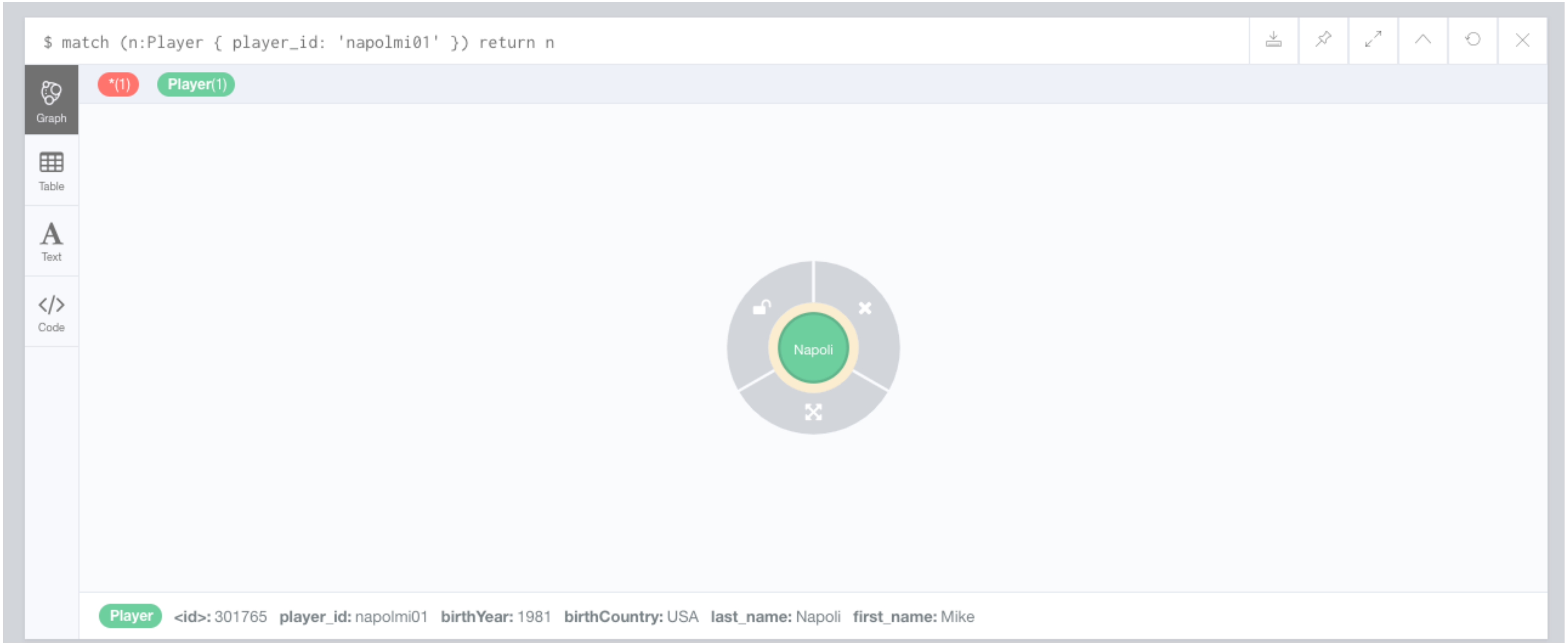


```
LOAD CSV WITH HEADERS FROM 'file:///teams.csv' AS line
CREATE (:Team { team_id: line.teamID, name: line.name})

LOAD CSV WITH HEADERS FROM 'file:///players.csv' AS line
CREATE (:Player { player_id: line.playerID, birthYear: line.birthYear,
                  birthCountry: line.birthCountry, first_name: line.nameFirst, last_name: line.nameLast})

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///appearances.csv" AS row
  MATCH (p:Player {player_id: row.playerID}), (t:Team {team_id: row.teamID})
  CREATE (p)-[:APPEARANCE {year: row.yearID, games: row.g_all}]->(t);
```

- Statement 1 creates nodes with label :Team for each row in CSV file
- Statement 2 creates nodes with label :Player for each row in CSV
- Statement 3: For each row in CSV
  - Find the node (p) :Player that has the row's playerID
  - Find the node (t) :Team that has the row's teamID
  - Create a relation :APPEARANCE from (p) to (t) with properties year, games to represent the fact that (p) *appeared* in *games* games in *year* year for team (t)

© Donald F. Ferguson, 2017. All rights reserved.

Find Player with player_id ‘napolmi01’

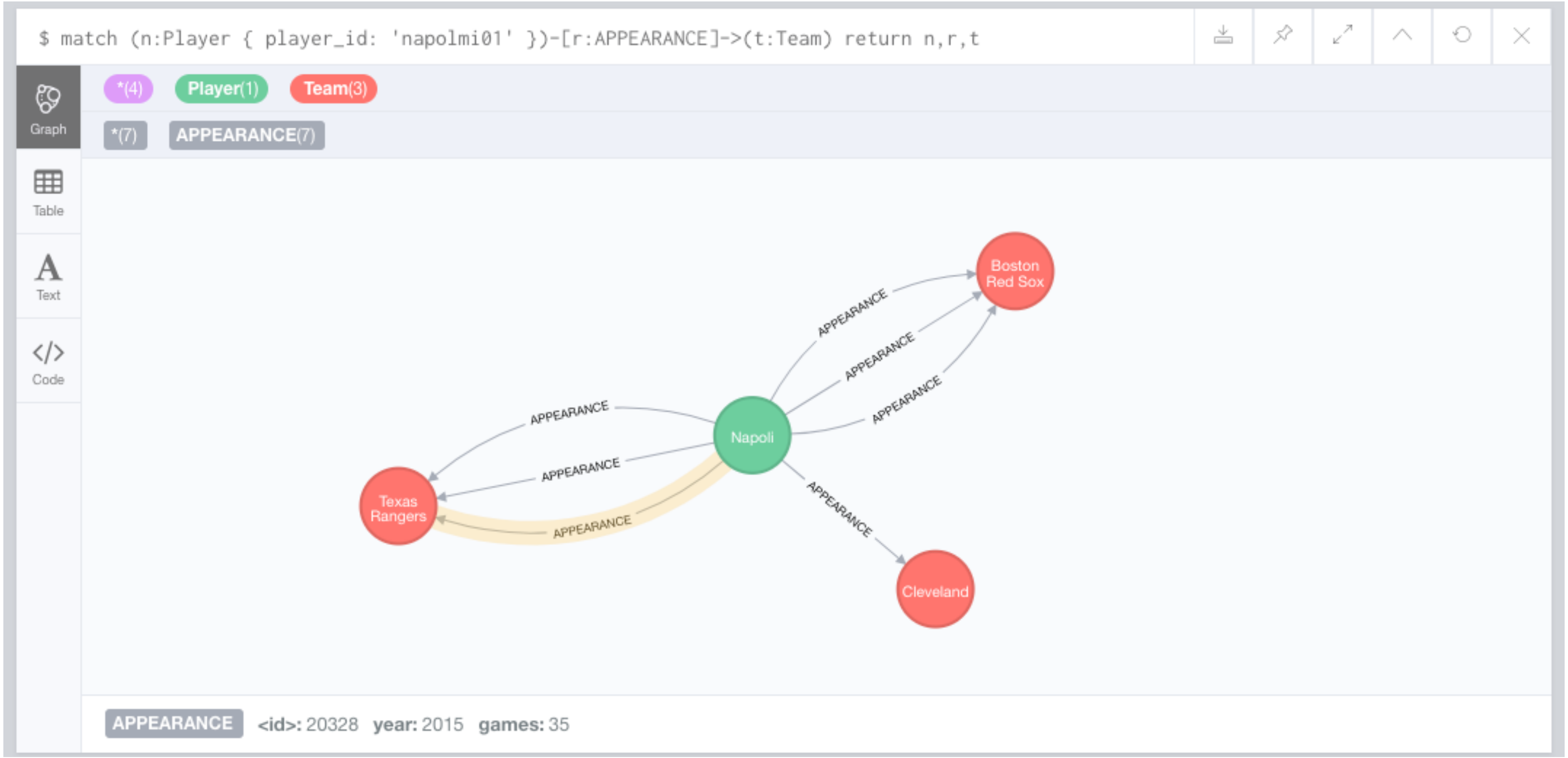


```
$ match (n:Player { player_id: 'napolmi01' }) return n
```

Player <id>: 301765 player_id: napolmi01 birthYear: 1981 birthCountry: USA last_name: Napoli first_name: Mike

© Donald F. Ferguson, 2017. All rights reserved.

Find the Teams/Years for Which Napoli Appeared



```
$ match (n:Player { player_id: 'napolmi01' })-[r:APPEARANCE]->(t:Team) return n,r,t
```

APPEARANCE <id>: 20328 year: 2015 games: 35

© Donald F. Ferguson, 2017. All rights reserved.

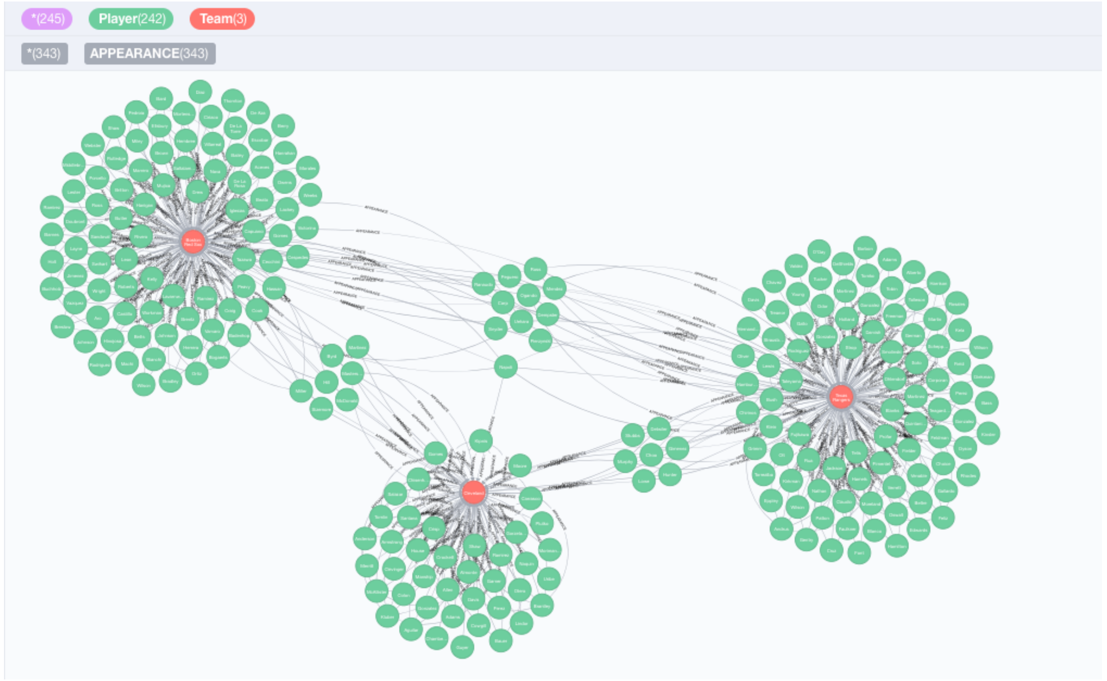
Final (Standalone) Project

- Implement a Players microservice
  - /Players
    - GET (with query params)
    - POST
    - /Players/{playerid}
      - GET
      - PUT
      - DELETE
  - /Players/{id}/teams (GET only)
  - /Players/{id}/teammates
    - GET only.
    - One optional query parameter ?year=1234
    - Returns anyone with whom the player was on a team

© Donald F. Ferguson, 2017. All rights reserved.

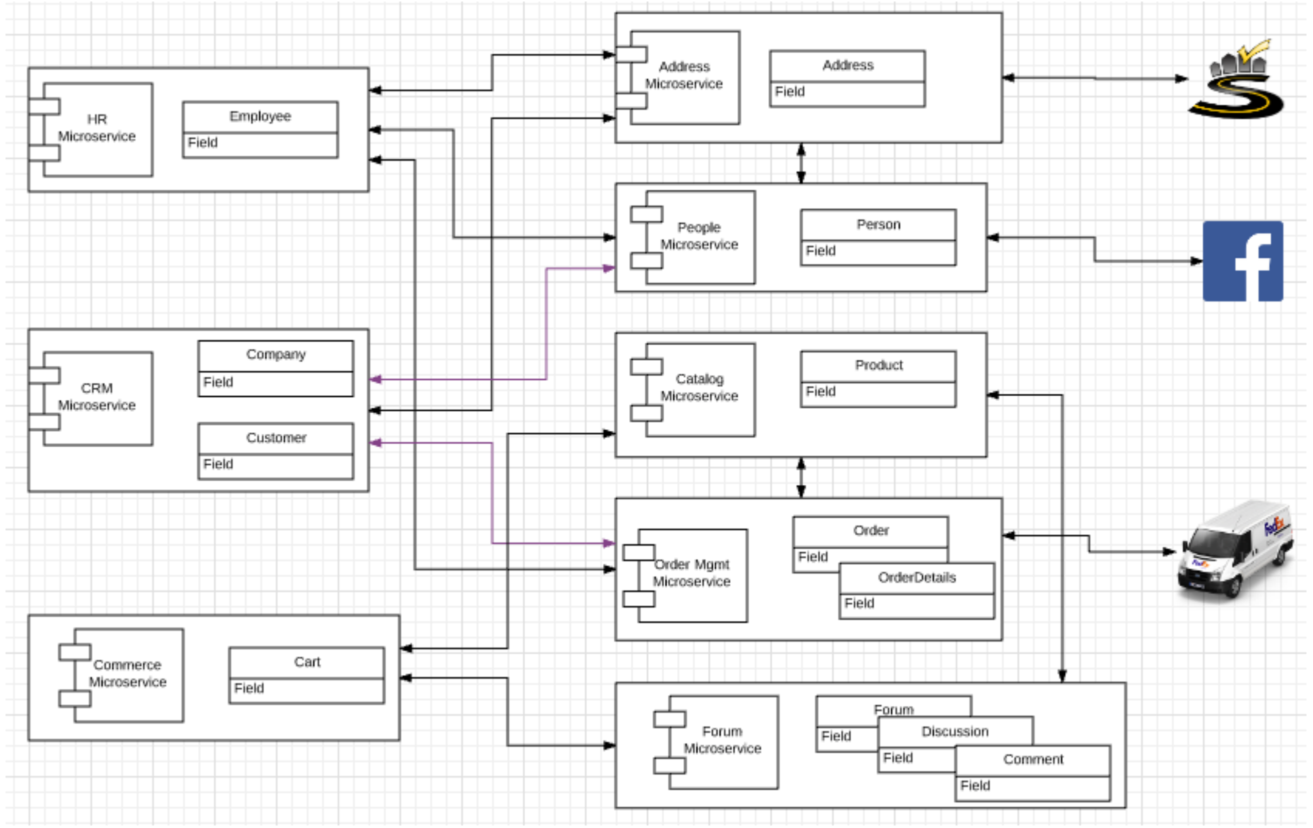
This is Where it Gets Freaky

- Players A and B
  - Were teammates
  - If
    - A and B
    - Both appeared for
      - Any team T
      - In the same year.
- You will
  - Write queries to find
  - All of the teammates of last_name='Napoli'
  - Using Neo4J and SQL



© Donald F. Ferguson, 2017. All rights reserved.

Original Big Picture



© Donald F. Ferguson, 2017. All rights reserved.

Other Projects

© Donald F. Ferguson, 2017. All rights reserved.

0th Project

- Teams
  - Form your teams (approx. 5 people)
  - Identify contact focal point.
  - Give your team a “cool” name.
- Signup (reuse) and Amazon Web Service Account
  - Free Tier should be fine.
  - Provide access to team members.
- Create an Elastic Beanstalk instance/application.
  - Use one of the sample application.
  - Will have to make more sophisticated starting next week;
    I will use Node JS with Express for Elastic Beanstalk examples.

© Donald F. Ferguson, 2017. All rights reserved.

1st Project – Part 1

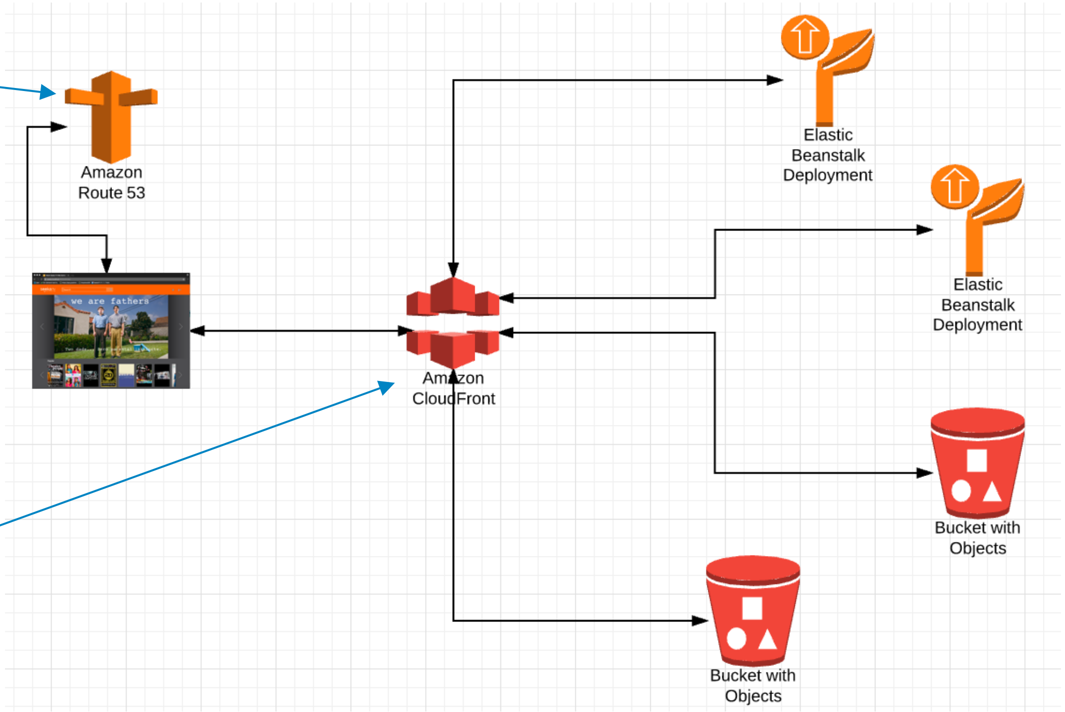
- Implement two distinct microservices
  - Person
  - Address
- Tasks
  - Use Swagger Editor to define and document REST APIs.
  - Implement an Elastic Beanstalk application (microservice) for each resource that implements the relevant REST API.
  - Each microservice should support
    - GET and POST on resource, e.g. /Person
    - GET, PUT, DELETE on resource/id, e.g. /Person/dff9
    - Simple query, e.g. /Person?lastName=Ferguson
    - Pagination
    - Relationship paths: /Person/dff9/address and /Addresses/someID/persons
    - HATEOAS links where appropriate.
  - Simple HTML/Angular demo UI.
- Due: 11:59 PM on 26-Sep-2017

© Donald F. Ferguson, 2017. All rights reserved.

Single Site

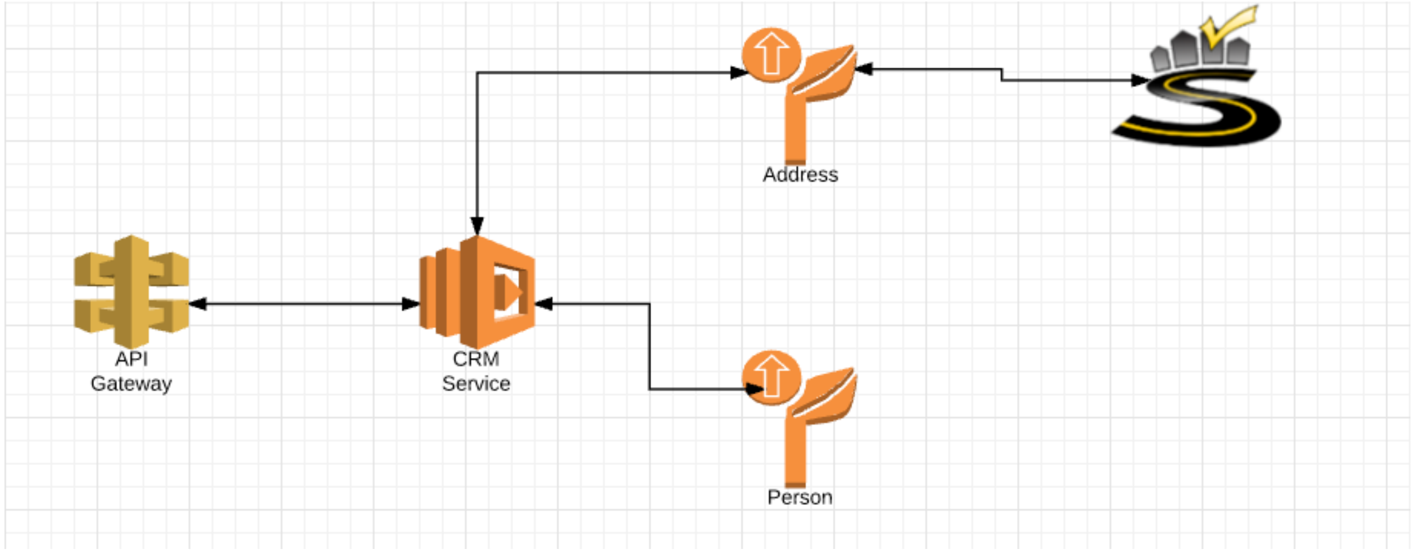
- DNS
  - Resolve dff-company.com to
  - Something.amazon.com
  - Under the covers
- Map
  - /api/person
  - /api/address
  - /app
    - /js
    - /views
    - /app-content
  - /digital-assets
    - /images
    - /videos

To correct IP addresses and sub-paths



© Donald F. Ferguson, 2017. All rights reserved.

Project 2 – Part 1



Build on 1st to microservices

- Implement the CRM Service (and HR Service) using Lambda functions, and orchestration approach from previous lecture.
- Integrate with SmartyStreets
- Deploy all 3 microservices via API Gateway.

• Deliver web content via CloudFront and S3.

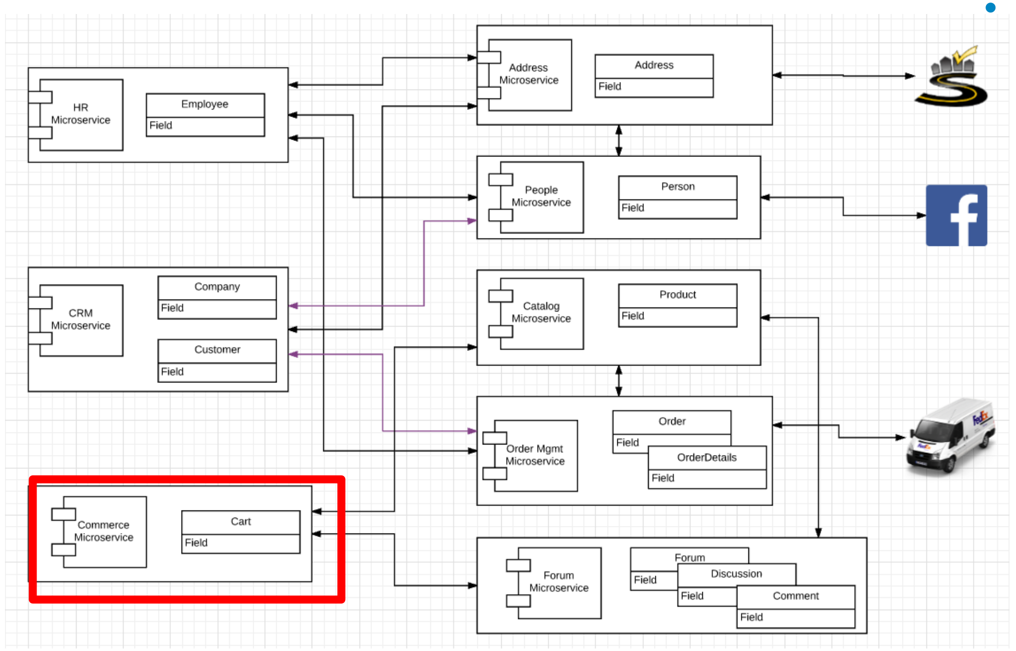
© Donald F. Ferguson, 2017. All rights reserved.

Project 2 – Part 2

- Implement middleware plugins and microservices implementing
  - ETag generation and processing.
  - Idempotent functions.
  - SNS Event generation on PUT, POST and DELETE.
- Plug/deploy the plugin layer in
  - Each of the Beanstalk Microservices
  - The CRM Lambda function.
- Write an empty, placeholder Lambda function that reacts to the SNS events. We will do some interesting things with this later.

© Donald F. Ferguson, 2017. All rights reserved.

Project 3 ~~4~~ – Start/Expand Commerce Microservice



- Use Lambda function

Customers may
- Register with
- And subsequently logon with
- Facebook or Twitter

Just
- Implement register/logon
- We will later
  - Use Step Functions for composing microservices and APIs
  - Publish commerce actions to FB and Twitter.

Also, write
- A placeholder
- API Gateway Custom Authorizer
- Which we will later use to manager authorization to orders.

© Donald F. Ferguson, 2017. All rights reserved.

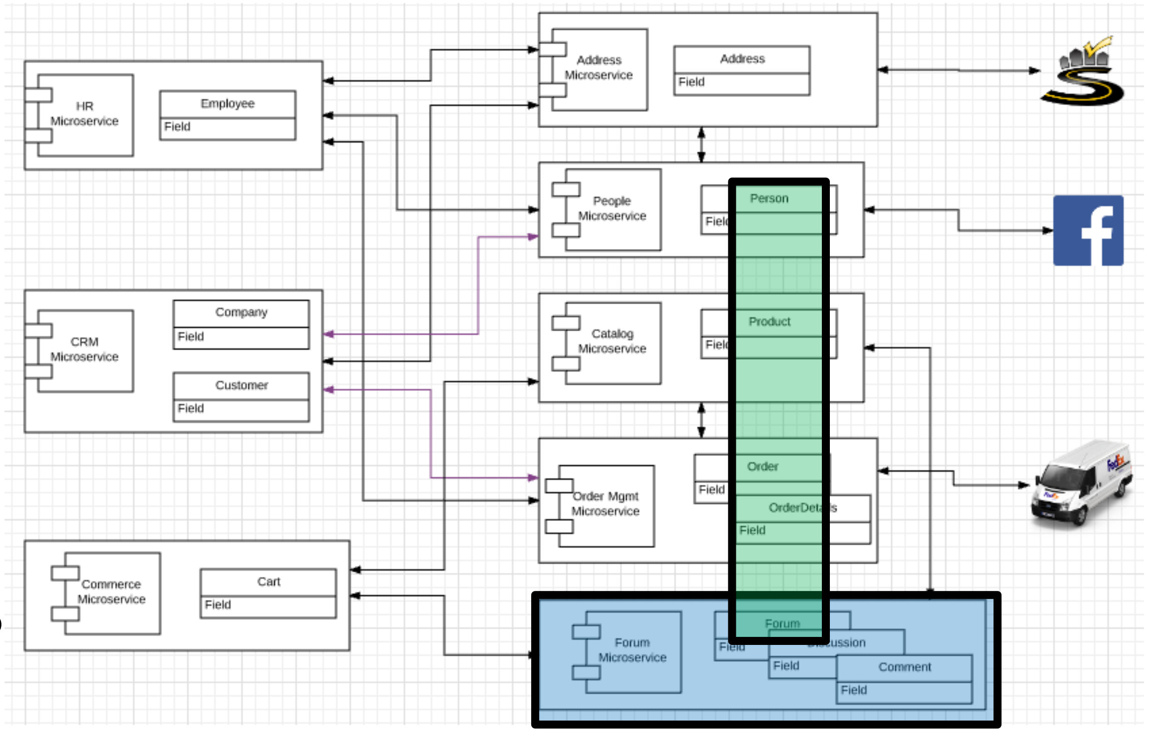
Last Week’s Lecture – CAP and New Datamodels

The initial idea was

- DynamoDB for the forum
- Neo4J to track/query
  - Who bought what?
  - Who has bought things similar to whom?
  - Who commented on what?
  - etc.
- Use Redis to optimize
  - Idempotency
  - Etag

But

- we do not have enough time to do in context of solution.
- Will have to do smaller scenarios and use cases.



© Donald F. Ferguson, 2017. All rights reserved.

My View on Status

Project	Subtask	Element
Project 0	Set up team Run ELB	
Project 1	Implement 2 microservices	• **Implement** • **REST API/HATEOAS** • **Query, relationships**
	Swagger	• **Document, test** • Integrate with API GW, development tools, …
Project 2	Cloud APIs	• **SmartyStreets (UI, backend)**
	Orchestration	• In code
	Single Site Image	• S3 for content • CloudFront • API Gateway
	Middleware	• **ETag and idempotency technical microservice** • **SNS event generation**
Project 3	OAuth2	• **Logon, Register** • **Custom API GW Authorizer → Authorization**
Additional Goals		• **Step functions** • **SNS to integrate registration, commerce views, etc. with Neo4J** • **SNS → Swagger via WebHooks/Lambda** • **Redis: Optimize data sharing for middleware scenarios** • **Rules engines** • **Text search**

© Donald F. Ferguson, 2017. All rights reserved.

Timetable

- Project 2 and project 3
  - Complete (initial) reviews by 05-Dec.
  - Use 27-Nove and 05-Dec lectures if possible, then office hours, then extra OH.
- We will hold one final project review with each team
  - Covering all elements of projects 1, 2 and 3
    - Including modification/correction based on feedback from prior reviews.
    - I will provide a list of completion requirements metrics.
  - Project 4, which has two small subelements
    - Simple idempotency check microservice → Redis
      - POST token returns OK if token not previously seen.
      - POST returns error if token previously recorded.
    - Simple graph microservice on Neo4J using sample data.
      - POST a simple query command in JSON.
      - Returns results from Neo4J.

© Donald F. Ferguson, 2017. All rights reserved.