Moving from Monolithic to Microservices

Rany ElHousieny

Design Goals

- Reduce time/cost to create new ways to use Community Data
- Reduce endpoints
- Improve performance
- Network efficiency
- Mobile and Web Client-friendly interface
- No versioning
- Maintainability
- Resilient to changes
- Real-time usage (not necessary to have a separate cache layer)
- Domain Driven Architecture
- One source of truth for managing communities
- Support all existing functionality of clients
- Extensible for new requirements

Problem Statement

We are not able to fulfill the business needs due to the following issues

a. Monolith DB for all domain
b. Dependency on one DB and one team for all domains
c. Single point of failure
d. Separation of concerns
e. Single responsibility
f. Explosion of end points, lack of documentations, slow performance
g. Separation of domains
h. Non flexible design and not easily extendable without a re-write (PC - CC)
i. No separation between FE and BE
j. Upgrades and maintainability

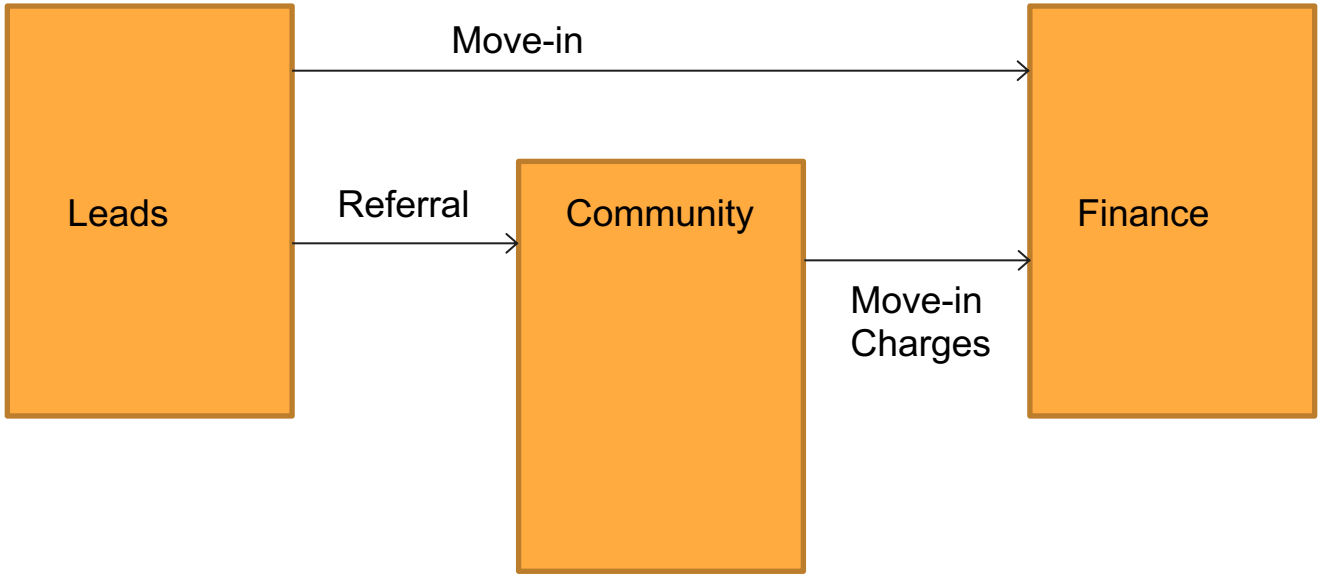
Design Principles

- Client should not need to know application specific details to use the API
- The API will be the source of truth for Community Data used by all APFM applications
- Single responsibility for API, only works with communities
- Open for addition, closed for modification
- Should support progressive migration of data out of YGL
- API should be loosely coupled from other systems
- Should be fault tolerant and can scale automatically
- Should support consumer facing web workloads
- Design for availability and partition tolerance and eventually consistent
- Support concurrency
- Should be secure, with authentication and authorization
    - Role based filtering of data
- It should be cost efficient
- It should contain logging and instrumentation
- Network efficiency, one call, minimize client aggregation and processing of data
- It should be easy or automatically documented, should be discoverable
- It should be testable by engineers

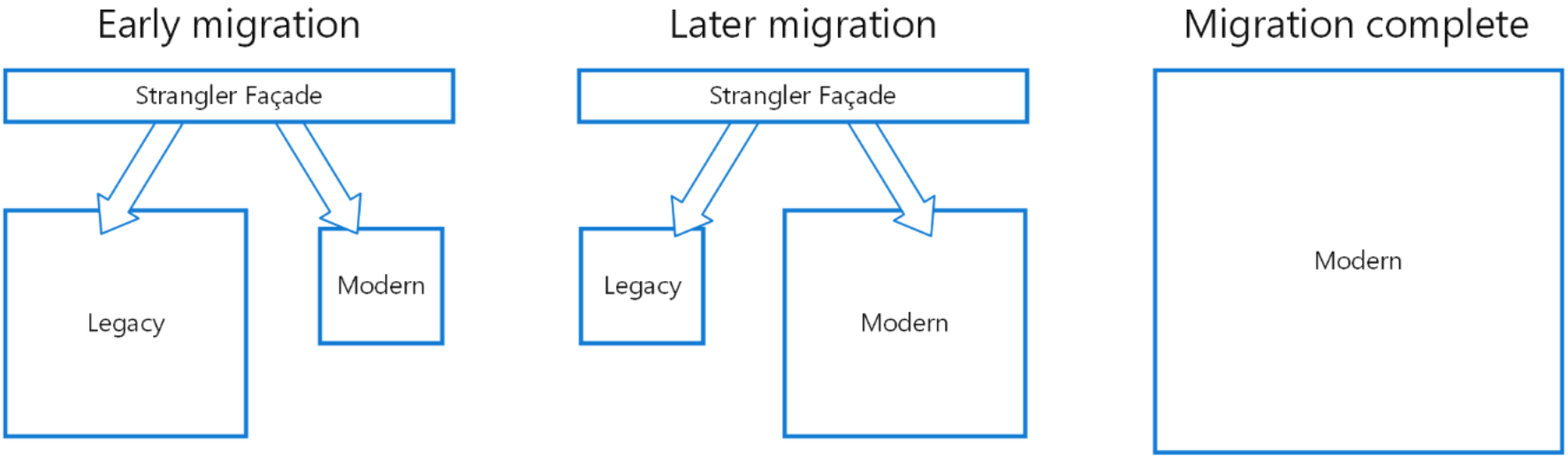
Design

Event Storming

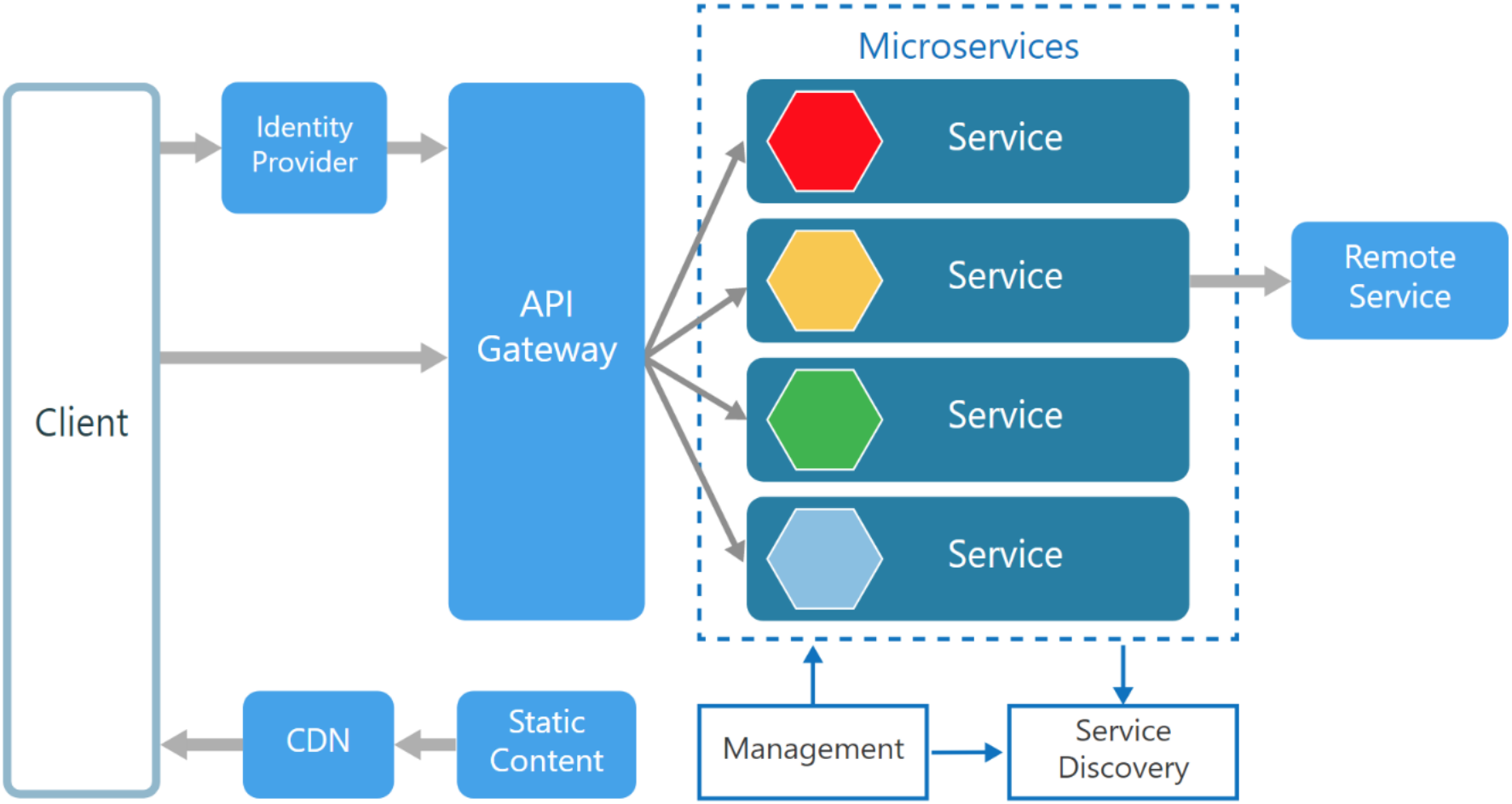
Domains and Bounded Context



The Strangler Pattern



Ref: https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler


Microservices
Client
Identity Provider
API Gateway
Service
Service
Service
Service
Remote Service
CDN
Static Content
Management
Service Discovery

REST vs GraphQL

REST

1. Current approach to most APIs with problems
2. Explosion of endpoints
3. Hard to keep track of documentation
4. Hard to standardize
5. Not network efficient
6. Multiple trips
7. Over/Under fetching
8. Versioning issues

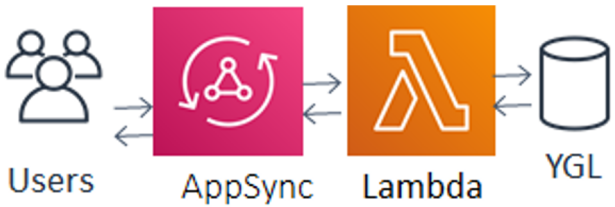
GraphQL

1. Better performance
2. One endpoint
3. Automatic documentation with GraphiQL with the same endpoint URL/GraphiQL
4. Standard Query interface
5. Network efficient
6. One single round trip
7. Clients get exactly what they ask for even from multiple sources
8. No need for versioning just introduce and deprecate fields

Containers vs Serverless

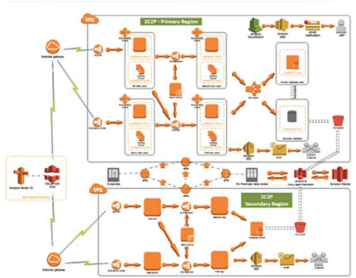
Serverless:

1. Autoscale
2. No cost for idle
3. Out of the box full solution: No need to build and manage containers, servers, APIs, events, upgrades ...



Build and Manage our own solution:

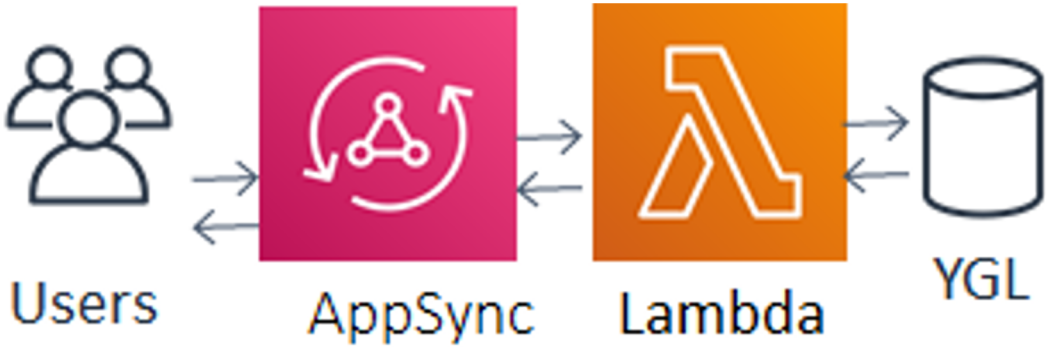
1. We have to build with LB and Scaling groups
2. With redundancy we pay for ideal
3. We have to interrupt to upgrade
    a. We have to manage operating systems
    b. We have to build, maintain, and upgrade Applications [SQL upgrade, PC …]
    c. Application can be obsolete and need to be rewritten
    d. You have to manage, queues messaging, schema, data sync



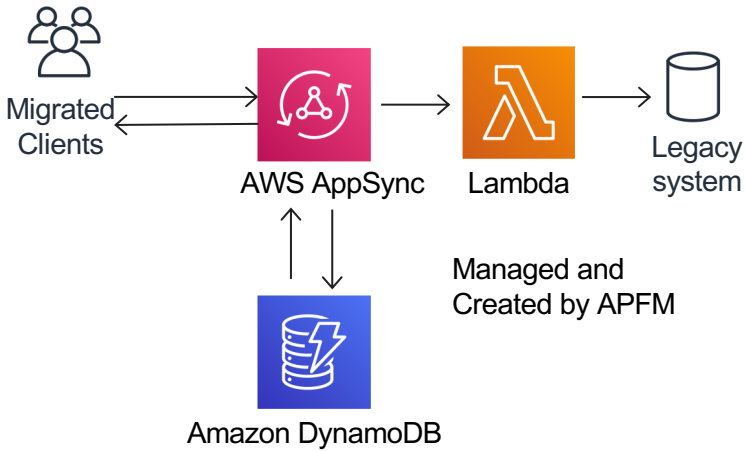
Proof of Concept

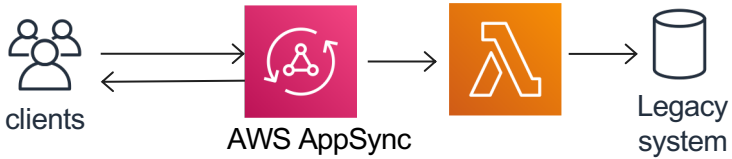
Using AppSync

Proof of Concept



With this design, we only provide schema/Types and Lambdas to integrate with SQL


Migrated
Clients
AWS AppSync
Lambda
Legacy
system
Managed and
Created by APFM
Amazon DynamoDB
Managed and
Created by AWS



Managed and
Created by AWS

Using AWS Amplify for POC

Rany

```
86
87  type Employee @model {
88      id: ID!
89      createdOn: String!
90      updatedOn: String
91      firstName: String
92      lastName: String
93      role: String
94      pcLastLoggedInDate: String
95      employedBy: [EmploymentEdge]
96      phones: [Phone]
97      emailAddresses: [Email]
98  }
```

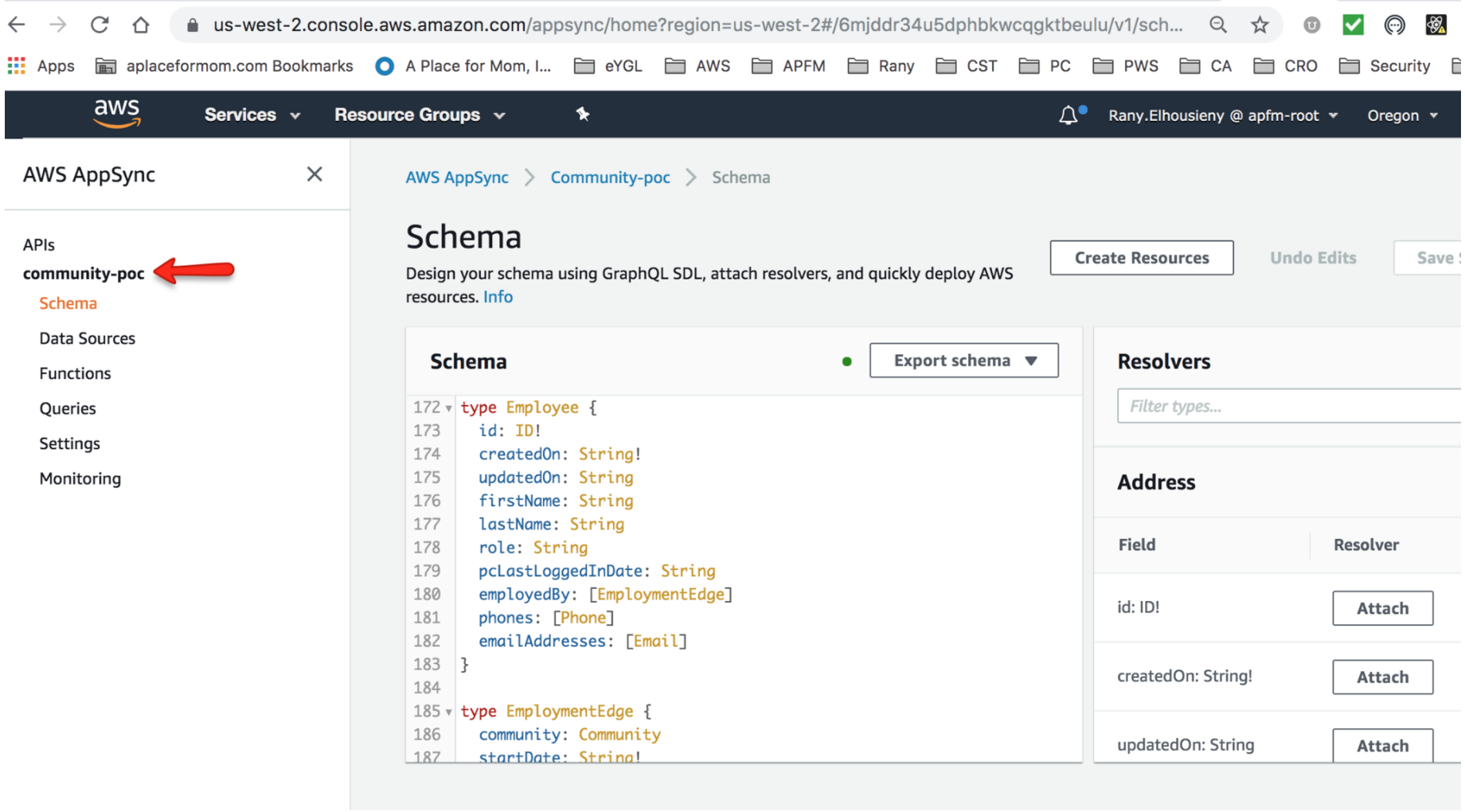
@model will create DynamoDB object and link to this Type Automatically

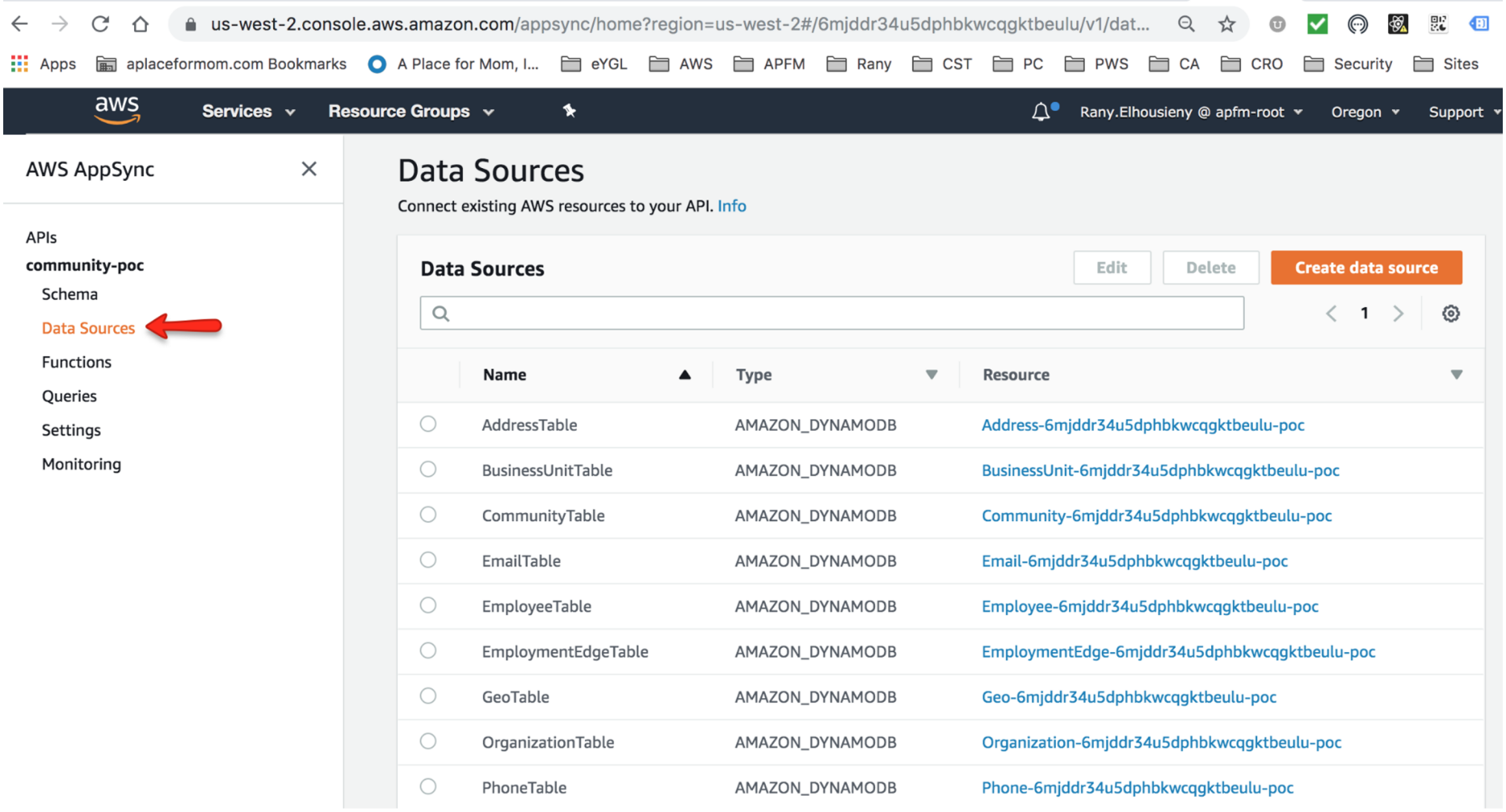
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

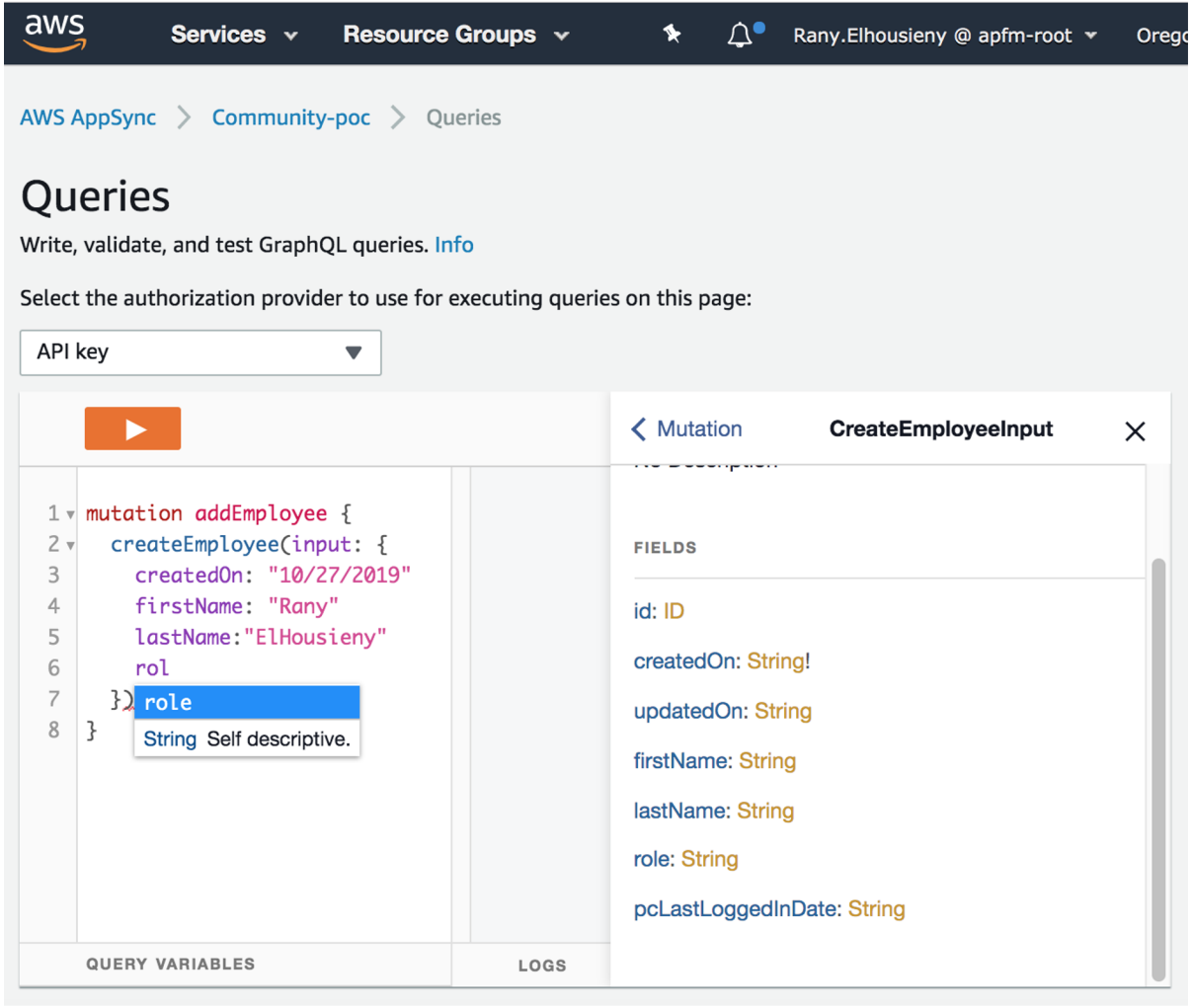
```
RanyMac:community ranyelhousieny$ amplify add api
? Please select from one of the below mentioned services GraphQL
? Provide API name: community
? Choose the default authorization type for the API API key
? Enter a description for the API key: communityapi
```

```
RanyMac:community ranyelhousieny$ amplify push

Current Environment: poc

| Category | Resource name | Operation | Provider plugin   |
| -------- | ------------- | --------- | ----------------- |
| Api      | community     | Create    | awscloudformation |
? Are you sure you want to continue? Yes
```

```
CREATE_COMPLETE apicommunity AWS::CloudFormation::Stack Sun Oct 27 2019
15:17:46 GMT-0700 (Pacific Daylight Time)
⠸ Updating resources in the cloud. This may take a few minutes...

UPDATE_COMPLETE                         community-poc-20191027145223 AWS::Cl
oudFormation::Stack Sun Oct 27 2019 15:17:49 GMT-0700 (Pacific Daylight
Time)
UPDATE_COMPLETE_CLEANUP_IN_PROGRESS community-poc-20191027145223 AWS::Cl
oudFormation::Stack Sun Oct 27 2019 15:17:48 GMT-0700 (Pacific Daylight
Time)
✔ Generated GraphQL operations successfully and saved at src/graphql
✔ All resources are updated in the cloud

GraphQL endpoint: https://44pic2336fafxp4k7exb4jxwiq.appsync-api.us-west
-2.amazonaws.com/graphql
```

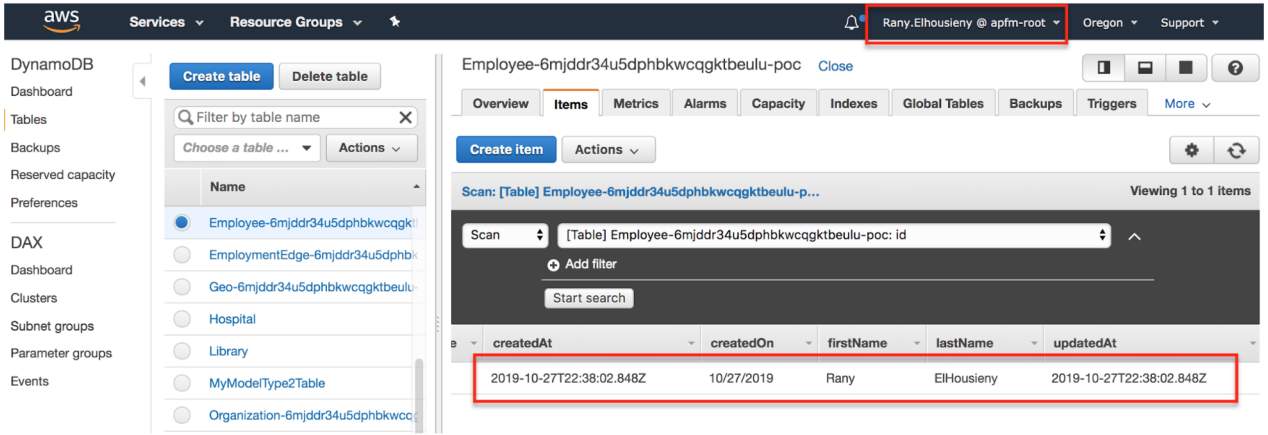

us-west-2.console.aws.amazon.com/appsync/home?region=us-west-2#/6mjddr34u5dphbkwcqgktbeulu/v1/sch...
Apps
aplaceformom.com Bookmarks
A Place for Mom, I...
eYGL
AWS
APFM
Rany
CST
PC
PWS
CA
CRO
Security
aws
Services
Resource Groups
Rany.Elhousieny @ apfm-root
Oregon
AWS AppSync
APIs
community-poc
Schema
Data Sources
Functions
Queries
Settings
Monitoring
AWS AppSync > Community-poc > Schema
Schema
Design your schema using GraphQL SDL, attach resolvers, and quickly deploy AWS resources. Info
Create Resources
Undo Edits
Save
Schema
Export schema
172 type Employee {
173 id: ID!
174 createdOn: String!
175 updatedOn: String
176 firstName: String
177 lastName: String
178 role: String
179 pcLastLoggedInDate: String
180 employedBy: [EmploymentEdge]
181 phones: [Phone]
182 emailAddresses: [Email]
183 }
184
185 type EmploymentEdge {
186 community: Community
187 startDate: String!
Resolvers
Filter types...
Address
Field
Resolver
id: ID!
Attach
createdOn: String!
Attach
updatedOn: String
Attach


us-west-2.console.aws.amazon.com/appsync/home?region=us-west-2#/6mjddr34u5dphbkwcqgktbeulu/v1/dat...
Apps
aplaceformom.com Bookmarks
A Place for Mom, I...
eYGL
AWS
APFM
Rany
CST
PC
PWS
CA
CRO
Security
Sites
aws
Services
Resource Groups
Rany.Elhousieny @ apfm-root
Oregon
Support
AWS AppSync
APIs
community-poc
Schema
Data Sources
Functions
Queries
Settings
Monitoring
Data Sources
Connect existing AWS resources to your API. Info
Data Sources
Edit
Delete
Create data source
1
Name
Type
Resource
AddressTable AMAZON_DYNAMODB Address-6mjddr34u5dphbkwcqgktbeulu-poc
BusinessUnitTable AMAZON_DYNAMODB BusinessUnit-6mjddr34u5dphbkwcqgktbeulu-poc
CommunityTable AMAZON_DYNAMODB Community-6mjddr34u5dphbkwcqgktbeulu-poc
EmailTable AMAZON_DYNAMODB Email-6mjddr34u5dphbkwcqgktbeulu-poc
EmployeeTable AMAZON_DYNAMODB Employee-6mjddr34u5dphbkwcqgktbeulu-poc
EmploymentEdgeTable AMAZON_DYNAMODB EmploymentEdge-6mjddr34u5dphbkwcqgktbeulu-poc
GeoTable AMAZON_DYNAMODB Geo-6mjddr34u5dphbkwcqgktbeulu-poc
OrganizationTable AMAZON_DYNAMODB Organization-6mjddr34u5dphbkwcqgktbeulu-poc
PhoneTable AMAZON_DYNAMODB Phone-6mjddr34u5dphbkwcqgktbeulu-poc


aws
Services
Resource Groups
Rany.Elhousieny @ apfm-root
AWS AppSync
Community-poc
Queries
Queries
Write, validate, and test GraphQL queries. Info
Select the authorization provider to use for executing queries on this page:
API key
mutation addEmployee {
createEmployee(input: {
createdOn: "10/27/2019"
firstName: "Rany"
lastName:"ElHousieny"
rol
role
String Self descriptive.
})
}
QUERY VARIABLES
LOGS
Mutation
CreateEmployeeInput
FIELDS
id: ID
createdOn: String!
updatedOn: String
firstName: String
lastName: String
role: String
pcLastLoggedInDate: String

```
mutation addEmployee {
  createEmployee(input: {
    createdOn: "10/27/2019"
    firstName: "Rany"
    lastName:"ElHousieny"
  }) {
    id
  }
}
```

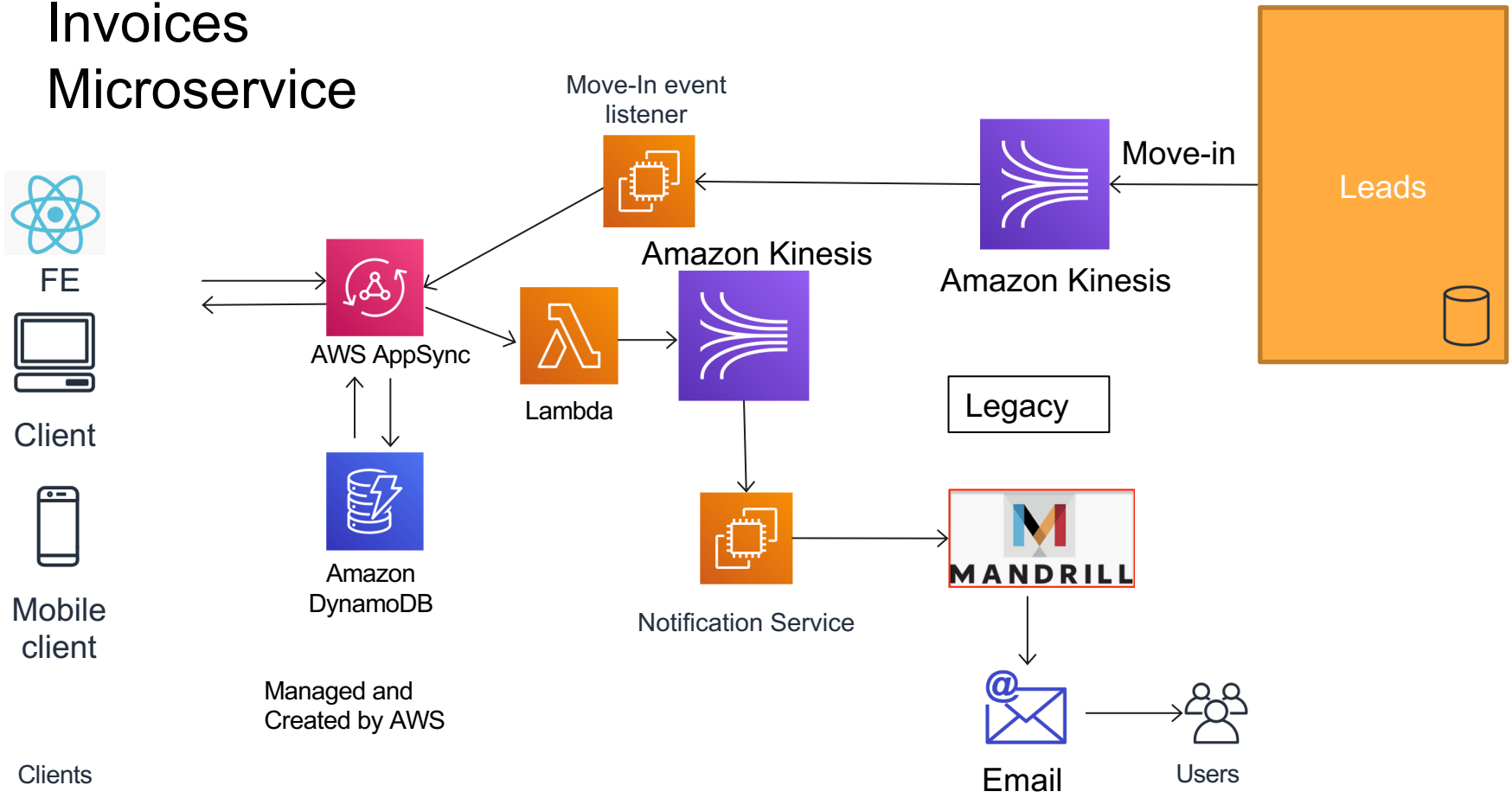
```
{
  "data": {
    "createEmployee": {
      "id": "fdc16e84-702c-4cc3-aa69-cd820a60207f"
    }
  }
}
```



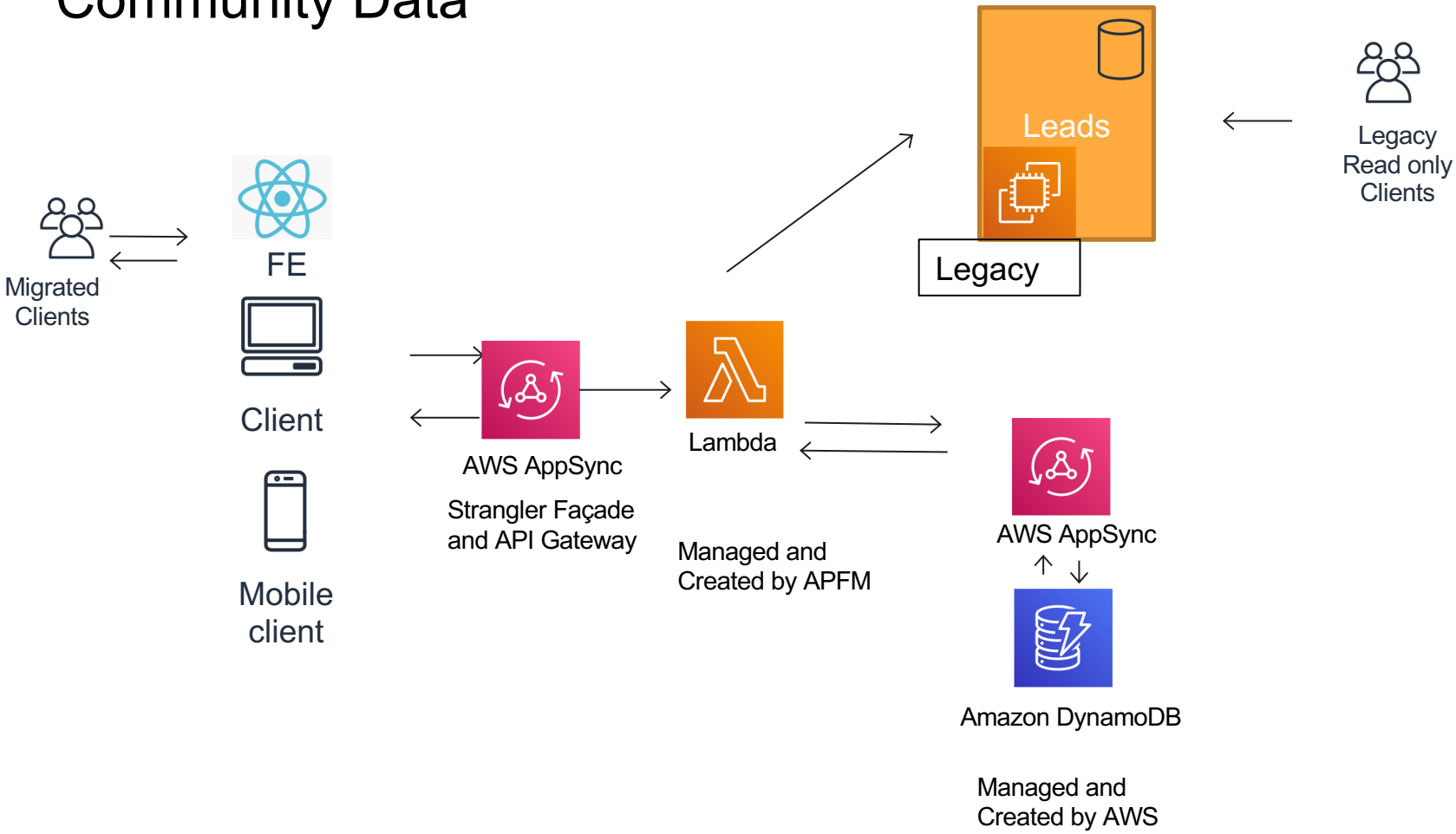
First Microservice

Invoicing Microservice
Independent internal DB
1400 hours/month

Invoices Microservice



Community Data



Clients:

For Community data we have two methods:

1. Read:
    a. Websites
    b. Community Central
    c. Affiliates
    d. Partners [Websites, Community Central]
    e. Internal teams: SLAs, Finance, Customer Enablement
2. Write:
    a. Partners [Community Central]
    b. SEO / content editors [CMS]
    c. internal teams: SLAs, Finance, Customer Enablement, SalesForce

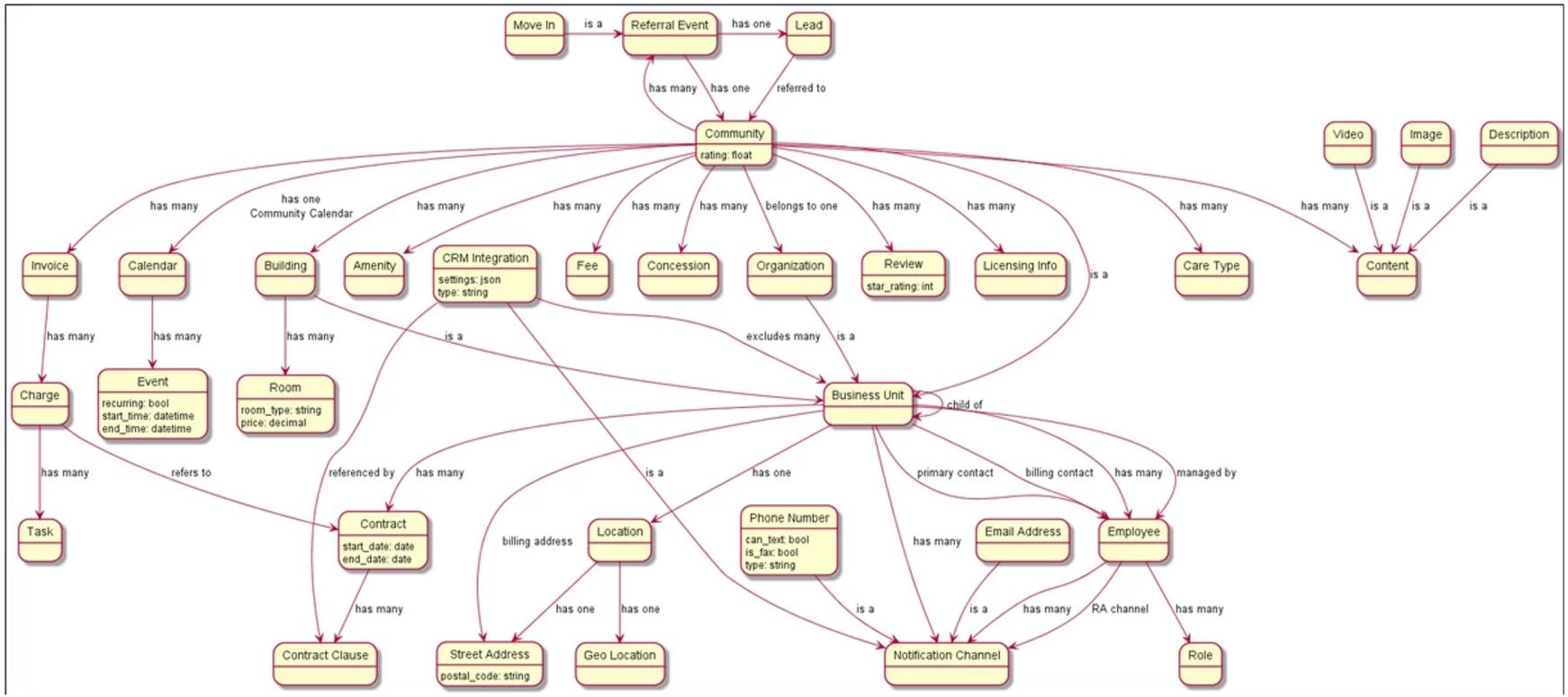
User Stories:

- As a family, I would like to search by location
- As a family, I would like to search by price
- As a family, I would like to search by amenities
- As an SLA, I would like to filter by care type
- As an SLA, I would like to find the primary contact for a community
- As an SLA, I would like to find pricing and fee information for a community
- As a referral service, I would like to find referral recipients for a community
- As a community, I would like to manage my employee contacts
- As a community, I would like to set referral recipients

User Stories (Cont.)

- As an account manager, I would like to manage community employees
- As an account manager, I would like to set referral recipients
- As an account manager, I would like to manage organization hierarchies
- As finance, I would like to associate contracts with an organization
- As finance, I would like to associate contracts with a community

Representation of Data



Graph Schema (Types)

We started with Employee

Sample Type: Employee

```
type Employee implements Node {
    id: ID!
    createdOn: String!
    updatedOn: String
    createdBy: String
    updatedBy: String

    address: Address
    emailAddresses: [Email]
    employedBy: [EmploymentEdge]
    firstName: String!
    jobTitle: String
    lastName: String
    pcAccountStatus: String
    pcLastLoggedInDate: String
    phones: [Phone]
}
```

```
type EmploymentEdge {
    community: Community
    endDate: Date
    isPrimaryContactForCommunity: Boolean
    receivesReferralAlerts: Boolean
    referralEmailIds: [ID]
    referralFaxIds: [ID]
    role: String
    startDate: Date!
}

type Query {
    community(id: ID!): Community
    employee(id: ID!): Employee
}

type Mutation {
    createEmployee(input: EmployeeInput):
Employee
}
```

Next Steps

CI/CD
Form The Team

Implement Employee Type

Migrate all Employee APIs calls to the new Endpoint