Securing Serverless Applications
Part 1

Using Amazon API Gateway, AWS Lambda and Amazon Cognito

Karthi Thyagarajan
Enterprise Solutions Architect

Imagine for a minute…

Being able to develop a mobile backend API that:

- Requires no infrastructure
- Scales automatically to meet demand
- Has granular costs that grow with usage

The services we are going to use

AWS Lambda

Execute our app's business logic

Amazon API Gateway

Host the API and route API calls

Amazon Cognito

Generate temporary AWS credentials

Amazon DynamoDB

Data store

What to Expect from the Session

1. Start with a basic 3-tier web app
    - Pure serverless
2. Add authentication with **Amazon Cognito**
    - Integrate with Cognito
    - Login by leveraging BYOI (bring your own identity)
3. Authorization with **AWS IAM**
4. Segue to part 2

Key takeaways

1. AWS Lambda + Amazon API Gateway means no infrastructure to manage – we scale for you

2. Security is important, and complex – make the most of AWS Identity and Access Management by leveraging Cognito

3. Flexibility – API Gateway, Lambda and Cognito give you choices for authentication and authorization

First building block:
AWS Lambda

AWS Lambda Overview

Lambda functions: Stateless, trigger-based code execution

1

No Infrastructure to manage

Focus on business logic, not infrastructure. You upload code; AWS Lambda handles everything else.

2

High performance at any scale; Cost-effective and efficient

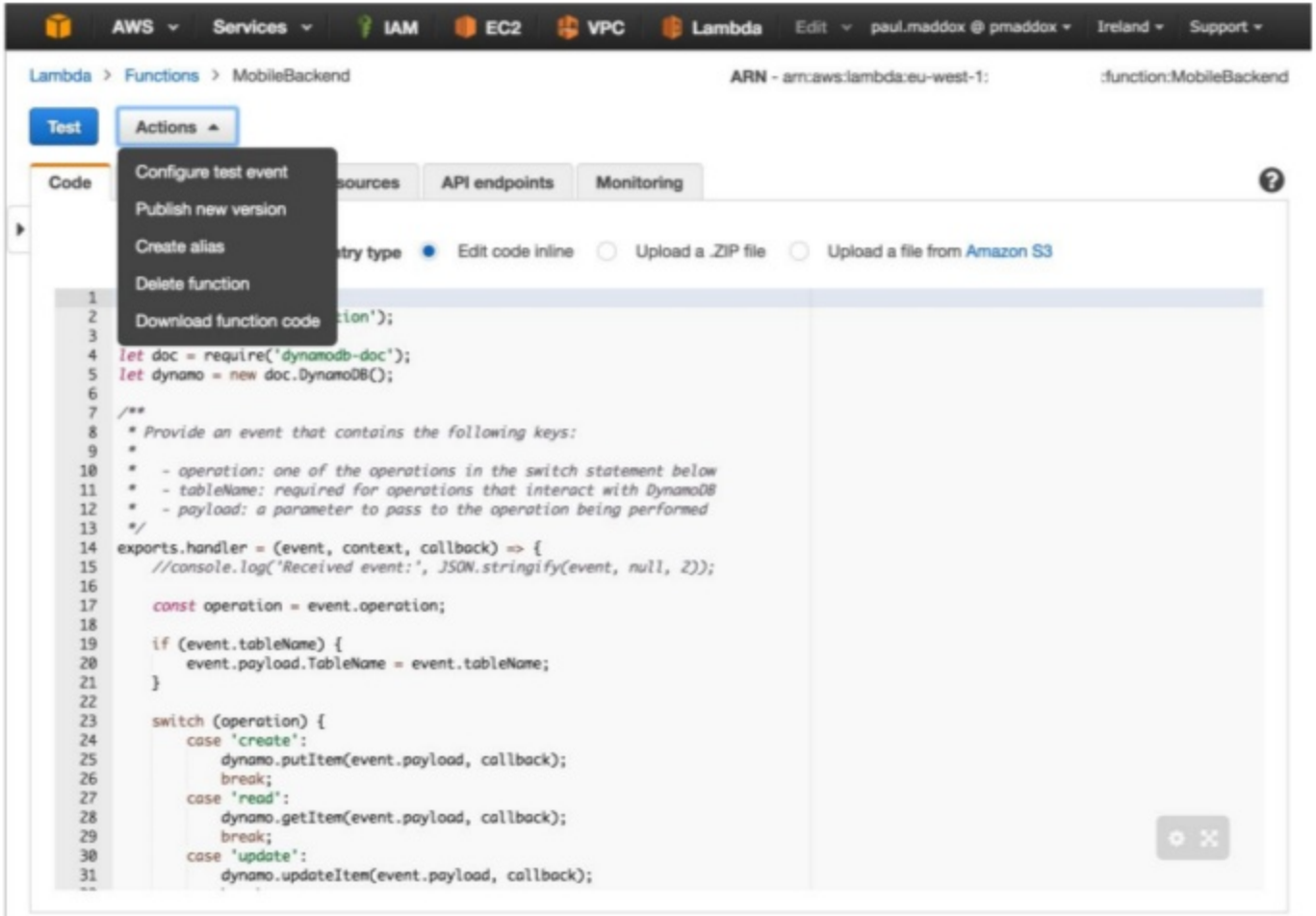
Pay only for what you use: Lambda automatically matches capacity to your request rate. Purchase compute in 100ms increments.

3

Bring Your Own Code

Run code in a choice of standard languages. Use threads, processes, files, and shell scripts normally.

AWS Lambda Console



Develop, test and publish your Lambda functions either by the **AWS Management Console**, **AWS CLI** or our **SDKs**.

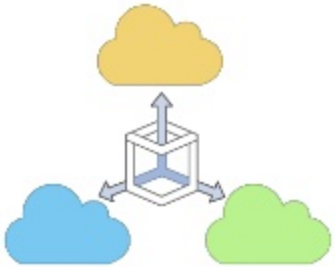
Or use community frameworks such as **serverless.com**, **gosparta.io** and more…

Second building block:
Amazon API Gateway

Amazon API Gateway overview

1

Define and host APIs



Manage deployments to multiple versions and environments

2

Manage network traffic

DDoS protection and request throttling to safeguard your back end

3

Leverage AWS Auth

Leverage Identity and Access Management to authorize access to your cloud resources

Your Feedback

Managing multiple versions and stages of an API is difficult

Monitoring 3rd party developers' access is time consuming

Access authorization is a challenge

Traffic spikes create operational burden

What if I don't want servers at all?

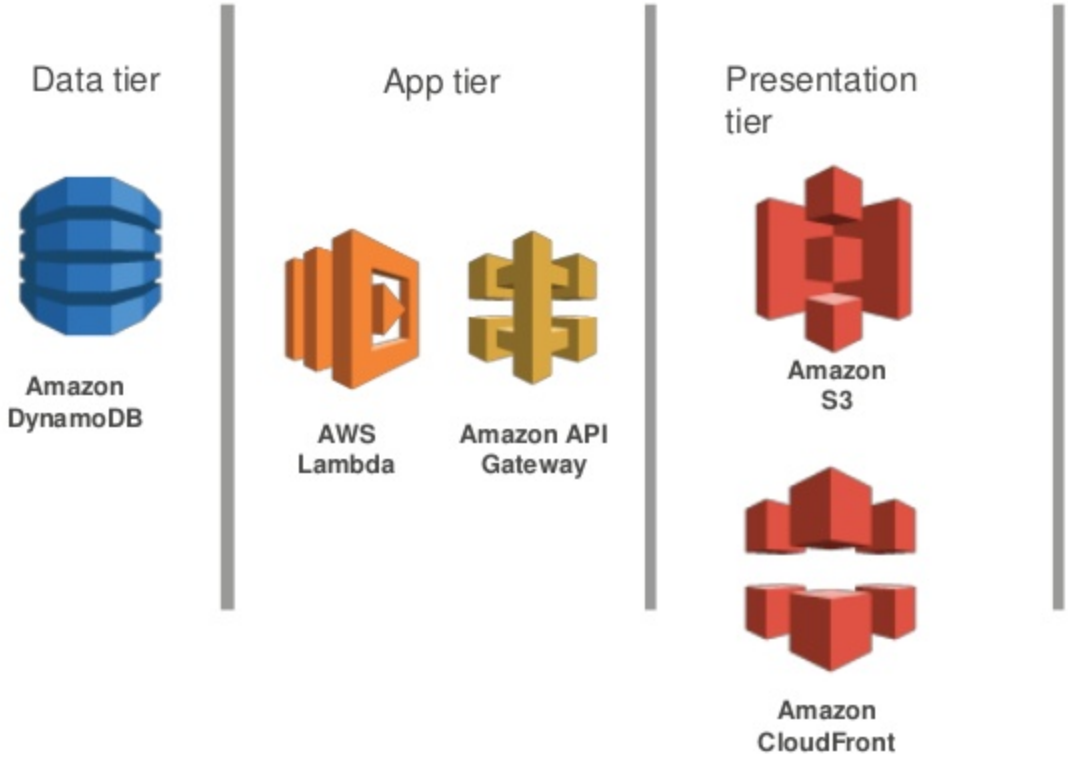
Introducing Amazon API Gateway

- Host multiple versions and stages of your APIs
- Create and distribute API Keys to developers
- Authenticate and authorise API consumers
- Throttle and monitor requests to protect your backend
- Utilizes AWS Lambda 

Introducing Amazon API Gateway

- Managed cache to store API responses
- Reduced latency and DDoS protection through CloudFront
- SDK Generation for iOS, Android and JavaScript
- Swagger import and export support
- Request / Response data transformation and API mocking

Serverless 3-tier Web Architecture



Demo – notes app

Third building block:
Amazon Cognito

Amazon Cognito overview

1

Identity management

Manage authenticated and guest users across identity providers

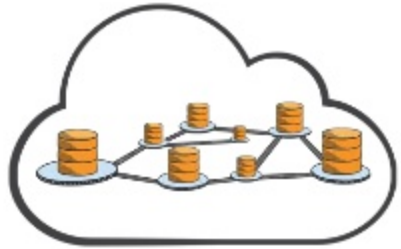
2

Secure AWS access

Securely access AWS services from mobile devices and platforms

3

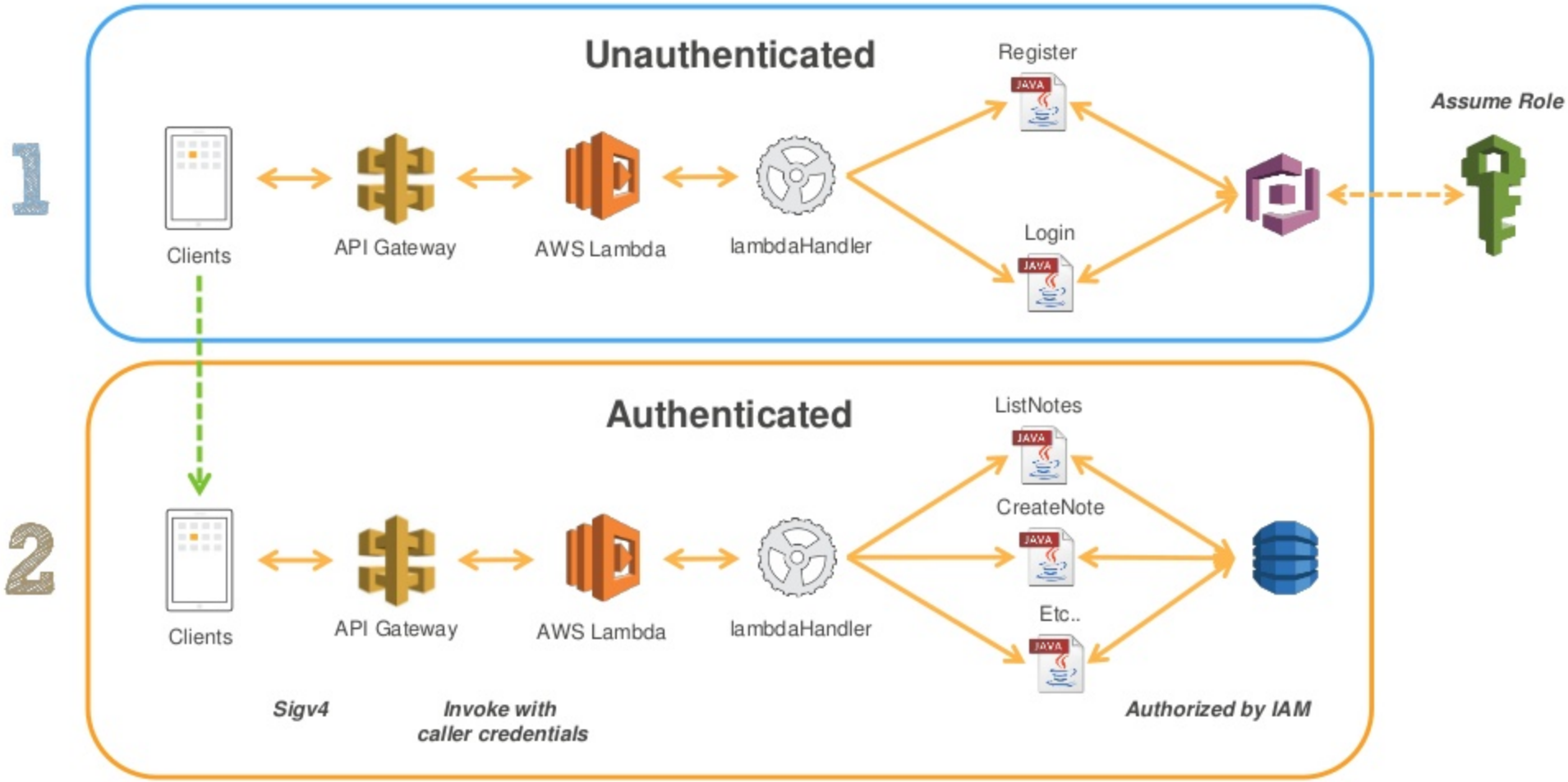
Data synchronization



Synchronize users' data across devices and platforms via the cloud

The notes API

API call flows



Retrieving AWS credentials

The API definition

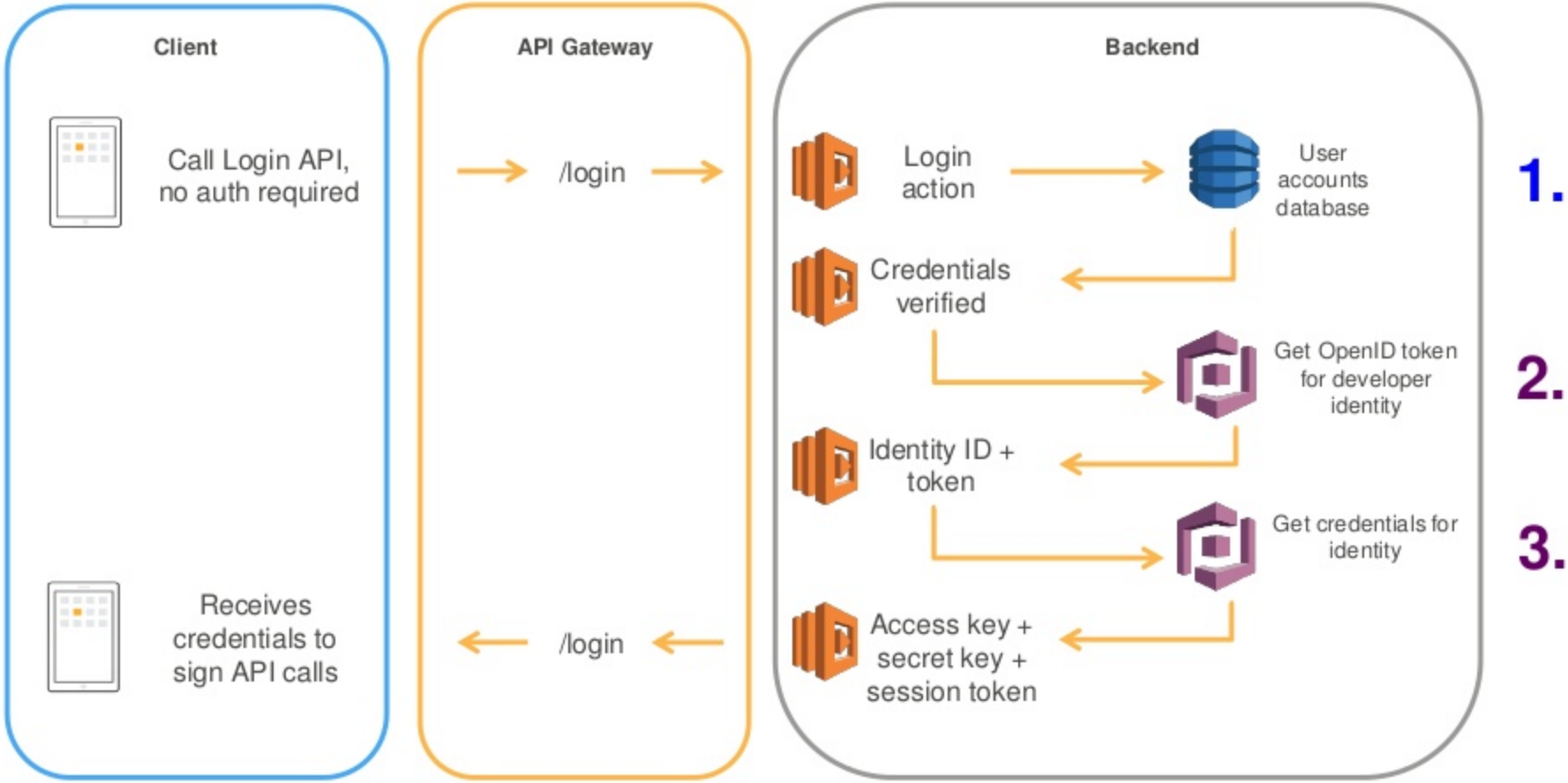
/users

- **POST**
  - Receives a user name and password
  - Encrypts the password and creates the user account in DynamoDB
  - Calls Amazon Cognito to generate credentials
  - Returns the user + its credentials

/login

- **POST**
  - Receives a user name and password
  - Authenticates the user against the DynamoDB database
  - Calls Amazon Cognito to generate credentials
  - Returns a set of temporary credentials

Retrieving temporary AWS credentials



Demo – login with Cognito

Authorizing API calls

The Pets resources require authorization

/pets

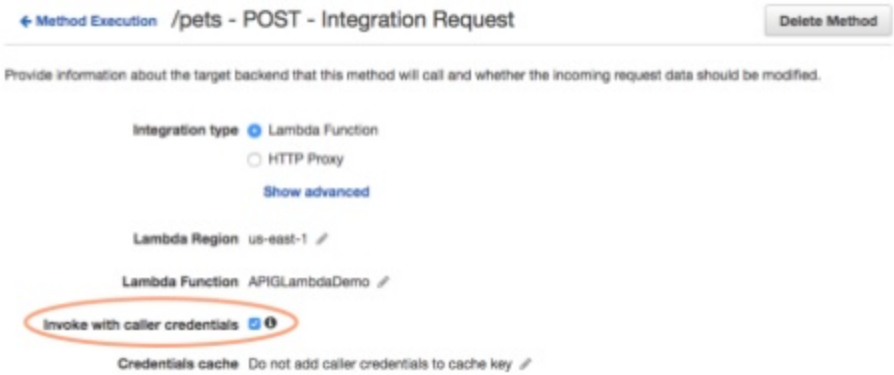
- **POST**
  - Receives a Pet model
  - Saves it in DynamoDB
  - Returns the new Pet ID
- **GET**
  - Returns the list of Pets stored in DynamoDB

/pets/{petId}

- **GET**
  - Receives a Pet ID from the path
  - Uses mapping templates to pass the path parameter to the Lambda function
  - Loads the Pet from DynamoDB
  - Returns a Pet model

Using the caller credentials

Using the console



Using Swagger

```
credentials:

arn:aws:iam::*:user/*
```

The IAM role defines access permissions

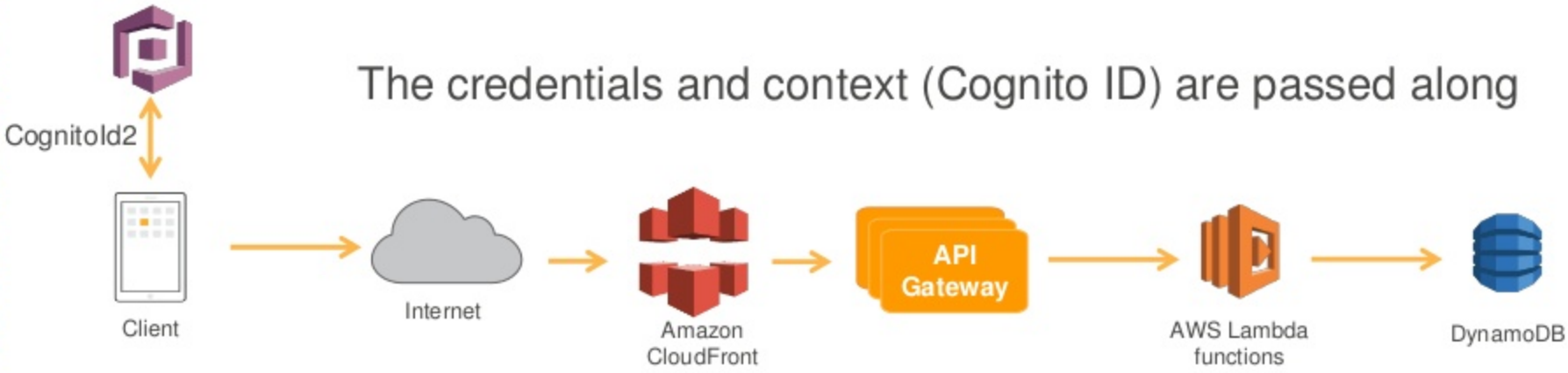
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:PutItem",
          "dynamodb:Scan",
          "lambda:InvokeFunction",
          "execute-api:invoke"
        ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:xxxxxx:table/notes",
        "arn:aws:lambda:us-east-1:xxxxx:function:NotesGet”,
        "arn:aws:execute-api:us-east-1:xxxx:API_ID/*/POST/notes"
      ]
    }
  ]
}
```

The role allows calls to:

- DynamoDB
- API Gateway
- Lambda

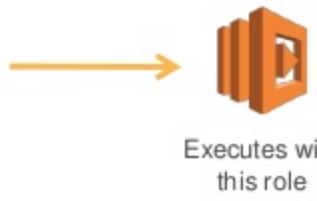
The role can access specific resources in these services

One step further: Fine-grained access permissions



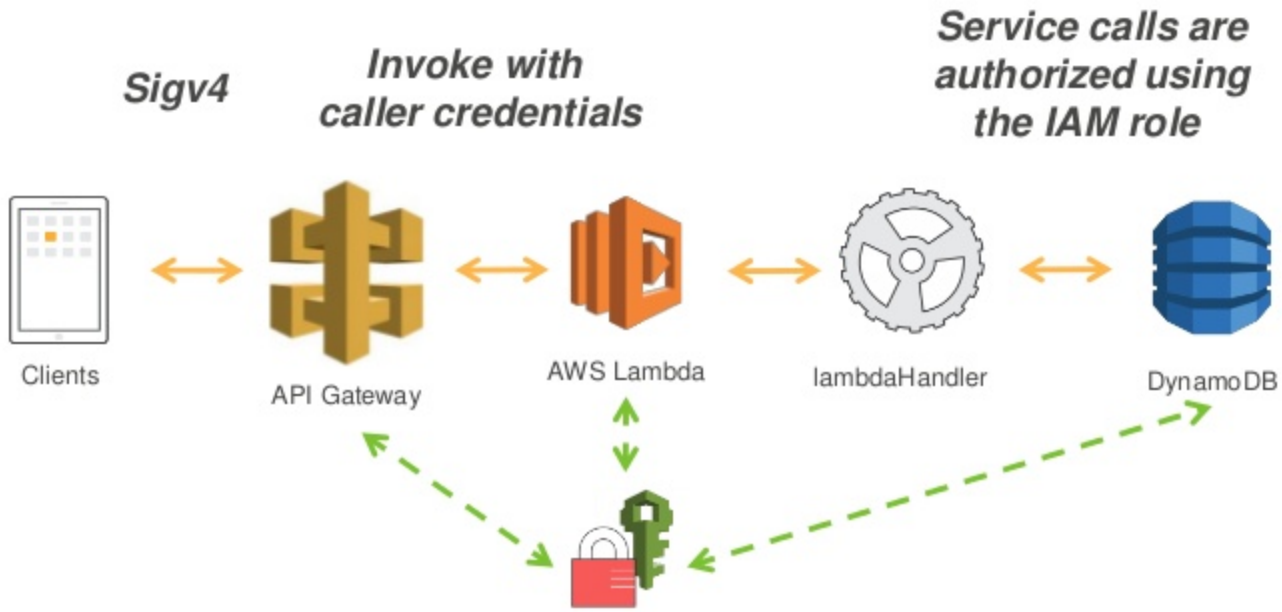
Both AWS Lambda & DynamoDB will follow the access policy

```
…
"Condition": {
    "ForAllValues:StringEquals": {
        "dynamodb:LeadingKeys":  ["${cognito-
identity.amazonaws.com:sub}"],
        "dynamodb:Attributes": [
            "UserId","GameTitle","Wins","Losses",
            "TopScore","TopScoreDateTime"
        ]
    },
    "StringEqualsIfExists": {
        "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
    }
}
…
```



	UserID	Wins	Losses
✗	cognitoId1	3	2
✓	cognitoId2	5	8
✗	cognitoId3	2	3

Authenticated flow in depth



Learn more about fine-grained access permissions

http://amzn.to/1YkxcjR

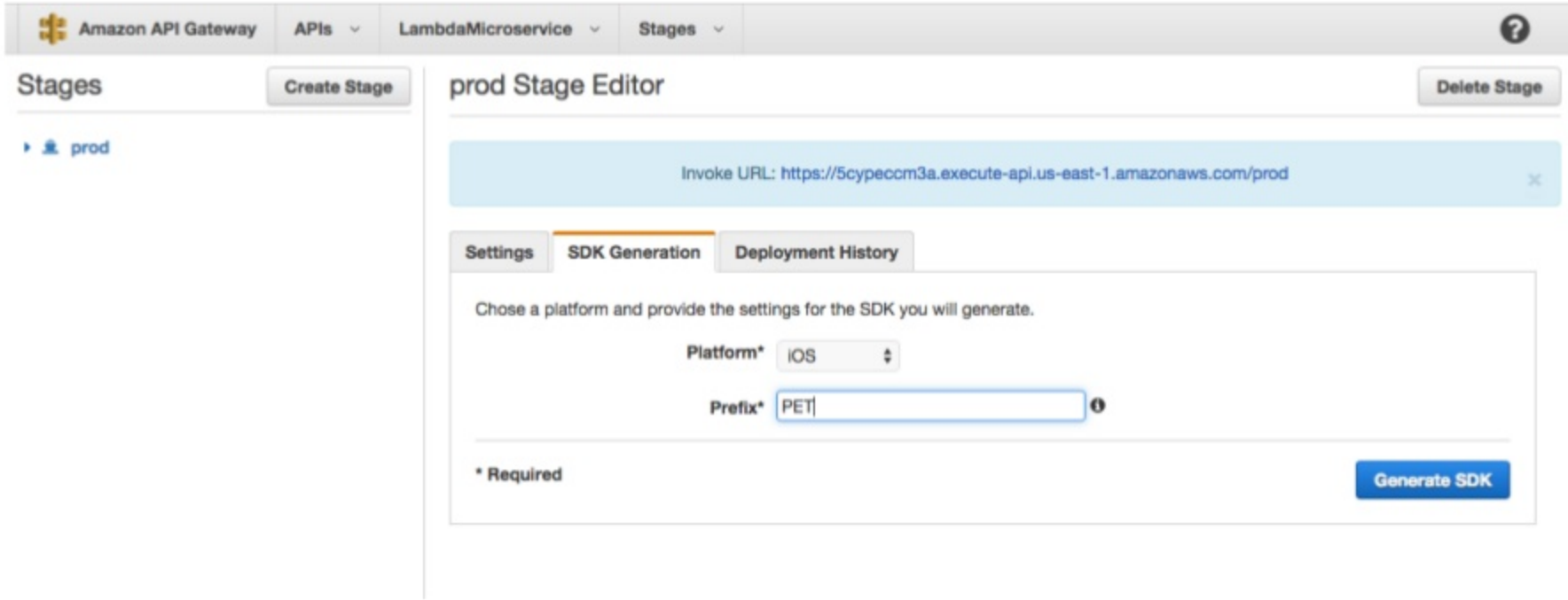
Demo – Authorization

Benefits of using AWS auth & IAM

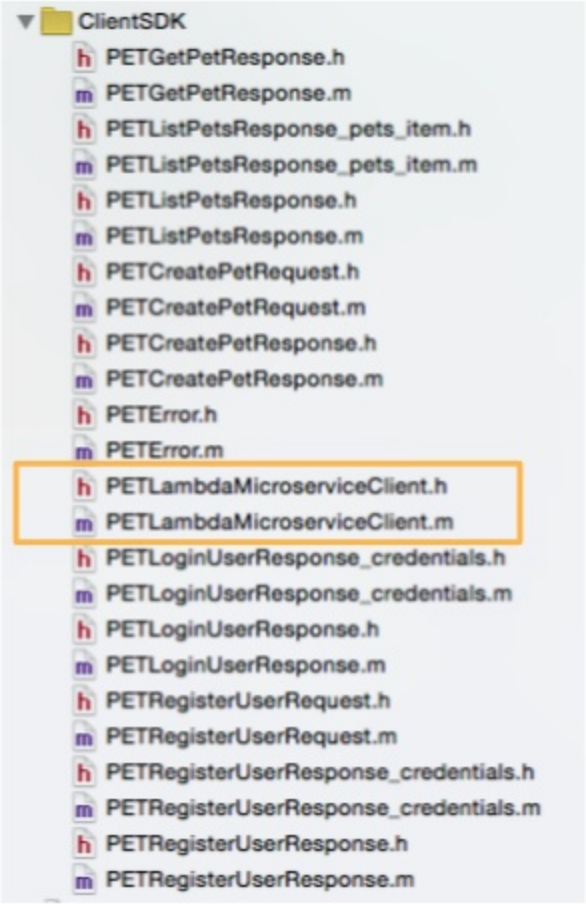
- Separation of concerns – our **authorization strategy is delegated** to a dedicated service
- We have **centralized access management** to a single set of policies
- **Roles and credentials can be disabled** with a single API call

AWS credentials on the client

1-click SDK generation from the console



The client SDK declares all methods




```
PETLambdaMicroserviceClient *client = [PETLambdaMicroserviceClient defaultClient];
[[client petsGet] continueWithBlock:^id(AWSTask *task) {
    PETListPetsResponse *pets = task.result;
    self.objects = [NSMutableArray arrayWithArray:pets.pets];
    dispatch_async(dispatch_get_main_queue(), ^{
        [self.tableView reloadData];
        [hud hide:YES];
    });

    return nil;
}];
```

The AWSCredentialsProvider

We implement the AWSCredentialsProvider interface

```
@interface APIGSessionCredentialsProvider : NSObject <AWSCredentialsProvider>
```

The refresh() method is called whenever the SDK needs new credentials

```
- (AWSTask *)refresh {
    PETLambdaMicroserviceClient *client = [PETLambdaMicroserviceClient clientForKey:
        APIGClientConfigurationKey];
    PETRegisterUserRequest *req = [PETRegisterUserRequest new];
    req.username = _credentials.username;
    req.password = _credentials.password;
    return [[client loginPost:req] continueWithBlock:^id(AWSTask *task) {
        PETLoginUserResponse *resp = task.result;

        PETLoginUserResponse_credentials *credentials = resp.credentials;

        _accessKey = credentials.accessKey;
        _secretKey = credentials.secretKey;
        _sessionKey = credentials.sessionToken;
        _expiration = [NSDate dateWithTimeIntervalSince1970:[credentials.expiration doubleValue]/
            1000];
        return nil;
    }];
}
```

Generated SDK benefits

The generated client SDK knows how to:

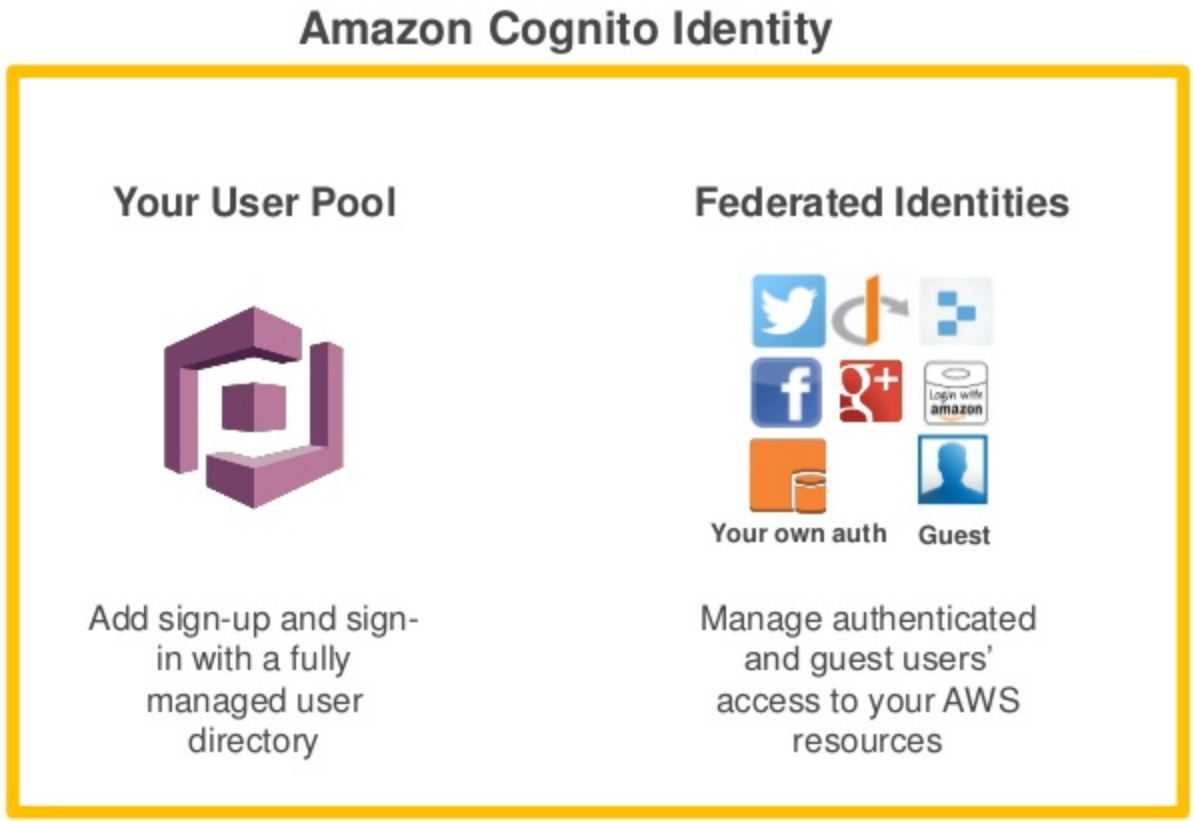
- **Sign API calls** using AWS signature version 4
- **Handle-throttled responses** with exponential back-off
- **Marshal and unmarshal** requests and responses to model objects

Options for authentication and authorization

Options

- Cognito facilitates BYOI
  - Google, FB, etc…
  - Roll your own
  - **Cognito User Pools**
- Authorization options – facilitated by API Gateway
  - AWS IAM
  - Custom Authorizer

Amazon Cognito Identity



What have we learned?

1. AWS Lambda + Amazon API Gateway means no infrastructure to manage – we scale for you

2. Security is important, and complex – make the most of AWS Identity and Access Management by leveraging Cognito

3. Flexibility – API Gateway, Lambda and Cognito give you choices for authentication and authorization

Questions?