Build and Manage your APIs with Amazon API Gateway

Agenda

Why we built Amazon API Gateway

What is Amazon API Gateway?

Amazon API Gateway Features & Functionality

Q&A

At AWS, We run a lot of APIs

…Over time, we have learned a few lessons

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

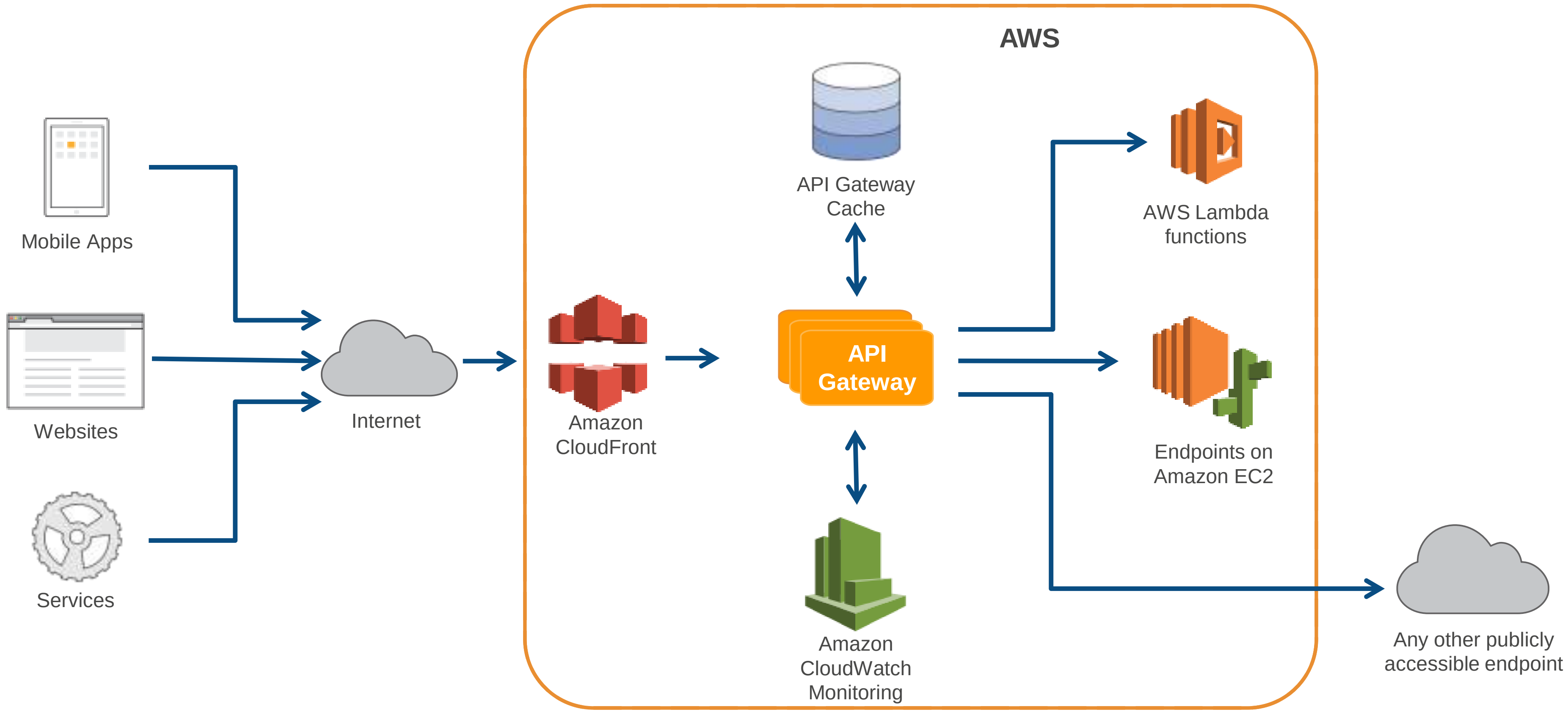
- ✓ Host multiple versions and stages of your APIs
- ✓ Create and distribute API Keys to developers
- ✓ Leverage AWS Sigv4 to authorize access to APIs
- ✓ Throttle and monitor requests to protect your backend
- ✓ Utilizes AWS Lambda

Introducing Amazon API Gateway

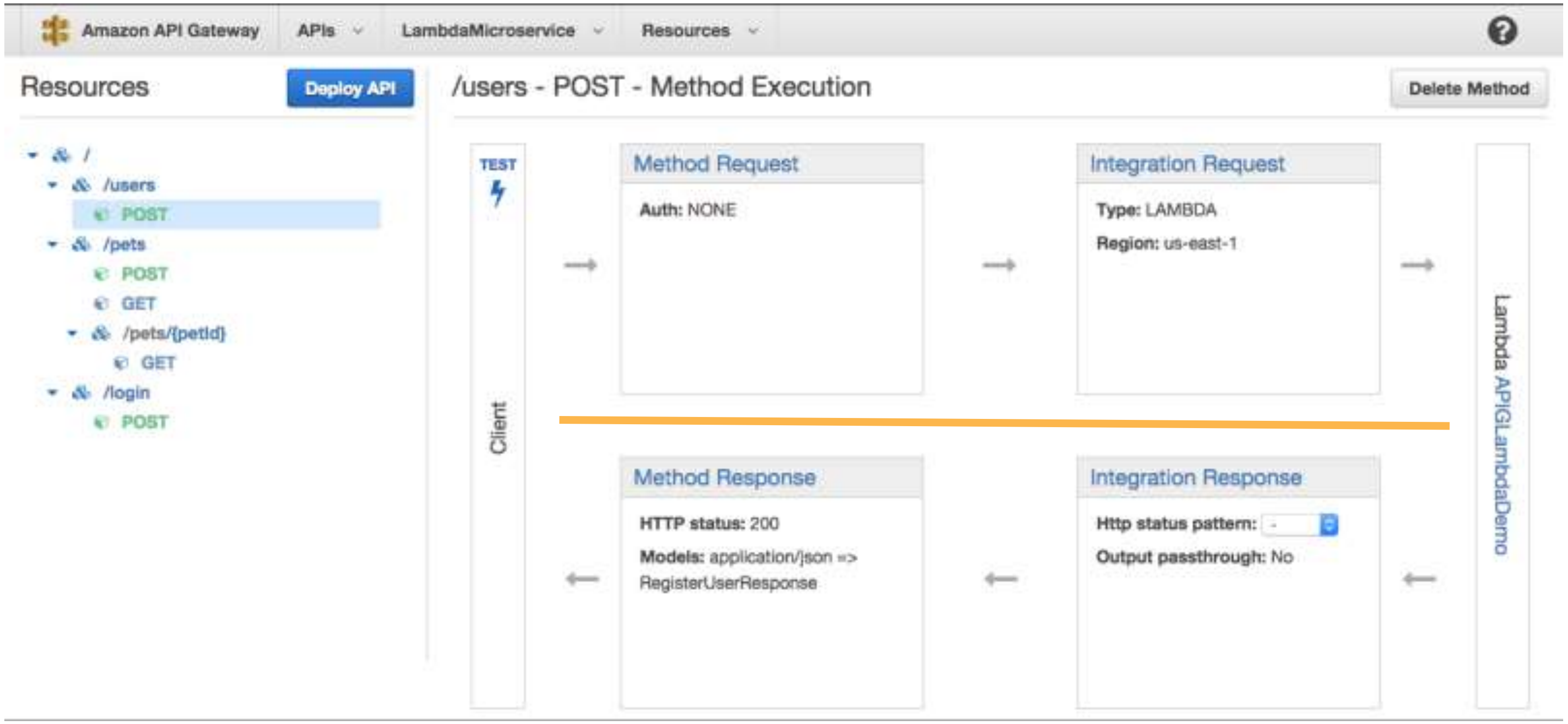
- ✓ Managed cache to store API responses
- ✓ Reduced latency and DDoS protection through CloudFront
- ✓ SDK Generation for iOS, Android and JavaScript
- ✓ Swagger support
- ✓ Request / Response data transformation and API mocking

How Does Amazon API Gateway Work?

An API Call Flow



Methods and Integrations



/users - POST - Method Execution

Method Request

Auth: NONE

Integration Request

Type: LAMBDA

Region: us-east-1

Method Response

HTTP status: 200

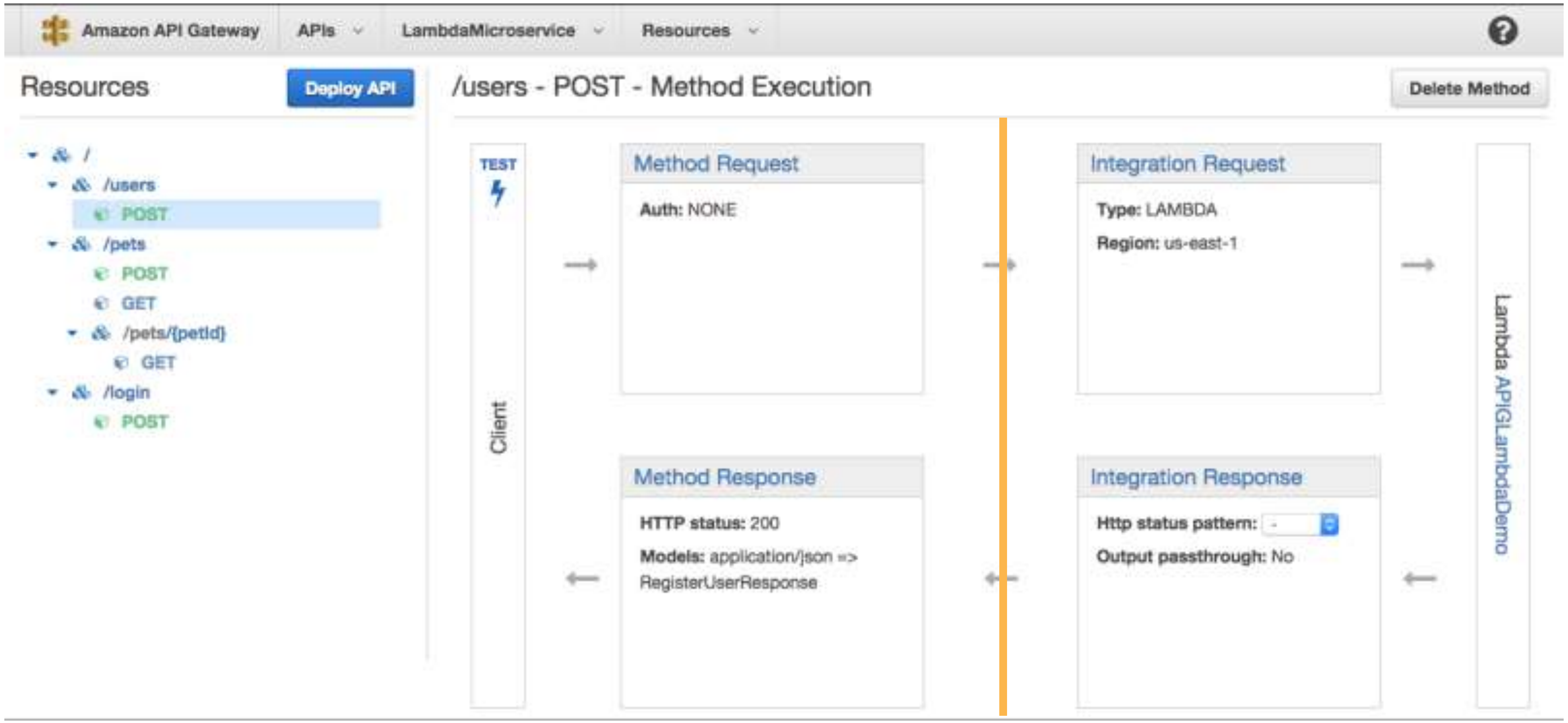
Models: application/json => RegisterUserResponse

Integration Response

Http status pattern: -

Output passthrough: No

Methods and Integrations



Auth: NONE

Type: LAMBDA

Region: us-east-1

HTTP status: 200

Models: application/json => RegisterUserResponse

Http status pattern: -

Output passthrough: No

Build, Deploy, Clone & Rollback

Build APIs with their resources, methods, and settings

Deploy APIs to a Stage

- Users can create as many Stages as they want, each with its own Throttling, Caching, Metering, and Logging configuration

Clone an existing API to create a new version

- Users can continue working on multiple versions of their APIs

Rollback to previous deployments

- We keep a history of customers' deployments so they can revert to a previous deployment

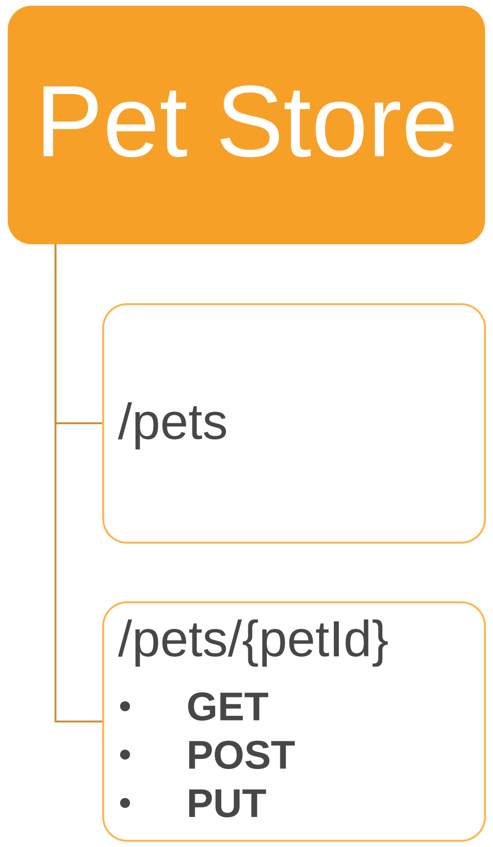
API Configuration

You can create APIs

Define resources within an API

Define methods for a resource

- Methods are Resource + HTTP verb



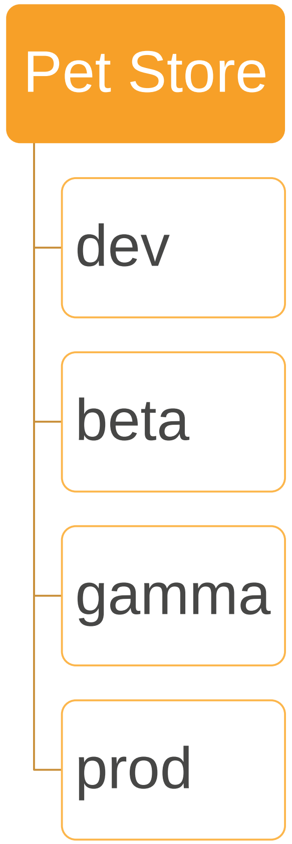
API Deployments

API Configuration can be deployed to a stage

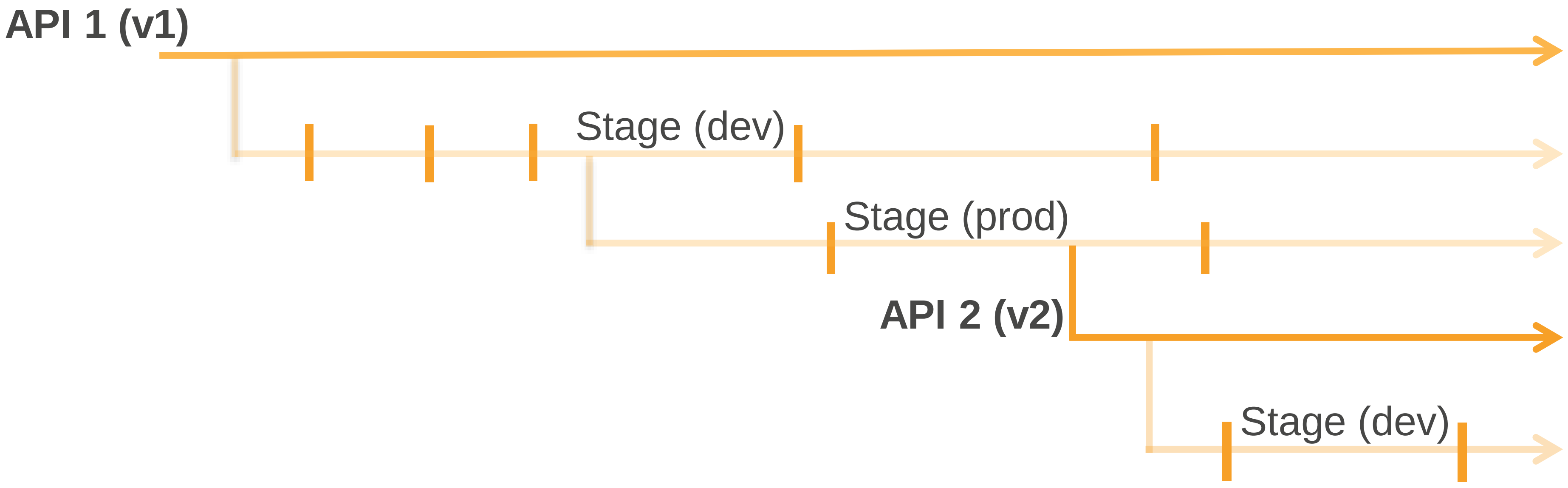
Stages are different environments

For example:

- Dev (e.g. awsapigateway.com/dev)
- Beta (e.g. awsapigateway.com/beta)
- Prod (e.g. awsapigateway.com/prod)
- As many stages as you need



Manage Multiple Versions and Stages of your APIs



Custom Domain Names

You can configure custom domain names with subdomains and base paths

Pointing to an API you have access to all Stages

- Beta (e.g. yourapi.com/beta)
- Prod (e.g. yourapi.com/prod)

Pointing directly to your “prod” Stage

- Prod (e.g. yourapi.com/)

Metering and Authorization

API Keys to Meter Developer Usage

Create API Keys

Set access permissions at the API/Stage level

Meter usage of the API Keys through CloudWatch Logs

API Keys

- API Keys should be used purely to meter app/developer usage
- API Keys should be used alongside a stronger authorization mechanism

Leverage AWS Sigv4, or Use a Custom Header

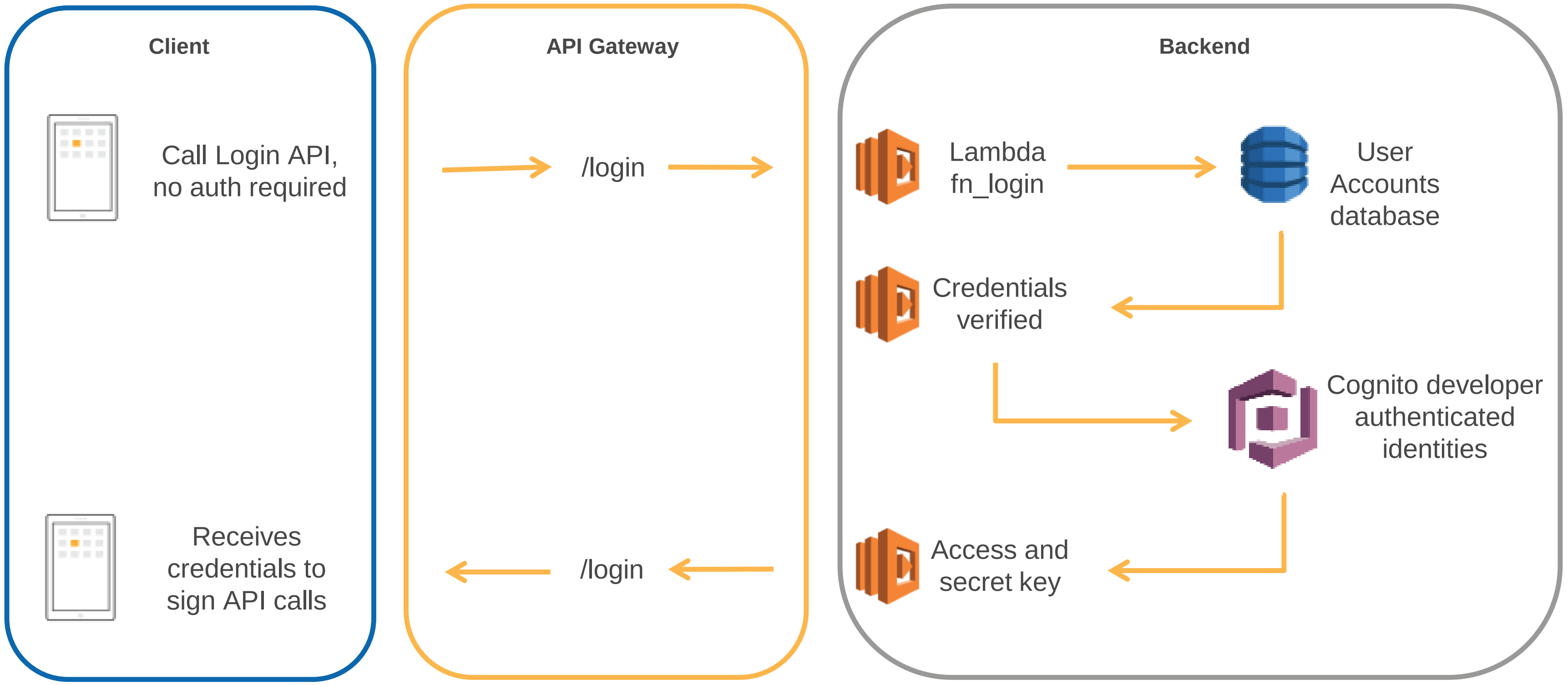
You can leverage AWS Sigv4 to sign and authorize API calls

- Amazon Cognito and AWS Security Token Service (STS) simplify the generation of temporary credentials for your app

You can support OAuth or other authorization mechanisms through custom headers

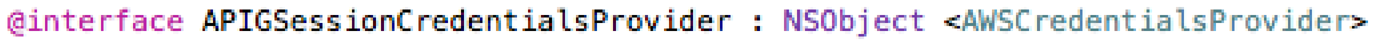
- Simply configure your API methods to forward the custom headers to you backend

Using Sigv4 to authenticate calls to your API



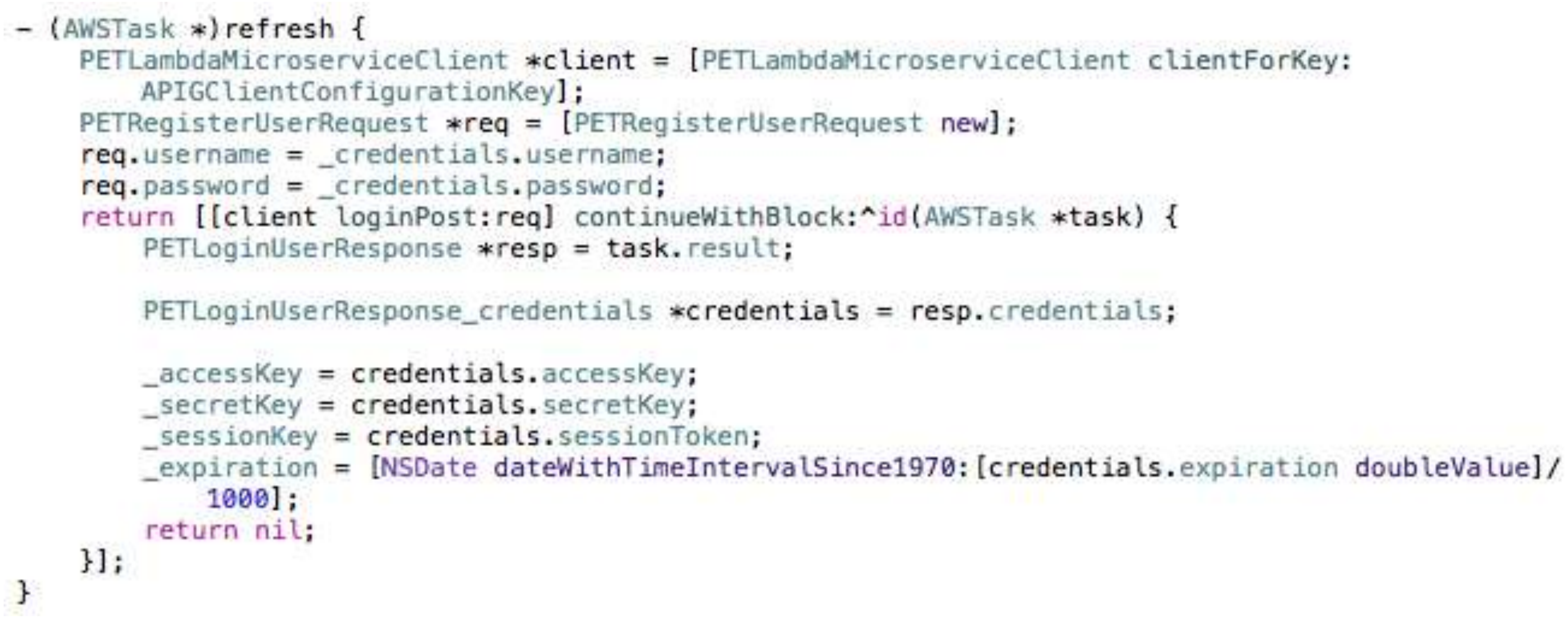
The AWSCredentialsProvider

We implement the AWSCredentialsProvider interface



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

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

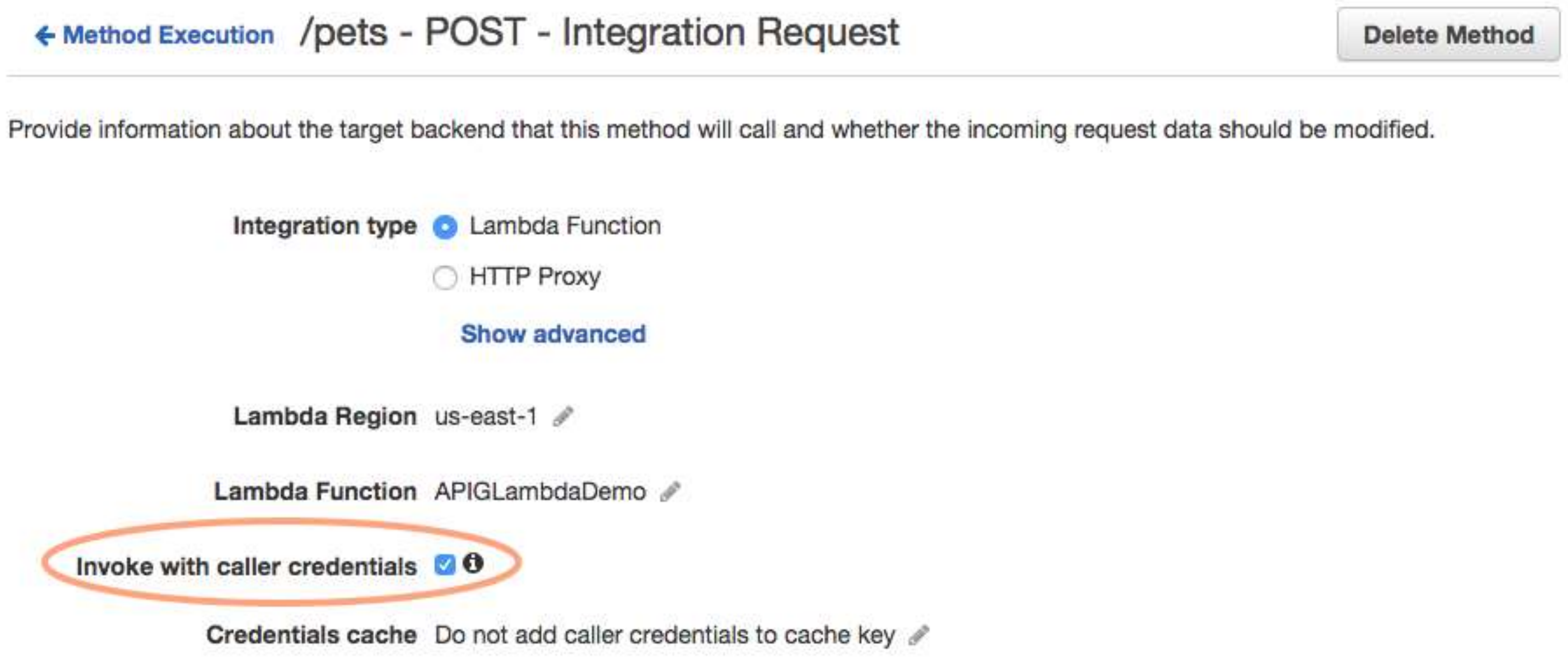


```
- (AWSTask *)refresh {
    PETLambdaMicroserviceClient *client = [PETLambdaMicroserviceClient clientForKey:
        APIGClientConfigurationKey];
    PETRegisterUserRequest *req = [PETRegisterUserRequest new];
    req.username = _credentials.username;
    req.password = _credentials.password;
    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;
    }];
}
```

AWS Services can use caller credentials



/pets - POST - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type	[✓ Lambda Function] HTTP Proxy
Lambda Region	us-east-1
Lambda Function	APIGLambdaDemo
Invoke with caller credentials	✓
Credentials cache	Do not add caller credentials to cache key

Throttling and Caching

API Throttling

Throttling helps you manage traffic to your backend

Throttle by developer-defined Requests/Sec limits

Requests over the limit are throttled

- HTTP 429 response

The generated SDKs retry throttled requests

Caching of API Responses

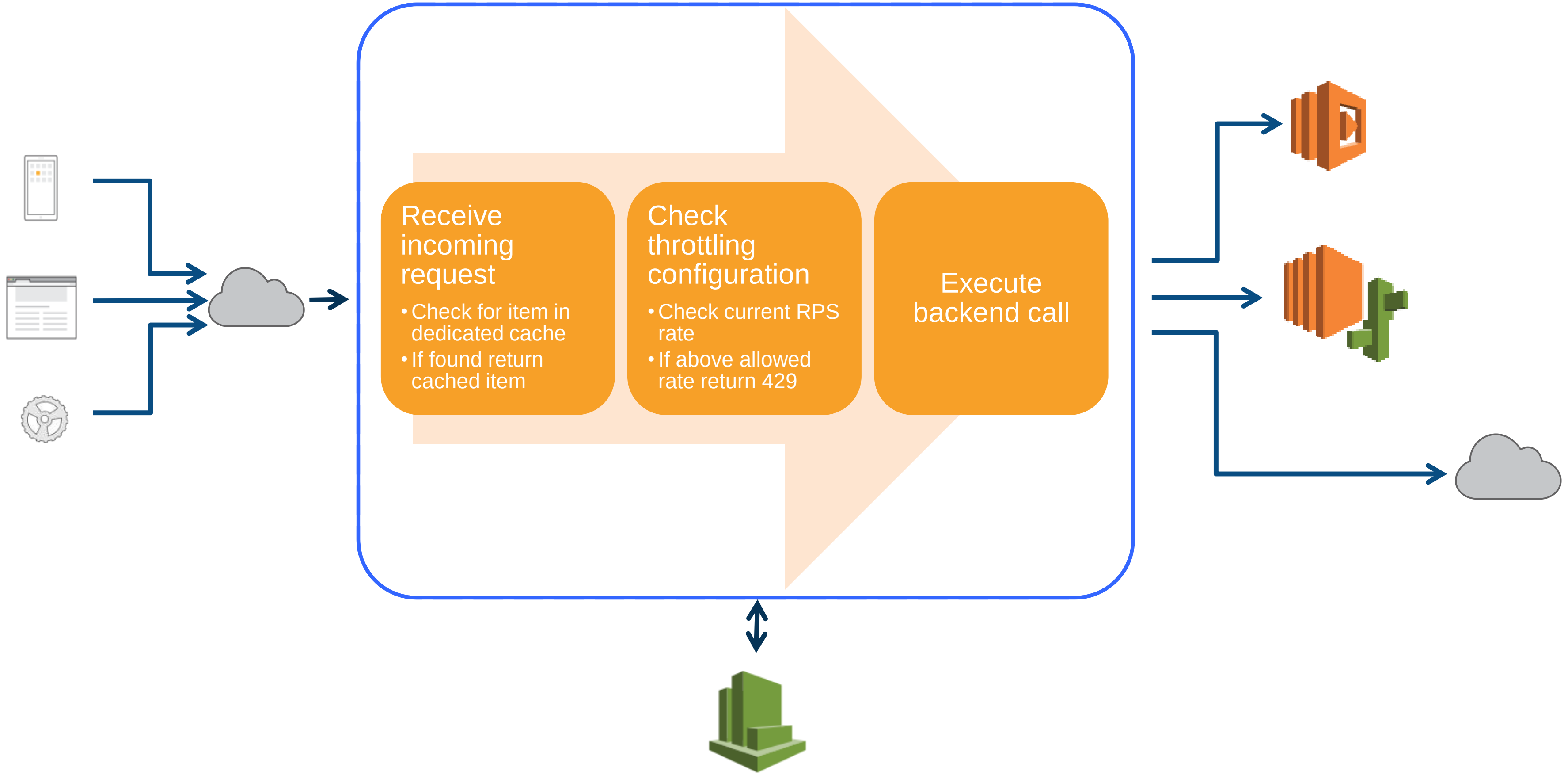
You can configure a cache key and the Time to Live (TTL) of the API response

Cached items are returned without calling the backend

A cache is dedicated to you, by stage

You can provision between 0.5GB to 237GB of cache

Request processing workflow


Receive incoming request
• Check for item in dedicated cache
• If found return cached item
Check throttling configuration
• Check current RPS rate
• If above allowed rate return 429
Execute backend call

Input / Output Transformation

Input / Output Transforms

Use Velocity Templates to transform data

Filter output results

- Remove private or unnecessary data
- Filter dataset size to improve API performance

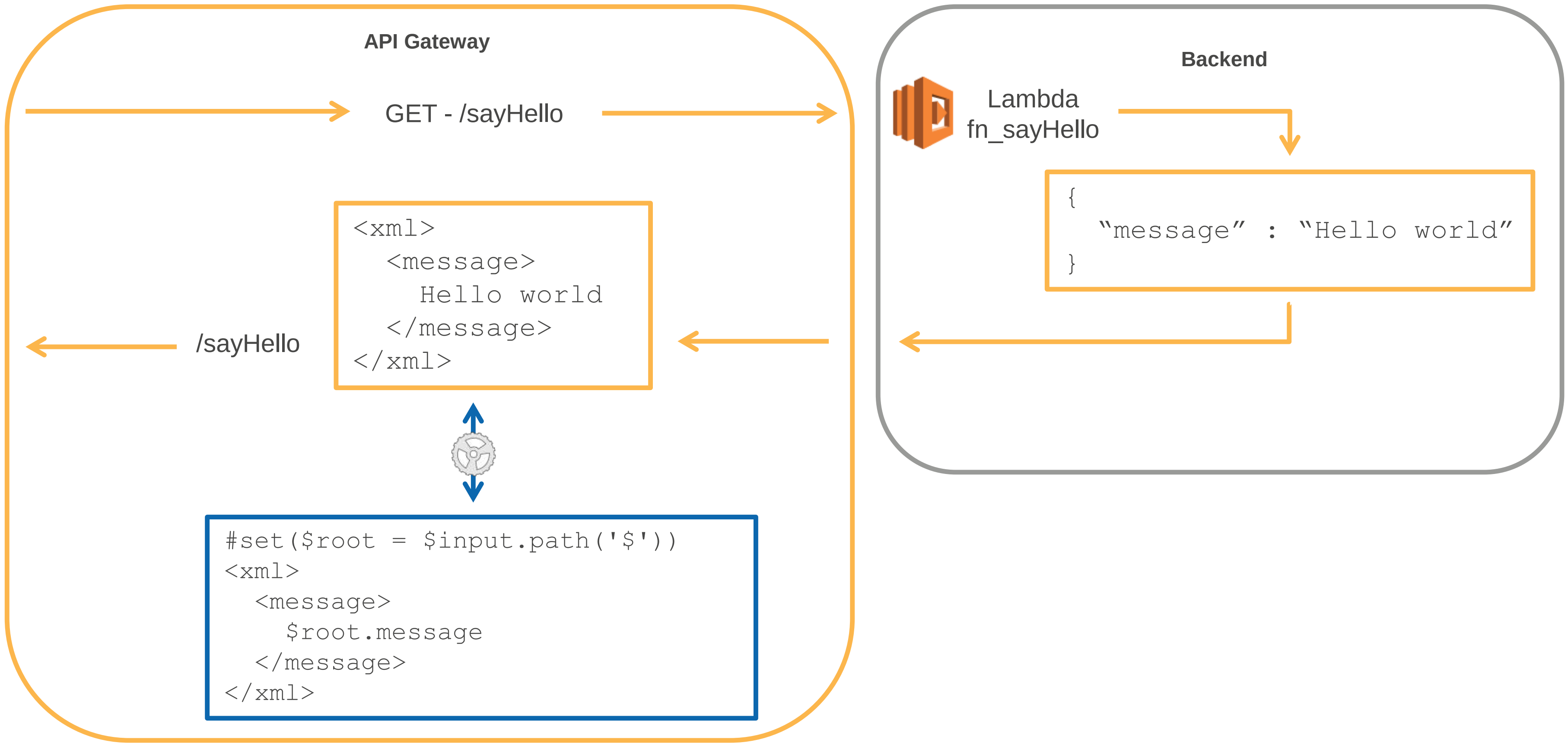
GET to POST

- Read all query string parameters from your GET request, and create a body to make a POST to your backend

JSON to XML

- Receive JSON input and transform it to XML for your backend
- Receive JSON from a Lambda function and transform it to XML

Transform Example: JSON to XML

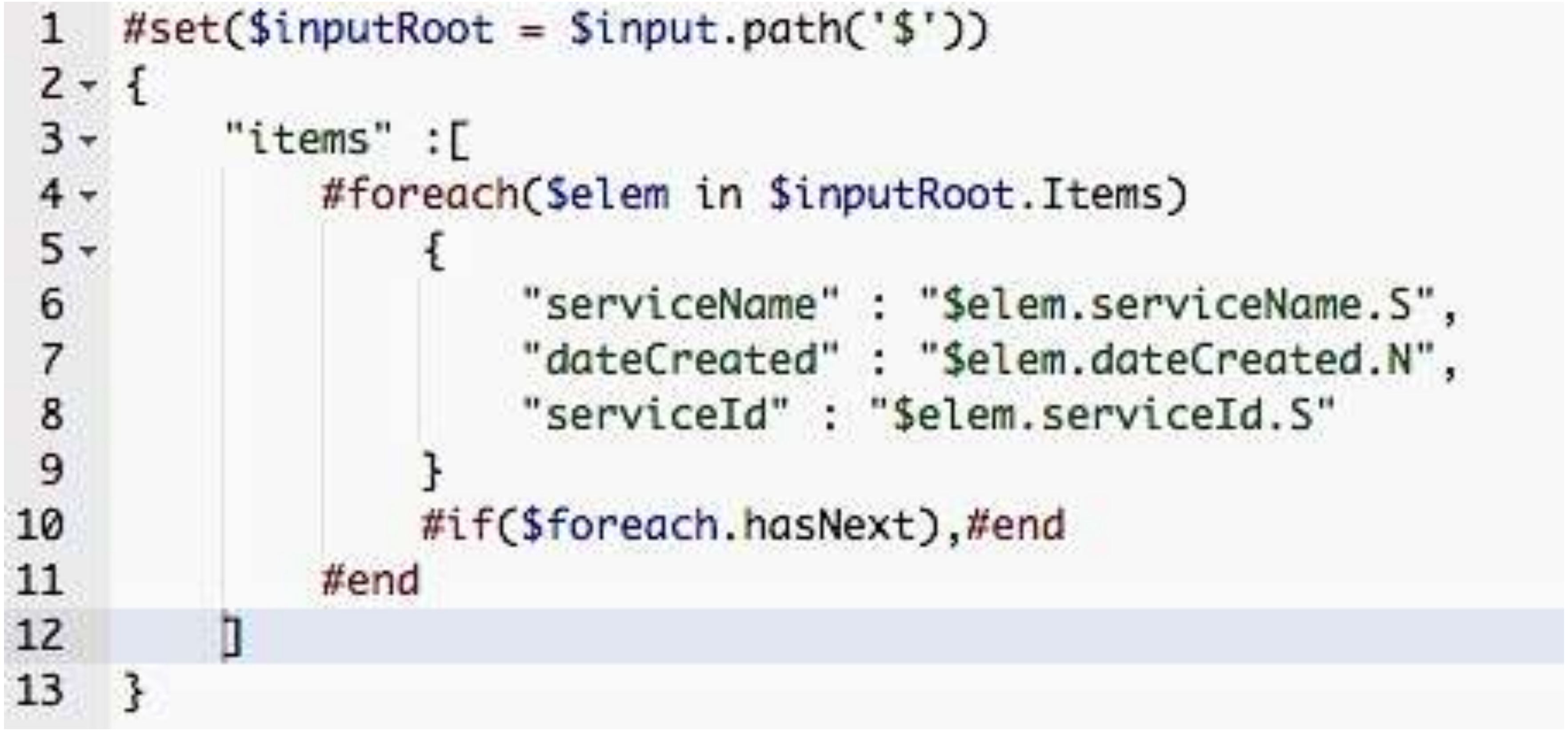


```
{
  "message" : "Hello world"
}
```

```
<xml>
  <message>
    Hello world
  </message>
</xml>
```

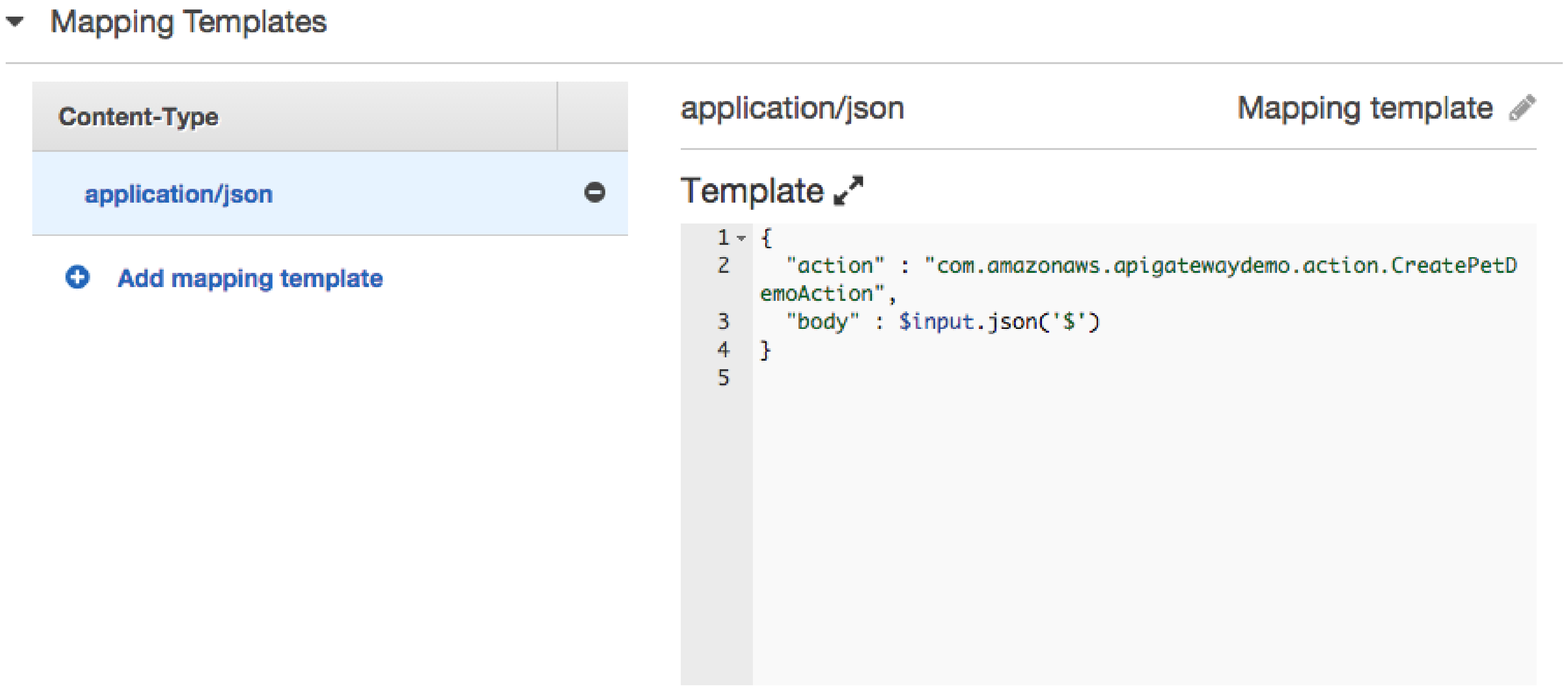
```
#set($root = $input.path('$'))
<xml>
  <message>
    $root.message
  </message>
</xml>
```

For Loops and if Statements



```
#set($inputRoot = $input.path('$'))
{
    "items" :[
        #foreach($elem in $inputRoot.Items)
            {
                "serviceName" : "$elem.serviceName.S",
                "dateCreated" : "$elem.dateCreated.N",
                "serviceId" : "$elem.serviceId.S"
            }
            #if($foreach.hasNext),#end
        #end
    ]
}
```

One Template per Content/Type



```
{
  "action" : "com.amazonaws.apigatewaydemo.action.CreatePetDemoAction",
  "body" : $input.json('$')
}
```

SDK Generation

API Models

Models are a JSON Schema representation of your API requests and responses

You can reuse models across multiple methods in your API

Models are used to generate objects for the client SDK

Generate Client SDKs Based on Your APIs

SDKs are generated based on API deployments (Stages)

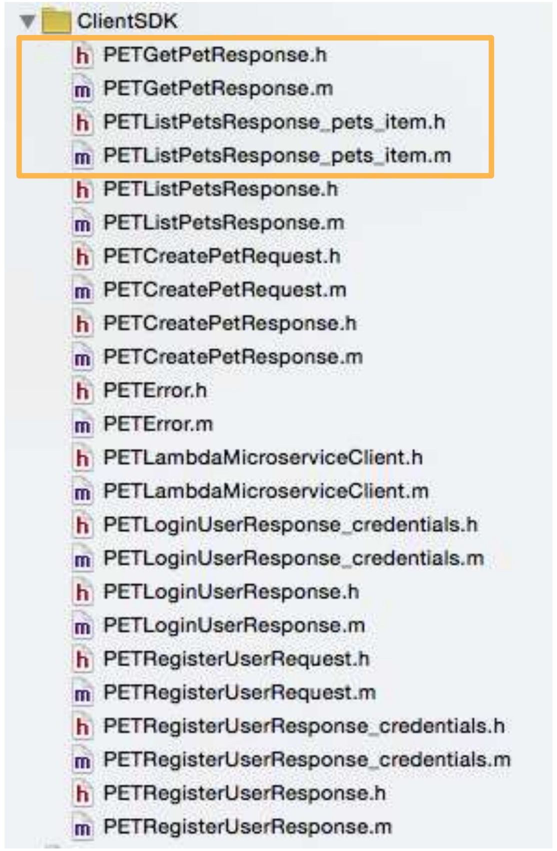
If Request and Response Models are defined, the SDK includes input and output marshalling of your methods

SDKs know how to handle throttling responses

SDKs also know how to sign requests with AWS temporary credentials (SigV4)

Support for Android, iOS, JavaScript, …

Models are included in the SDK

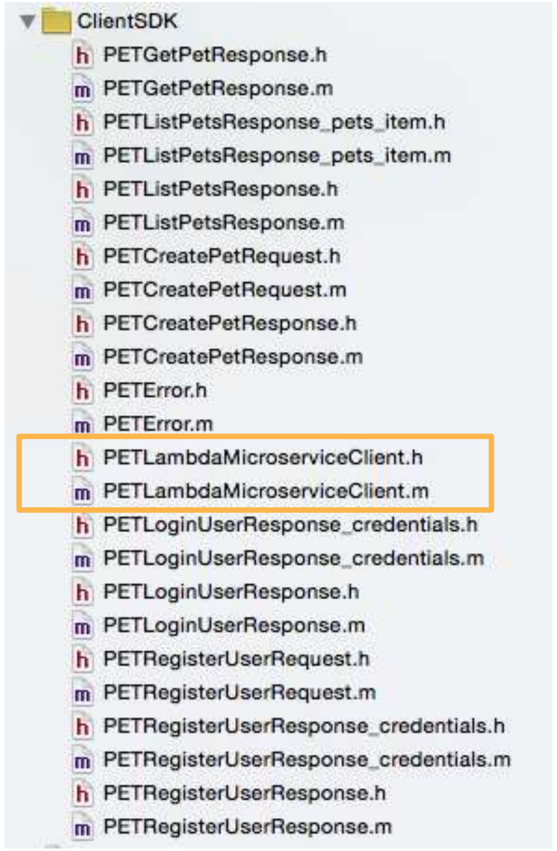


```
▼ ClientSDK
    PETGetPetResponse.h
    PETGetPetResponse.m
    PETListPetsResponse_pets_item.h
    PETListPetsResponse_pets_item.m
    PETListPetsResponse.h
    PETListPetsResponse.m
    PETCreatePetRequest.h
    PETCreatePetRequest.m
    PETCreatePetResponse.h
    PETCreatePetResponse.m
    PETError.h
    PETError.m
    PETLambdaMicroserviceClient.h
    PETLambdaMicroserviceClient.m
    PETLoginUserResponse_credentials.h
    PETLoginUserResponse_credentials.m
    PETLoginUserResponse.h
    PETLoginUserResponse.m
    PETRegisterUserRequest.h
    PETRegisterUserRequest.m
    PETRegisterUserResponse_credentials.h
    PETRegisterUserResponse_credentials.m
    PETRegisterUserResponse.h
    PETRegisterUserResponse.m
```

Models are generated using their name

Nested items inherit the name of their parent model

The Client SDK Declares All Methods



- ClientSDK
  - PETGetPetResponse.h
  - PETGetPetResponse.m
  - PETListPetsResponse_pets_item.h
  - PETListPetsResponse_pets_item.m
  - PETListPetsResponse.h
  - PETListPetsResponse.m
  - PETCreatePetRequest.h
  - PETCreatePetRequest.m
  - PETCreatePetResponse.h
  - PETCreatePetResponse.m
  - PETError.h
  - PETError.m
  - PETLambdaMicroserviceClient.h
  - PETLambdaMicroserviceClient.m
  - PETLoginUserResponse_credentials.h
  - PETLoginUserResponse_credentials.m
  - PETLoginUserResponse.h
  - PETLoginUserResponse.m
  - PETRegisterUserRequest.h
  - PETRegisterUserRequest.m
  - PETRegisterUserResponse_credentials.h
  - PETRegisterUserResponse_credentials.m
  - PETRegisterUserResponse.h
  - PETRegisterUserResponse.m

```
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;
}];
```

Thank You

Q&A