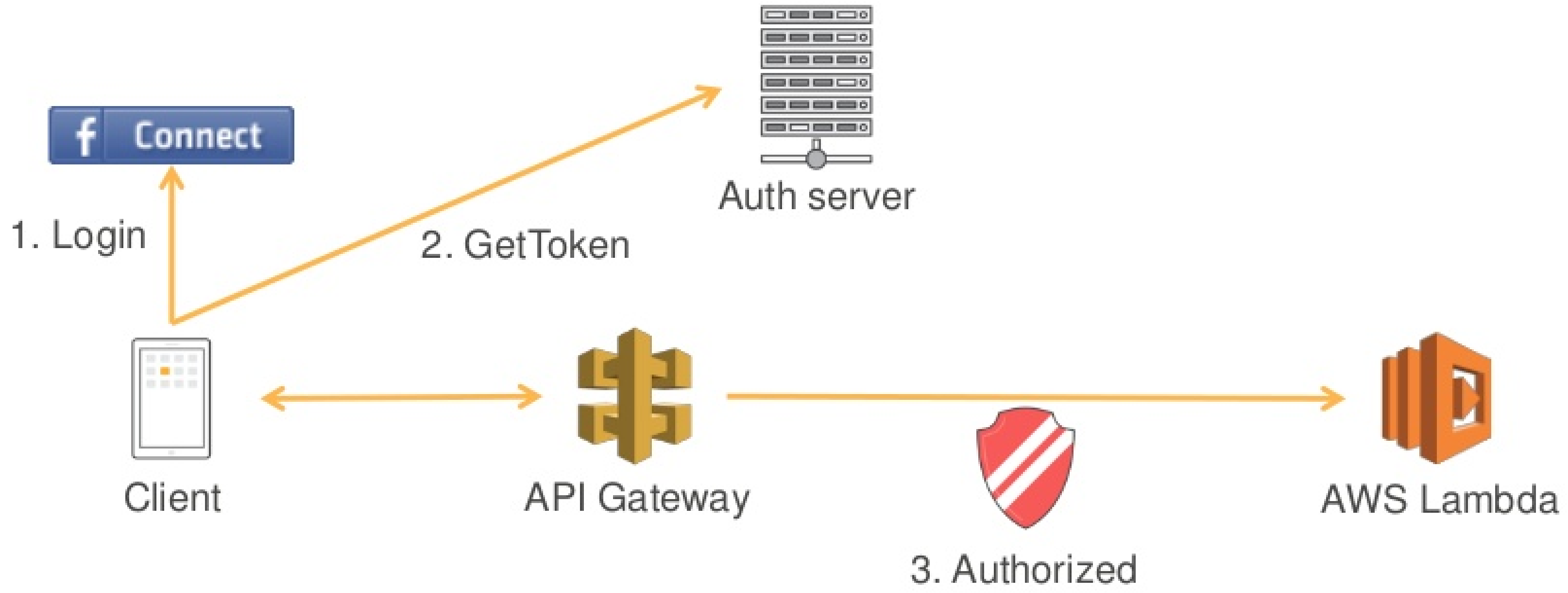
Securing Serverless Workloads with Cognito and API Gateway Part II

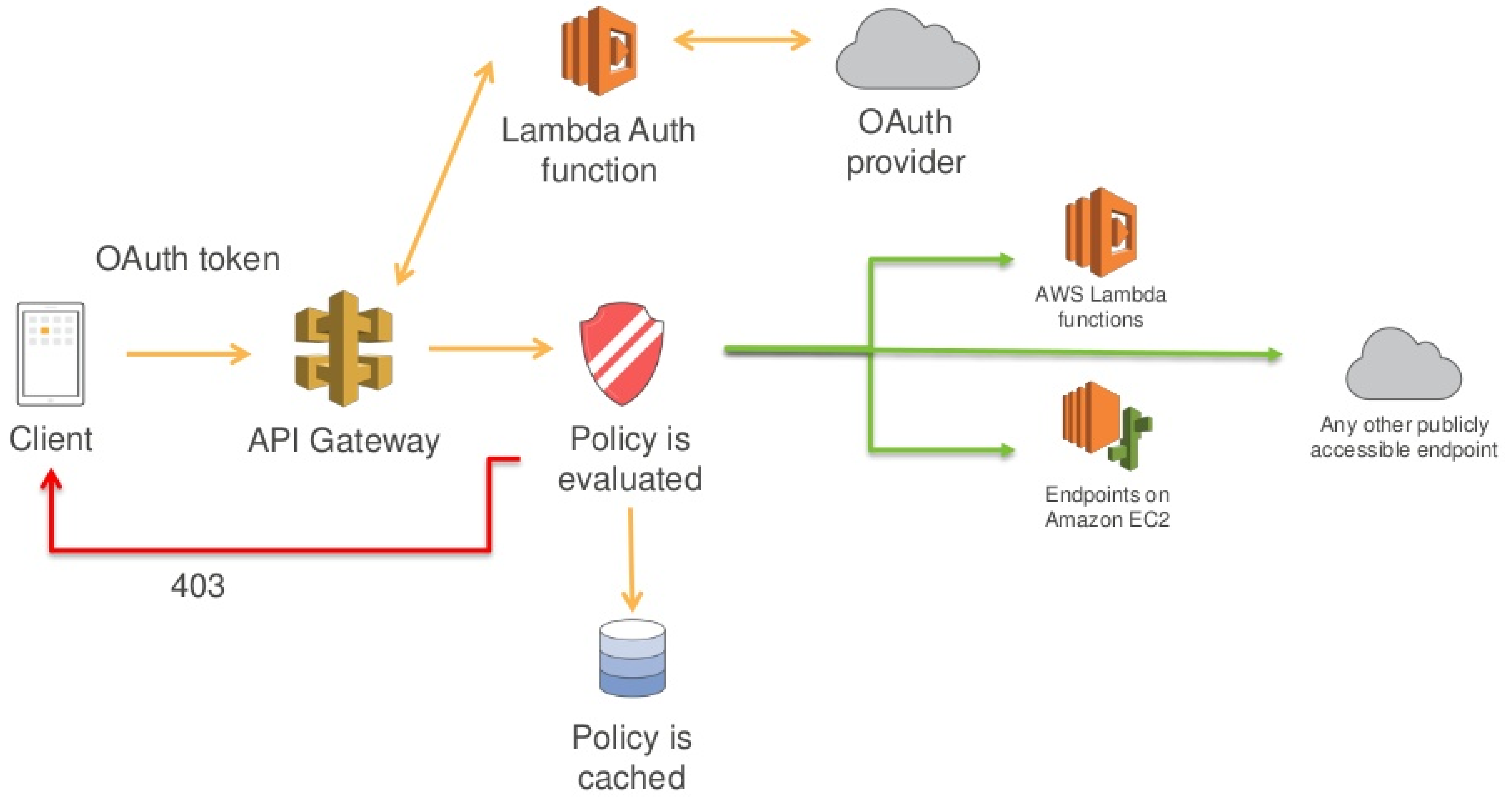
Drew Dennis
Solution Architect
drewdenn@amazon.com

API Gateway Custom auth via Lambda

- Support for bearer token auth (OAuth, SAML)



API Gateway Custom auth via Lambda



Option 2 – The policy

```
{
        "principalId": "meyjames",
        "policyName": "*",
        "policyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                        {
                                "Action": "execute-api:Invoke",
                                "Effect": "Allow",
                                "Resource": "arn:aws:execute-api:us-east-
1:9XXX5213927:z8fxjufsyf/*"
                        }
                ]
        }
}
```

The Lambda function returns a policy that specify the resources that the given token is allowed to invoke in the API

In this case all resources (*) in the API with id **z8fxjufsyf**

Option 2 – The policy

```
{
	"principalId": "meyjames",
	"policyName": "ReadOnly",
	"policyDocument": {
		"Version": "2012-10-17",
		"Statement": [
			{
				"Action": "execute-api:Invoke",
				"Effect": "Allow",
				"Resource": "arn:aws:execute-api:us-east-
1:9XXX5213927:z8fxjufsyf/*/GET/*"
			}
		]
	}
}
```

We could limit the permission to only **GET** methods.

Option 2 – The policy

```
{
        "principalId": "meyjames",
        "policyName": "ReadOnly",
        "policyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                         {
                                "Action": "execute-api:Invoke",
                                "Effect": "Deny",
                                "Resource": "arn:aws:execute-api:us-east-
1:9XXX5213927:z8fxjufsyf/*/*/users/*"
                        },
                        {
                                "Action": "execute-api:Invoke",
                                "Effect": "Allow",
                                "Resource": "arn:aws:execute-api:us-east-
1:9XXX5213927:z8fxjufsyf /*/*/users/myUserId"
                        }
                ]
        }
}
```

We could get clever with JWT tokens

Given the content of the token, only allow the user to access their own data (**user id** is part of the token)

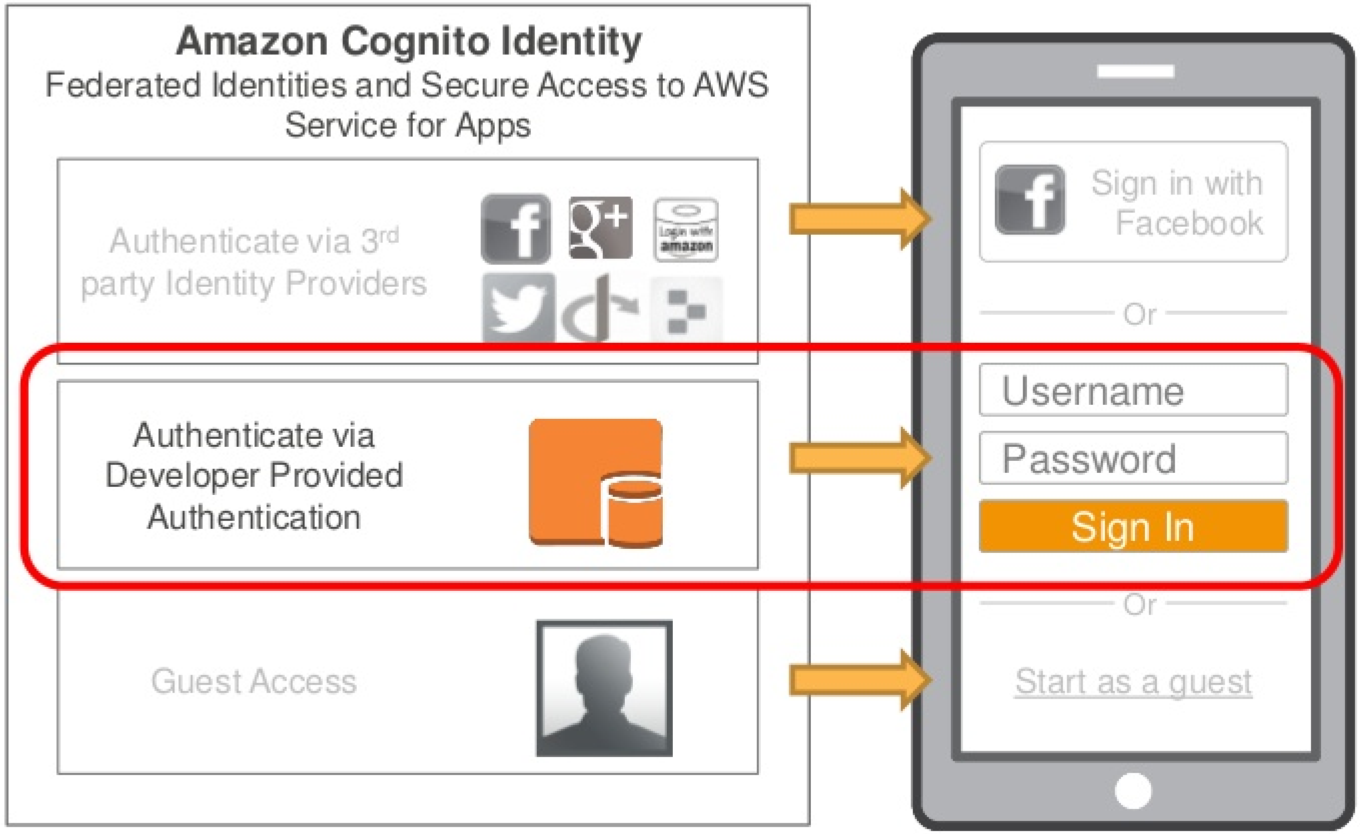
Option 2 – The policy

Generating a policy is not as hard as it looks…

```
var testPolicy = new AuthPolicy("meyjames", "XXXXXXXXXXXX", apiOptions);

testPolicy.denyAllMethods();

testPolicy.allowMethod(AuthPolicy.HttpVerb.GET, "/users/username");
testPolicy.allowMethod(AuthPolicy.HttpVerb.POST, "/users/username");

testPolicy.allowMethodWithConditions(AuthPolicy.HttpVerb.GET, "/pets/123123", {
        "StringEquals": {
          "s3:x-amz-acl": [
            "public-read"
          ]
        }
});

context.succeed(testPolicy.getPolicy());
```

DEMO

Developers Don't Want to Build Their Own User Directory


Amazon Cognito Identity
Federated Identities and Secure Access to AWS Service for Apps
Authenticate via 3rd party Identity Providers
Authenticate via Developer Provided Authentication
Guest Access
Sign in with Facebook
Or
Username
Password
Sign In
Or
Start as a guest

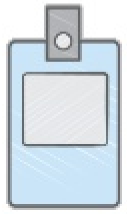
Developers do not want to take on the undifferentiated heavy lifting to:

- Build and maintain a directory
- Get security right
- Support workflows like forgot password
- Scale as their user base grows

A Fully Managed User Directory in Cognito

1

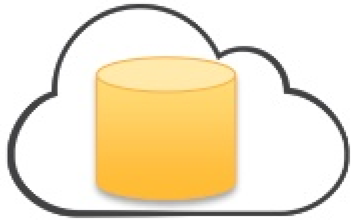
Easy User Management



Add sign-up and sign-in easily to your mobile and web apps

2

Managed User Directory



Launch a simple, secure, low-cost, and fully managed service to create and maintain a user directory that scales to 100s of millions of users

3

Enhanced Security Features

Verify phone numbers and email addresses and offer multi-factor authentication

Comprehensive User Scenarios

User sign-up and sign-in	Users sign-up using email, phone number or user name and password. Users can then sign-in.
Email or phone number Verification	Users verify their email address or phone number prior to activating an account
Forgot Password	Users can change their password if they forget it
User Profile	Retrieve and update user profiles, including custom attributes
SMS-based MFA	If enabled, users complete Multi-Factor Authentication (MFA) with a confirmation code via SMS as part of sign-in and forgot password flows

Comprehensive Administrator Scenarios

Create and manage User Pools	Create, configure and delete multiple User Pools in their AWS account
Manage users in a User Pool	List, search and perform actions on specific user(s) in the User Pool
Select Email and Phone Verification	Configure verifications of users' email addresses and phone numbers (via SMS)
Customize with Lambda Triggers	Create functions in AWS Lambda to customize workflows
Setup Password Policies	Control password requirements like minimum length, uppercase, and inclusion of special characters
Define Attributes	Select required attributes and Define custom user attributes

Secure Sign-in Made Easy

Token-based Authentication	Uses tokens based on OpenID Connect (OIDC) and OAuth 2.0 standards
Secure Remote Password Protocol	Uses Secure Remote Password (SRP) for secure password handling end to end
SMS-based Multi-factor Authentication	Enables your end users to user the text messaging functionality of a mobile phone as an extra layer of security

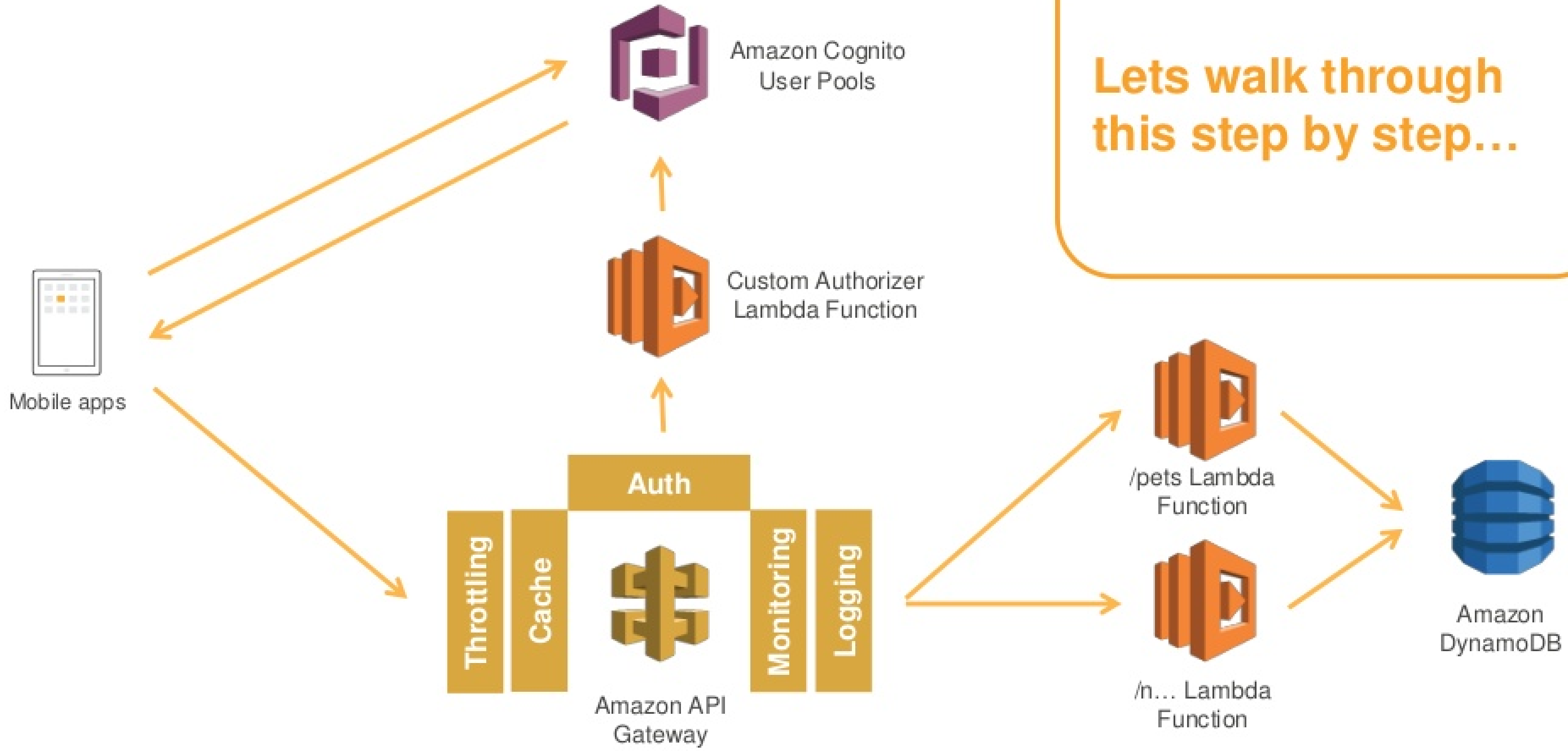
Cognito User Pool Tokens

- User Token
  - JWT
  - OpenID Connect
  - One Hour
- Access Token
  - JWT
  - OAuth2
  - One Hour
- Refresh Token
  - Long-lived
  - Sent to Cognito Identity when Token has expired

Customization using Lambda hooks

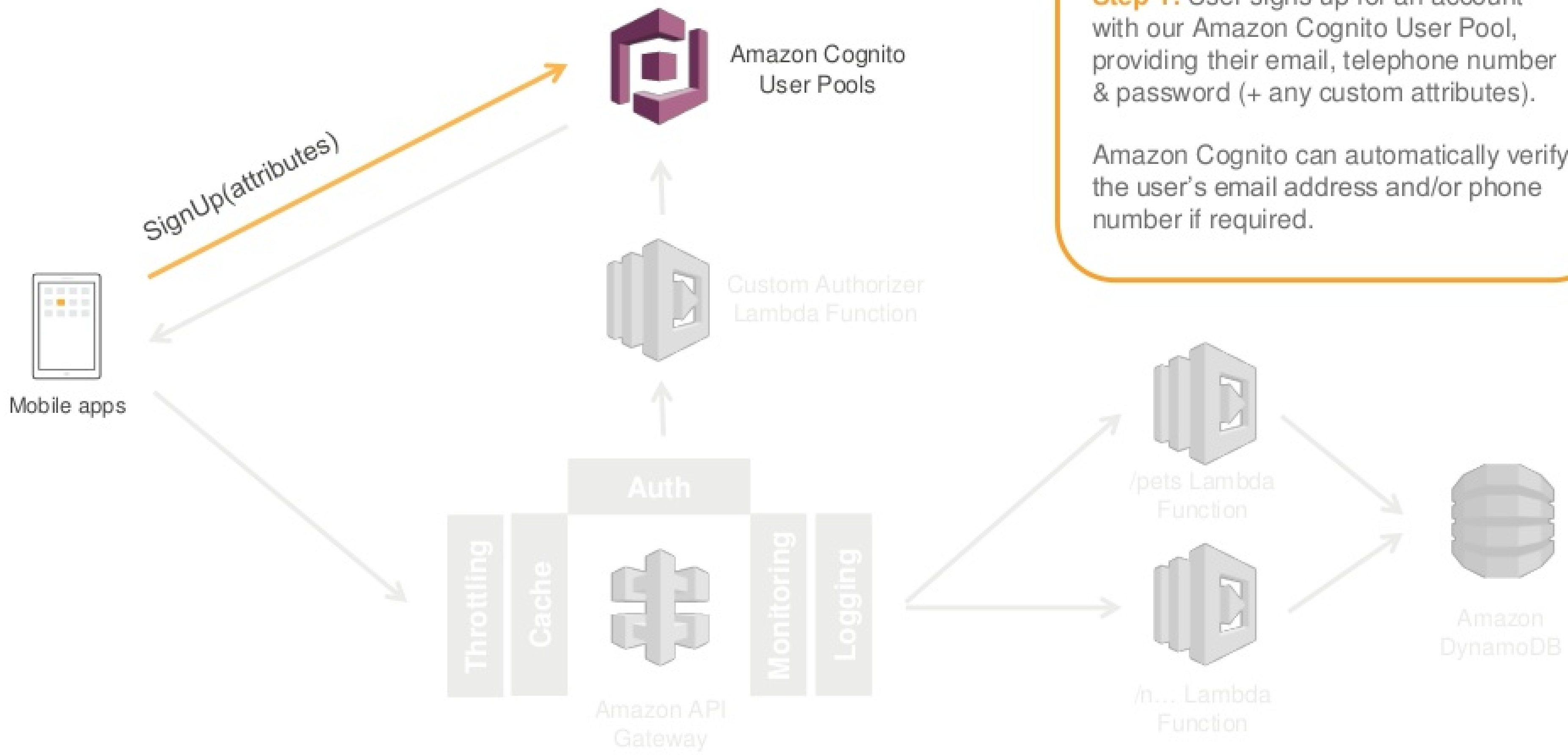
Lambda Hook	Example Scenarios
Pre user sign-up	Custom validation to accept or deny the sign-up request
Custom message	Advanced customization and localization of verification messages
Pre user sign-in	Custom validation to accept or deny the sign-in request
Post user sign-in	Event logging for custom analytics
Post user confirmation	Custom welcome messages or event logging for custom analytics

Authentication Flow



Lets walk through this step by step…

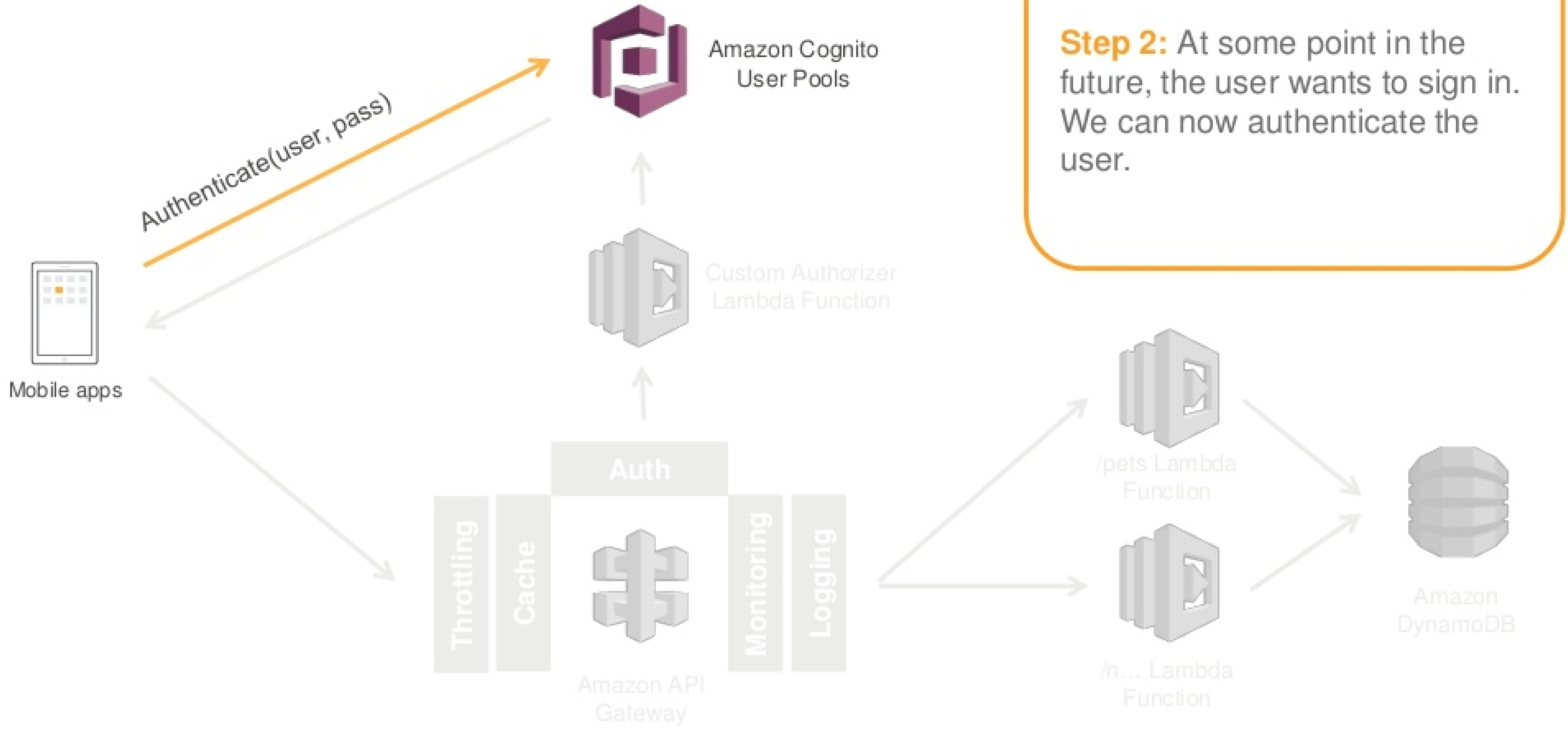
Authentication Flow



Step 1: User signs up for an account with our Amazon Cognito User Pool, providing their email, telephone number & password (+ any custom attributes).

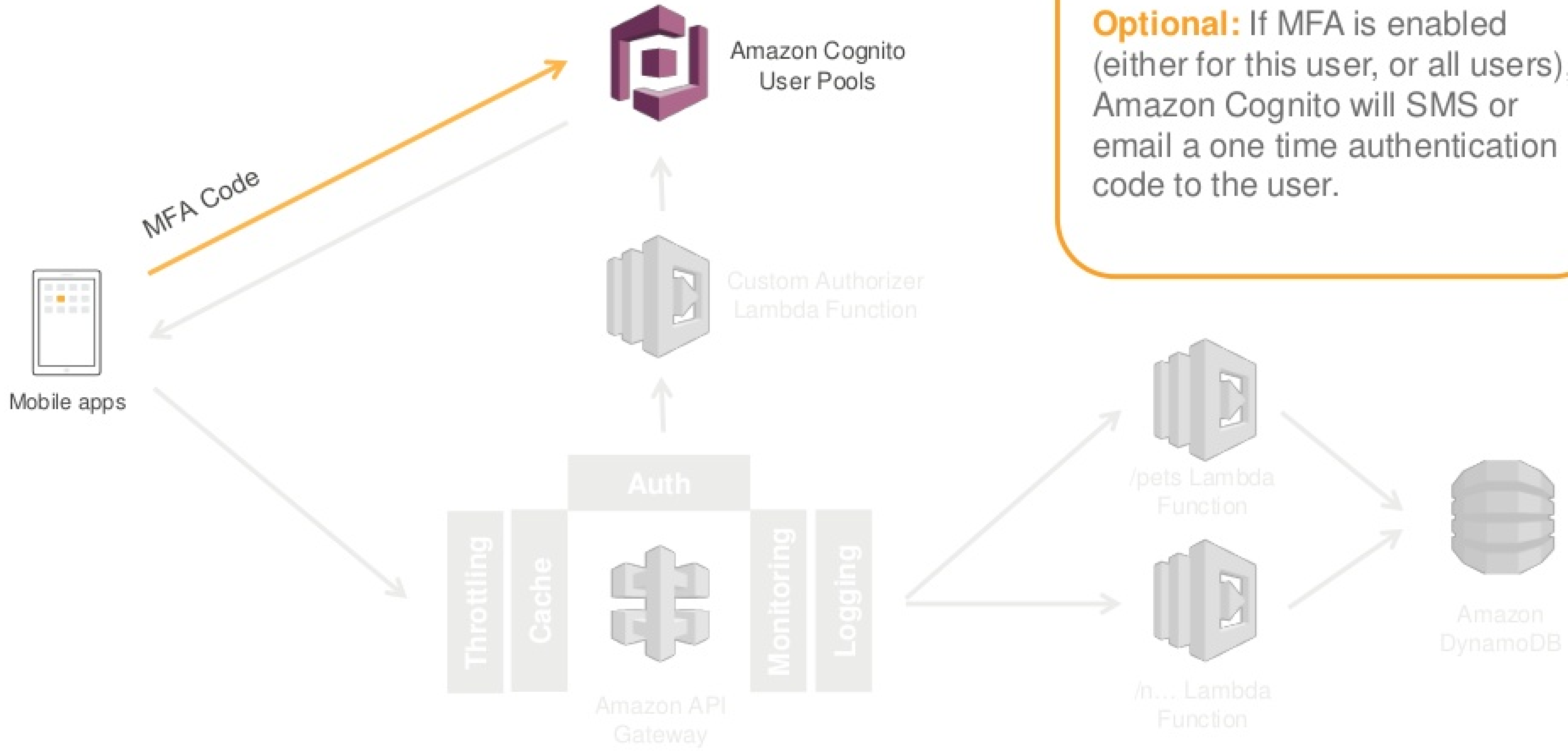
Amazon Cognito can automatically verify the user's email address and/or phone number if required.

Authentication Flow



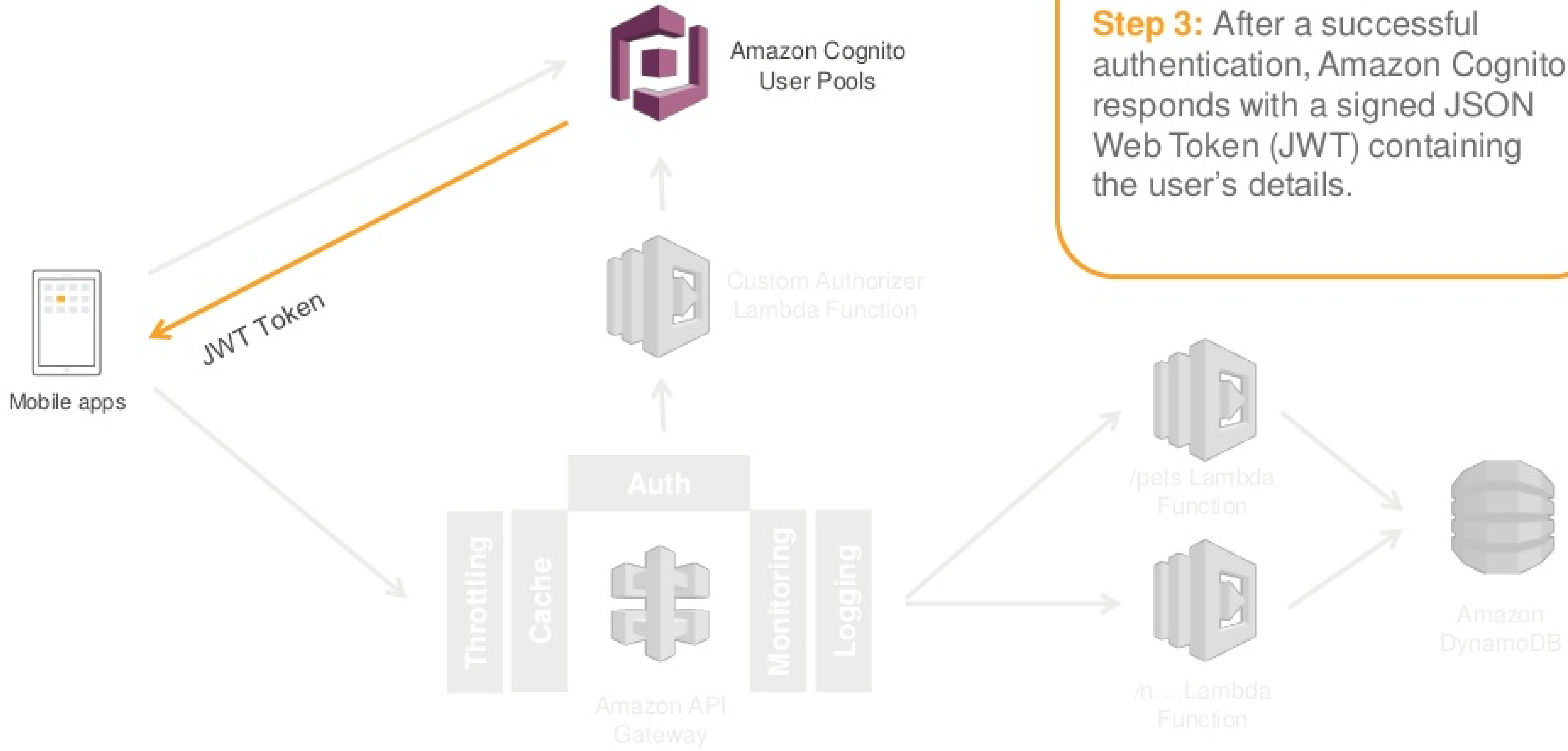
Step 2: At some point in the future, the user wants to sign in. We can now authenticate the user.

Authentication Flow



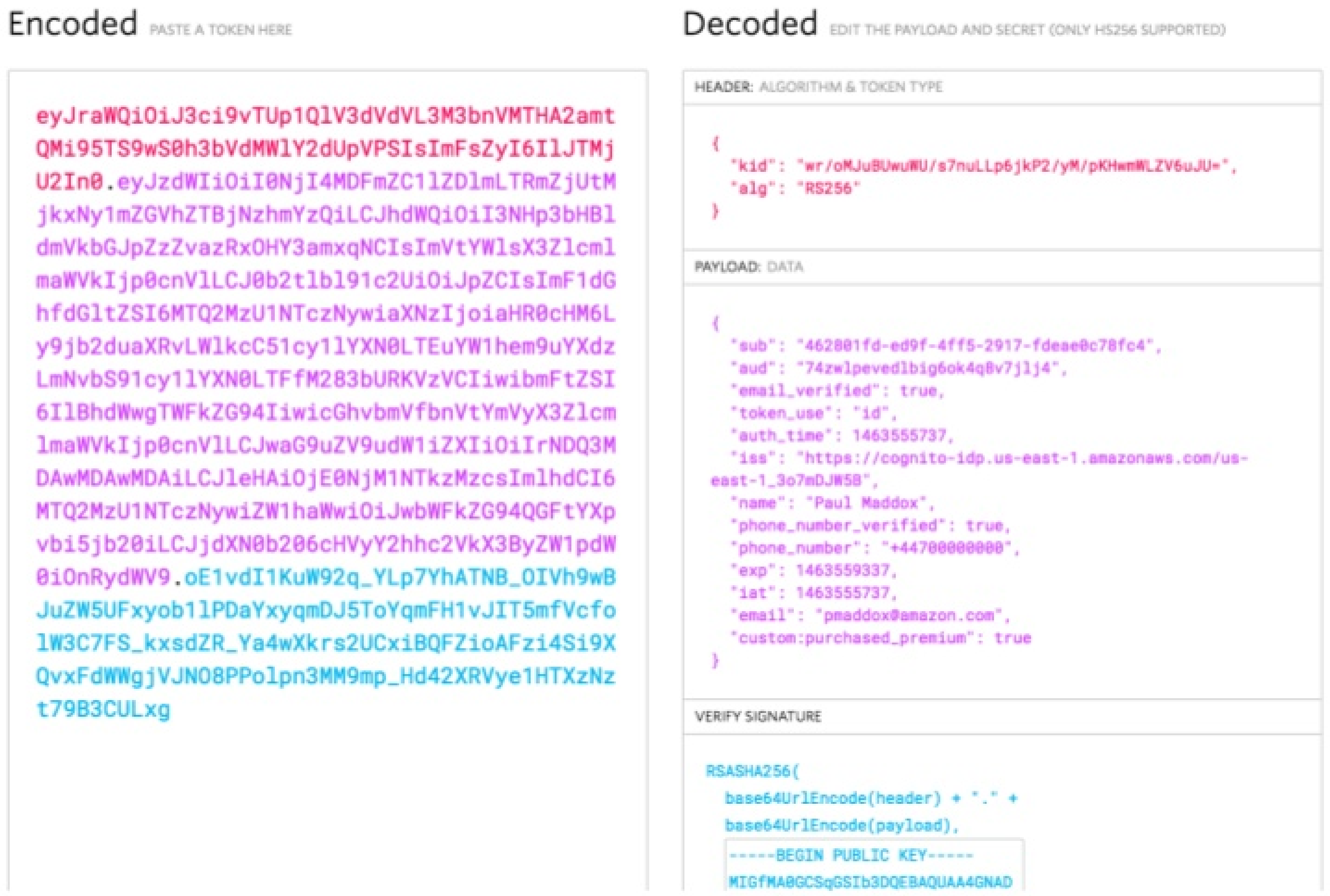
Optional: If MFA is enabled (either for this user, or all users), Amazon Cognito will SMS or email a one time authentication code to the user.

Authentication Flow



Step 3: After a successful authentication, Amazon Cognito responds with a signed JSON Web Token (JWT) containing the user's details.

Wait… What is a JSON Web Token (JWT)?



Encoded PASTE A TOKEN HERE

```
eyJraWQiOiJ3ci9vTUp1QlV3dVdVL3M3bnVMTHA2amt
QMi95TS9wS0h3bVdMWlY2dUpVPSIsImFsZyI6IlJTMj
U2In0.eyJzdWIiOiI0NjI4MDFmZC1lZDlmLTRmZjUtM
jkxNy1mZGVhZTBjNzhmYzQiLCJhdWQiOiI3NHp3bHBl
dmVkbGJpZzZvazRxOHY3amxqNCIsImVtYWlsX3Zlcml
maWVkIjp0cnVlLCJ0b2tlbl91c2UiOiJpZCIsImF1dG
hfdGltZSI6MTQ2MzU1NTczNywiaXNzIjoiaHR0cHM6L
y9jb2duaXRvLWlkcC51cy1lYXN0LTEuYW1hem9uYXdz
LmNvbS91cy1lYXN0LTFfM283bURKVzVCIiwibmFtZSI
6IlBhdWwgTWFkZG94IiwicGhvbmVfbnVtYmVyX3Zlcm
lmaWVkIjp0cnVlLCJwaG9uZV9udW1iZXIiOiIrNDQ3M
DAwMDAwMDAiLCJleHAiOjE0NjM1NTkzMzcsImlhdCI6
MTQ2MzU1NTczNywiZW1haWwiOiJwbWFkZG94QGFtYXp
vbi5jb20iLCJjdXN0b206cHVyY2hhc2VkX3ByZW1pdW
0iOnRydWV9.oE1vdI1KuW92q_YLp7YhATNB_OIVh9wB
JuZW5UFxyob1lPDaYxyqmDJ5ToYqmFH1vJIT5mfVcfo
lW3C7FS_kxsdZR_Ya4wXkrs2UCxiBQFZioAFzi4Si9X
QvxFdWWgjVJNO8PPolpn3MM9mp_Hd42XRVye1HTXzNz
t79B3CULxg
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "wr/oMJuBUwuWU/s7nuLLp6jkP2/yM/pKHwmWLZV6uJU=",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "462801fd-ed9f-4ff5-2917-fdeae0c78fc4",
  "aud": "74zwlpevedlbig6ok4q8v7jlj4",
  "email_verified": true,
  "token_use": "id",
  "auth_time": 1463555737,
  "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-
east-1_3o7mDJW5B",
  "name": "Paul Maddox",
  "phone_number_verified": true,
  "phone_number": "+44700000000",
  "exp": 1463559337,
  "iat": 1463555737,
  "email": "pmaddox@amazon.com",
  "custom:purchased_premium": true
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----
  MIGfMA0GCSqGSIb3DQEBAQUAA4GNAD
```

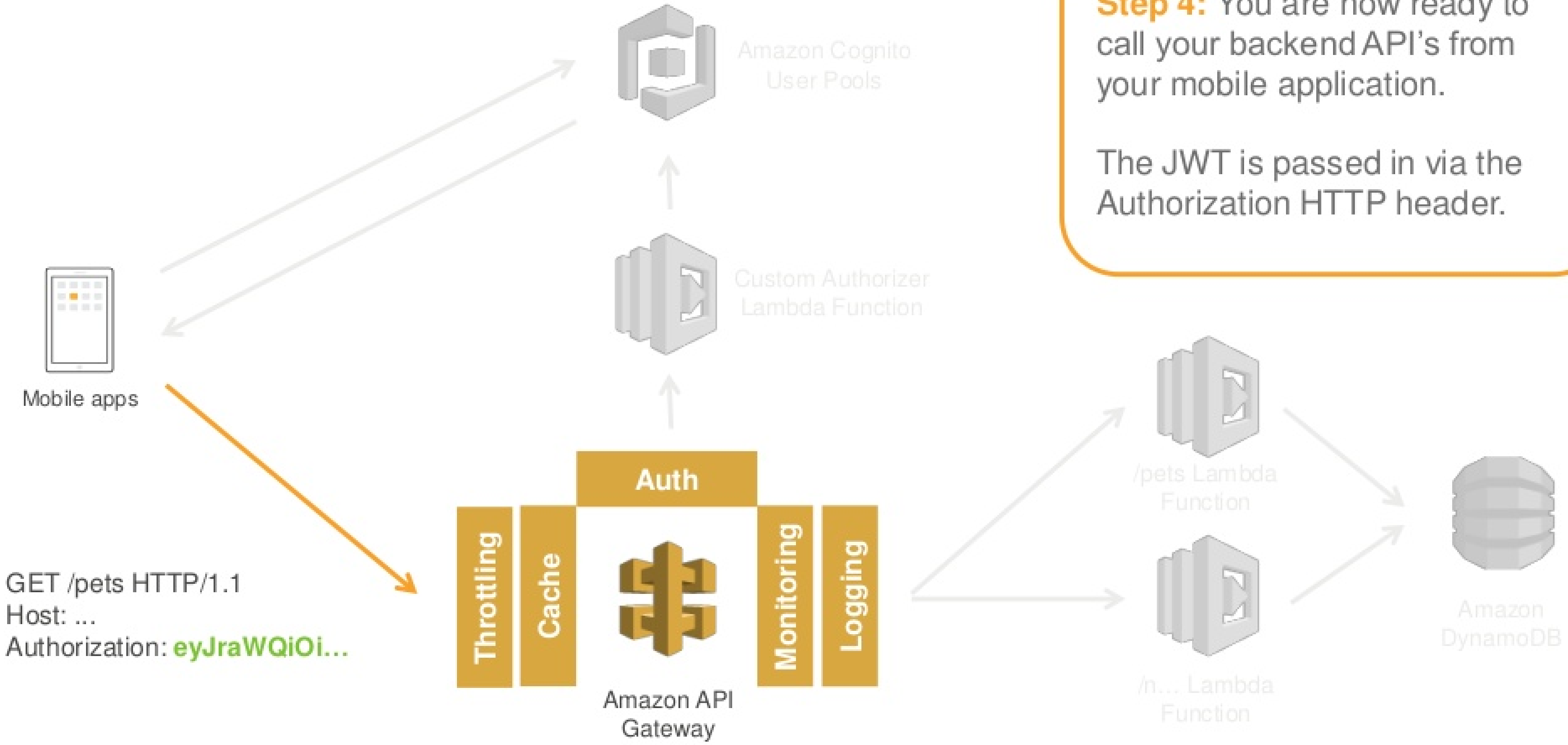
Cryptographically verifiable claims

* https://jwt.io

Authentication Flow

Step 4: You are now ready to call your backend API's from your mobile application.

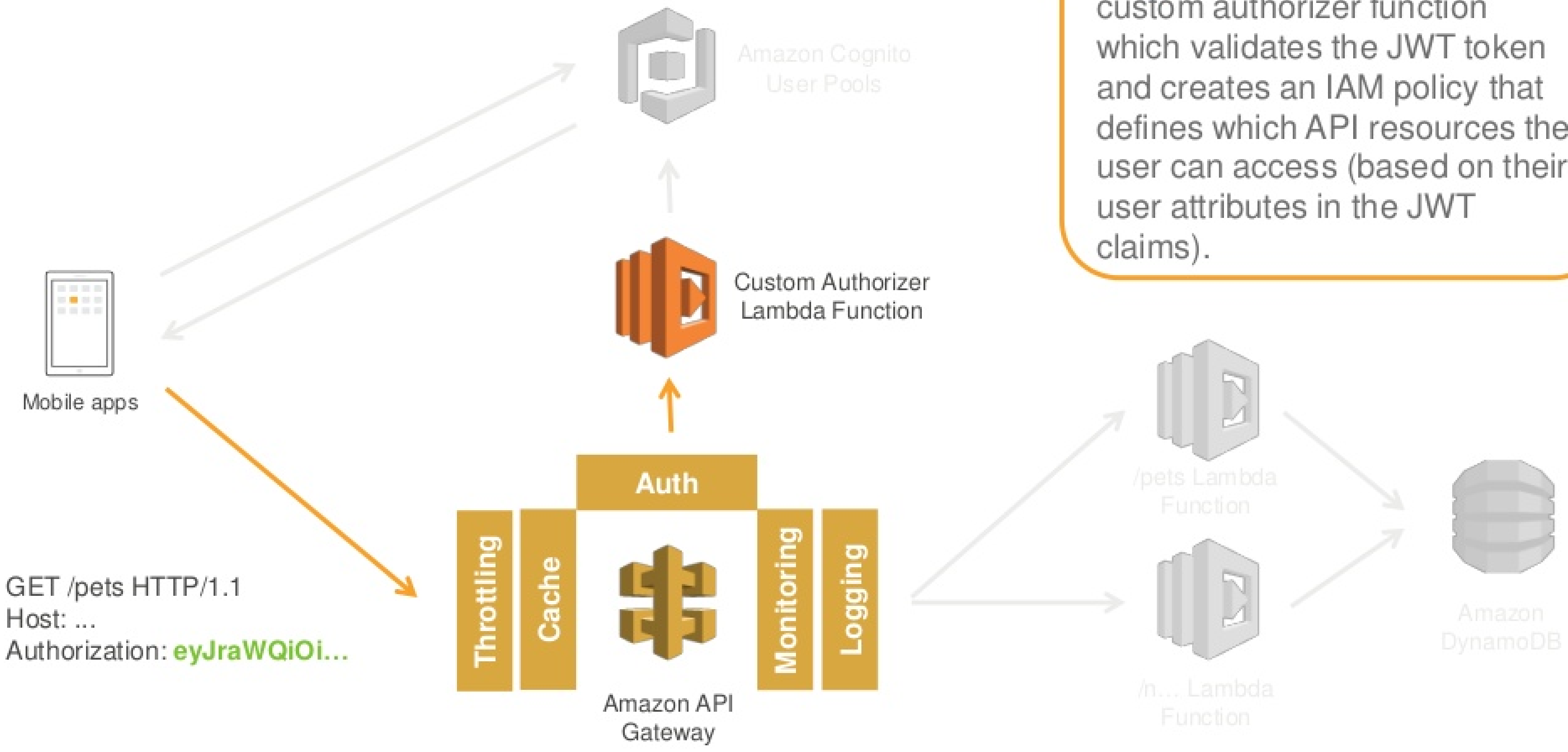
The JWT is passed in via the Authorization HTTP header.



```
GET /pets HTTP/1.1
Host: ...
Authorization: eyJraWQiOi...
```

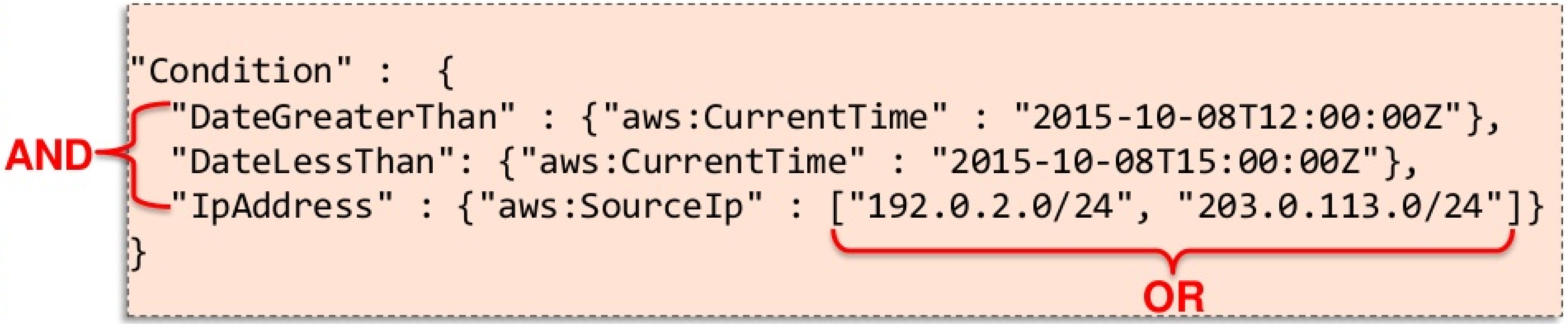
Authentication Flow

Step 5: API Gateway calls your custom authorizer function which validates the JWT token and creates an IAM policy that defines which API resources the user can access (based on their user attributes in the JWT claims).



GET /pets HTTP/1.1
Host: ...
Authorization: eyJraWQiOi…

Condition example



```
"Condition" :  {
  "DateGreaterThan" : {"aws:CurrentTime" : "2015-10-08T12:00:00Z"},
  "DateLessThan": {"aws:CurrentTime" : "2015-10-08T15:00:00Z"},
  "IpAddress" : {"aws:SourceIp" : ["192.0.2.0/24", "203.0.113.0/24"]}
}
```

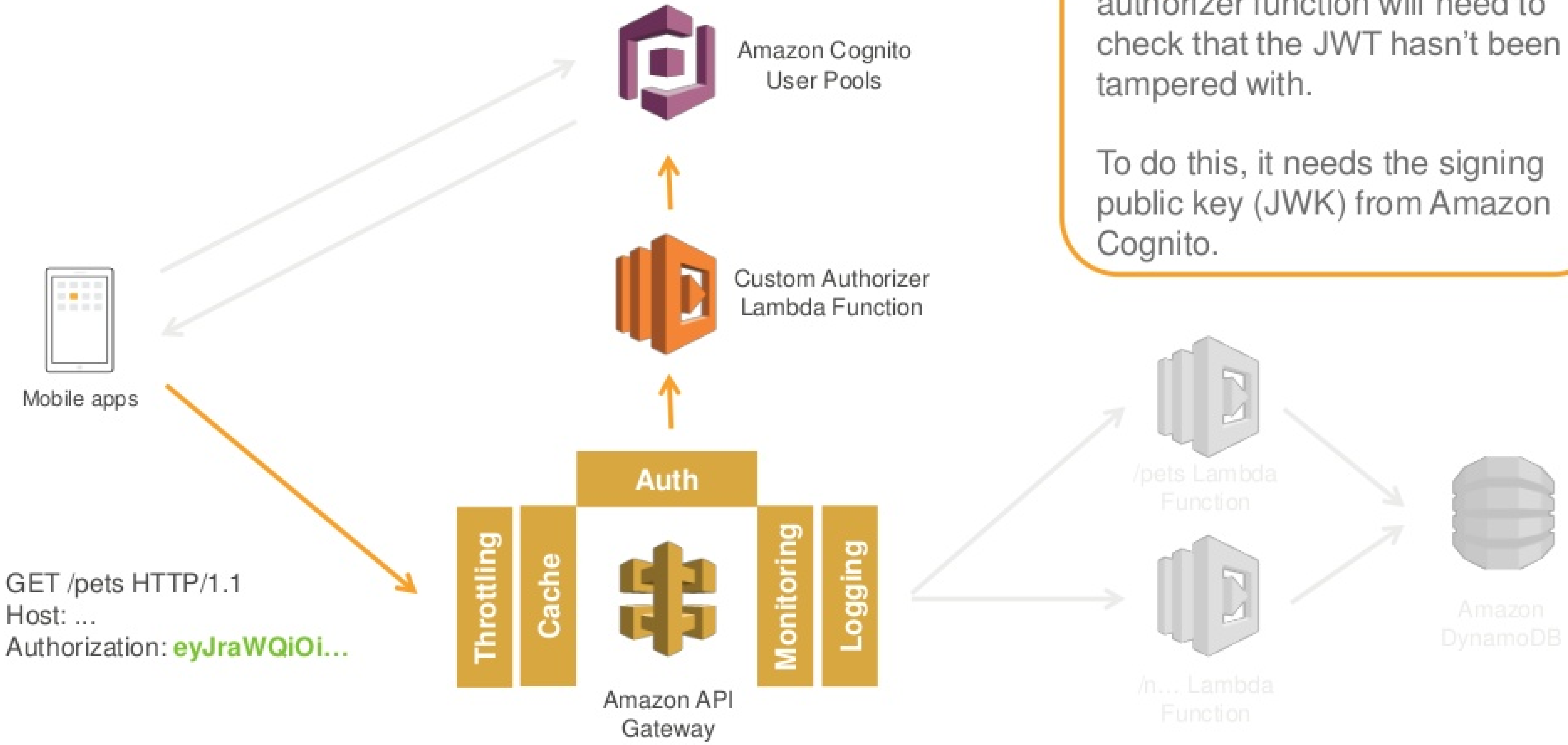
- Allows a user to access a resource under the following conditions:
  - The time is after 12:00 P.M. on 10/8/2015 AND
  - The time is before 3:00 P.M. on 10/8/2015 AND
  - The request comes from an IP address in the 192.0.2.0 /24 OR 203.0.113.0 /24 range
  - MFA Present, MFA Age, Secure Transport

All of these conditions <u>must be met</u> in order for the statement to evaluate to TRUE.

Authentication Flow

Step 6: Additionally, the custom authorizer function will need to check that the JWT hasn't been tampered with.

To do this, it needs the signing public key (JWK) from Amazon Cognito.

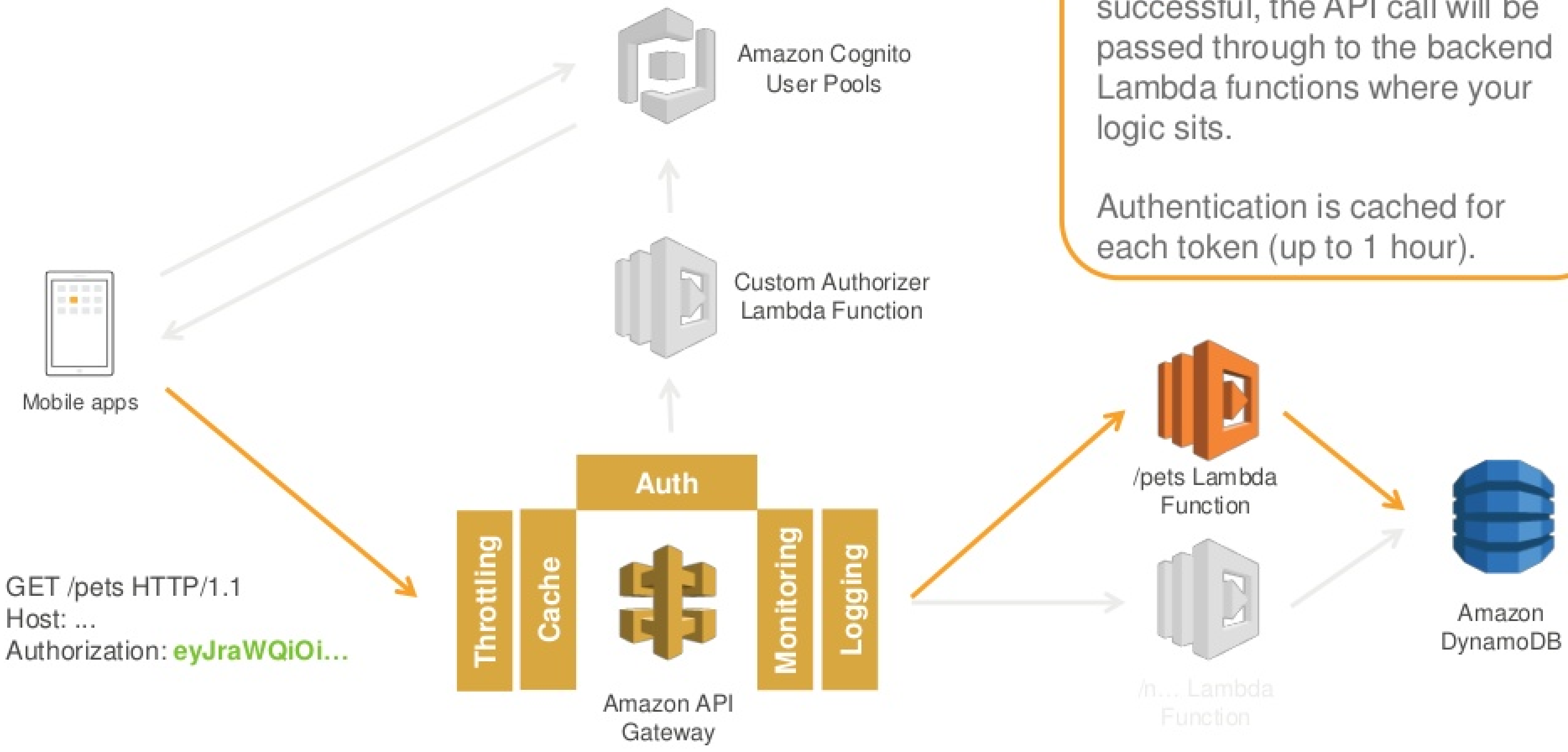


```
GET /pets HTTP/1.1
Host: ...
Authorization: eyJraWQiOi...
```

Authentication Flow

Step 7: If authentication was successful, the API call will be passed through to the backend Lambda functions where your logic sits.

Authentication is cached for each token (up to 1 hour).



```
GET /pets HTTP/1.1
Host: ...
Authorization: eyJraWQiOi...
```

DEMO