IJUG MEETUP APRIL 25, 2017

TAKING YOUR JAVA WEB APPS SERVERLESS

CHRIS JOHNSON BIDLER

- chris@hlprmnky.com
- Software engineer with ~17 years experience
- Mostly Java, some Ruby, some Clojure, one iOS app in the App Store
- Mostly backend, medium-to-Big Data, increasing focus on having and evangelizing “the devops”
- Currently a Senior Cloud Computing Engineer at TransUnion

EVOLUTION TOWARD SERVERLESS

- IaaS ("legacy") - you have some servers, you run software on them
  - AWS EC2, Google Compute Engine, Azure VMs
- Best practices are well-understood after 2.5 decades
- Hard problems: scaling, deployment, infrastructure management

EVOLUTION TOWARD SERVERLESS

- Containers ("PaaS") - you build and run containers on managed IaaS
  - AWS ECS, Google GKE, Azure Container Service/Docker Swarm
- Best practices are not "legacy" yet but are fairly well-understood
- Hard problems: service discovery, cluster management, container config management

EVOLUTION TOWARD SERVERLESS

- Serverless (what PaaS actually is) - you deploy functions and they run …somewhere
  - AWS Lambda, Google Cloud Functions, Azure Functions
- Practices are still actively evolving
- Hard problems: managing your API->function mappings, local integration testing of cloud things

WHY TAKE AN EXISTING APP SERVERLESS?

- Lower - possibly **much** lower - stack TCO
- Decreased complexity of deployment
  - Nothing to scale in compute-land
  - No security patches to apply
- Better software architecture arises naturally (?)
- Increased stack agility and scalability
  - More granular, lower “floor” (zero)

AWS LAMBDA – FUNCTIONS IN THE CLOUD

- https://aws.amazon.com/lambda/
- Run stateless functions as a service
- Runtimes for Java, node.js, Python, C#, C++
- Function can be triggered by many events in the AWS ecosystem, including being the target of API Gateway endpoints
- This is the business logic layer of a Serverless app

AWS LAMBDA – PROGRAMMING MODEL

- Each function receives from the Lambda service:
  - an *event* which has information about the trigger
    - S3 bucket PUT will have the URI of the PUT object
    - IoT event will have the event data
    - API Gateway call will have the HTTP request

AWS LAMBDA – PROGRAMMING MODEL

- Each function receives from the Lambda service:
  - a *context* which represents the execution context
    - can have information about the requesting application
    - has a Logger (in Java) that writes out to CloudWatch
    - can get remaining execution time before throttle, function version and name, other useful information

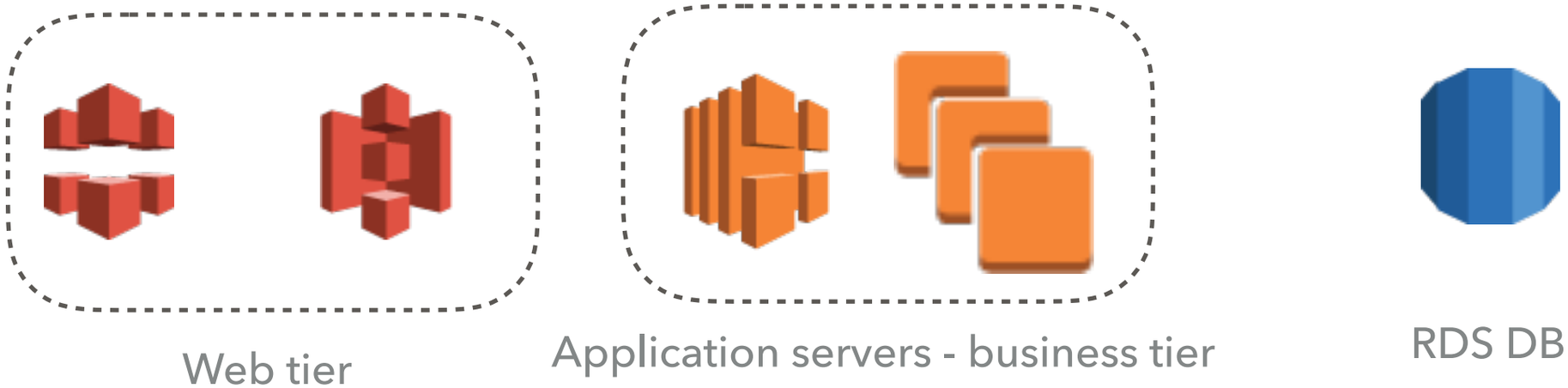
AWS API GATEWAY – MANAGED REST API

- https://aws.amazon.com/api-gateway/
- API-as-a-service from AWS, managed scale (via CloudFront), security/auth, versioning, dev/test staging
- Can import existing Swagger APIs
- Each endpoint can forward to different services
- Can have wildcard endpoints
  - Existing routing layer can make calls to other Lambda

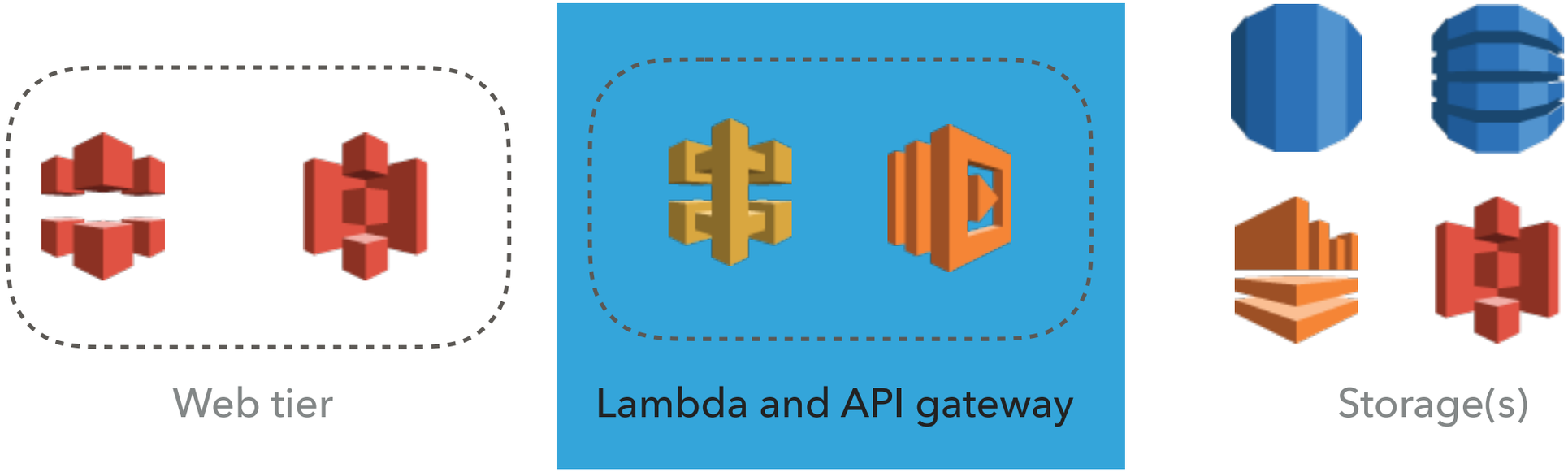
OTHER AWS SERVICES

- Lots of AWS services can do work you might do in application code today
- Step Functions - State machine for workflow management
- S3 - Lambda acting on S3 can be the “then process this data” step
- Kinesis - Event streams can be visited/processed by Lambda
- Cognito - Designed for mobile, but can also be your auth layer for AWS-native applications generally

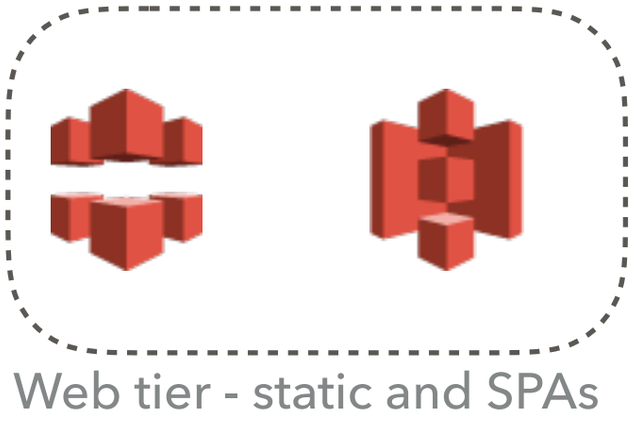
SHOPPING CART APP – EXISTING ARCHITECTURE



SHOPPING CART APP – EVENTUAL ARCHITECTURE

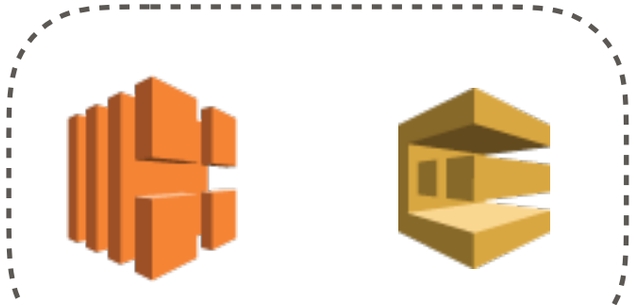


PAIN POINT – HOW TO DEPLOY ALL THESE MOVING PARTS

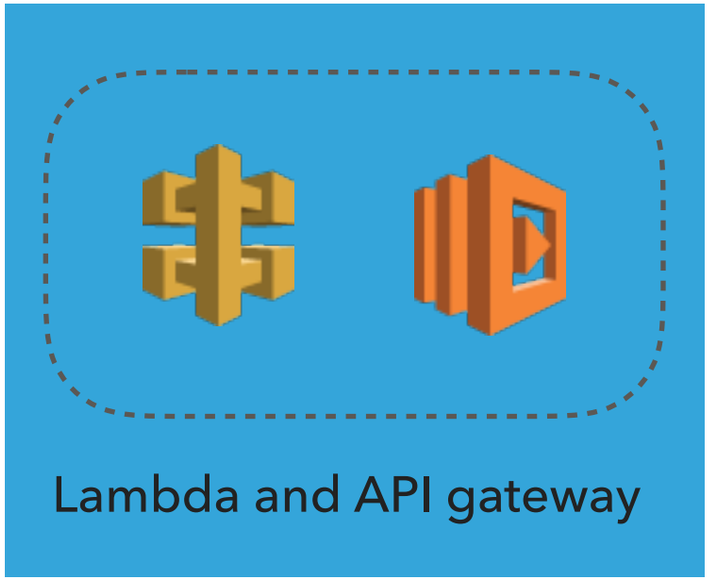


Web tier - static and SPAs

Mobile apps



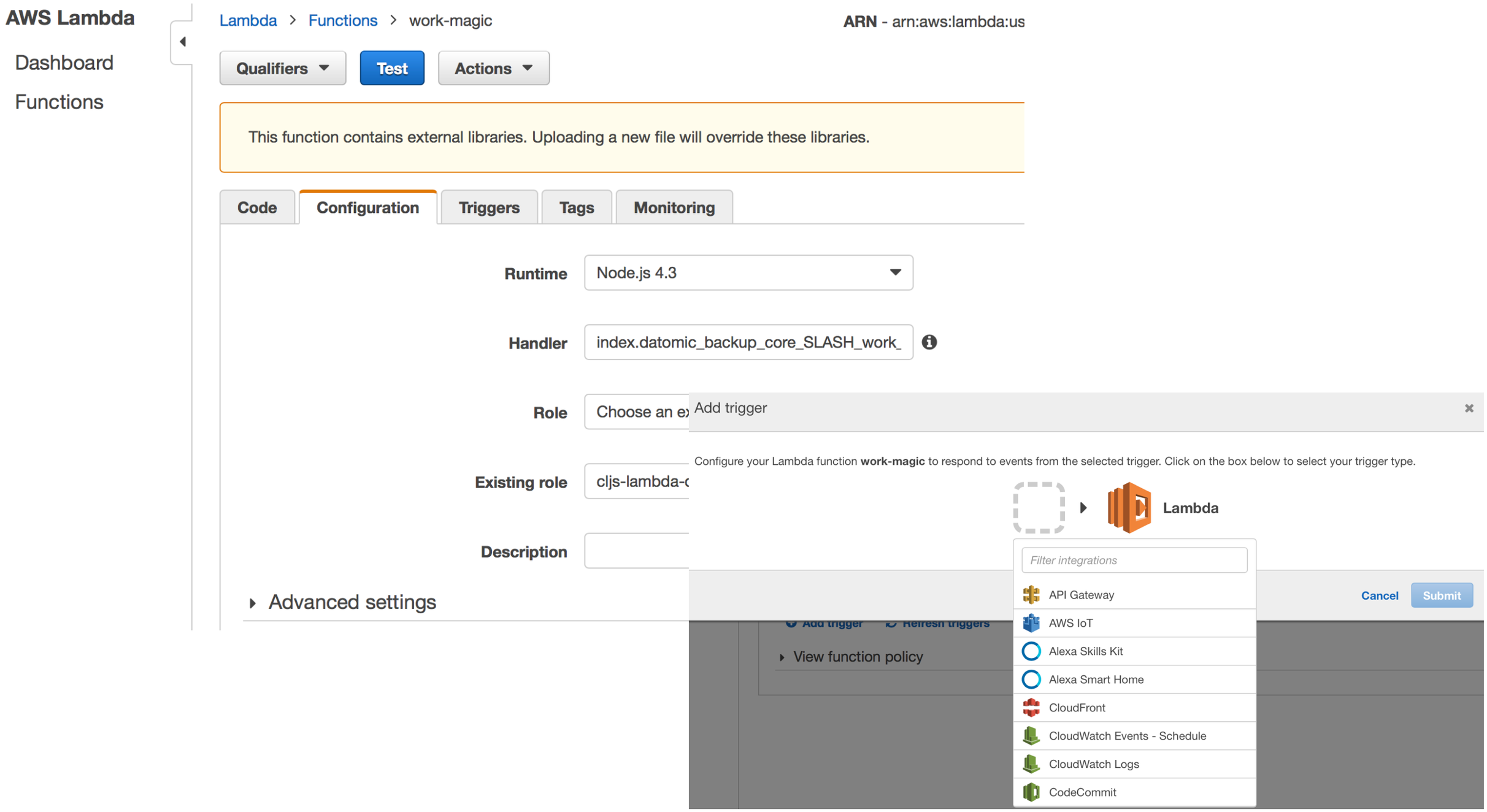
B2B integrations



Lambda and API gateway

Storage(s)

NOT AS SIMPLE AS THE DIAGRAM MAKES IT LOOK



NOT AS SIMPLE AS THE DIAGRAM MAKES IT LOOK

Create new API

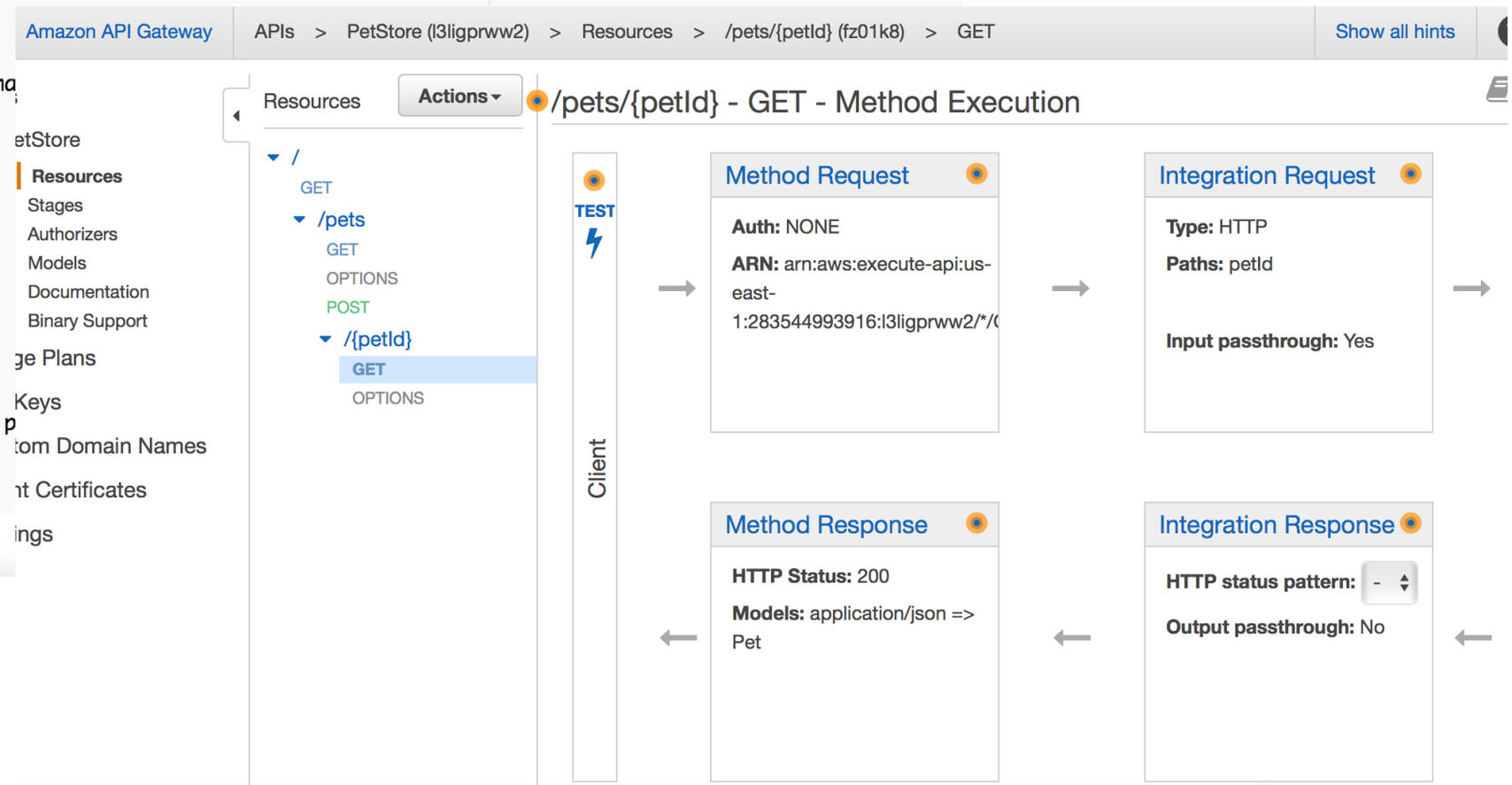
In Amazon API Gateway, an API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

○ **New API** ○ **Import from Swagger** ◉ **Example API**

Example API

Learn about the service by importing an example API and turning on hints throughout the console.

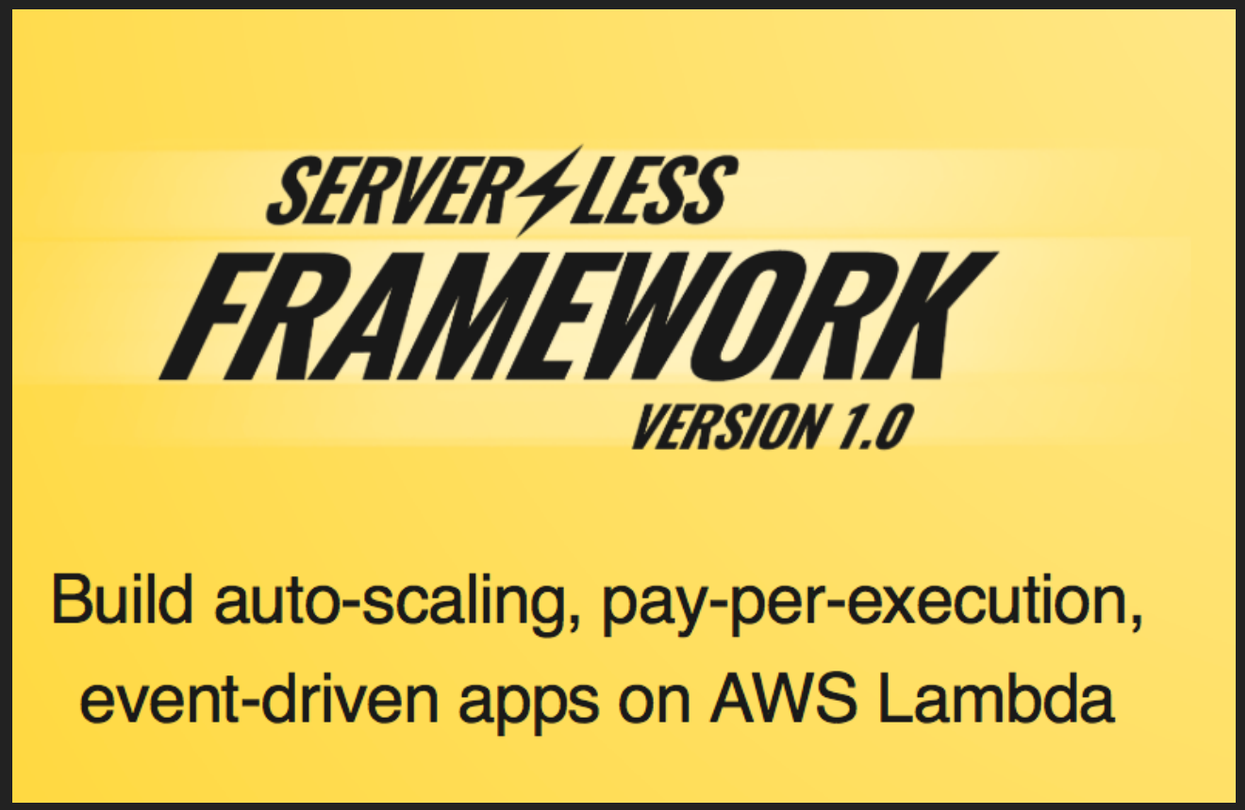
```
{
  "swagger": "2.0",
  "info": {
    "description": "Your first API with Ama
    "title": "PetStore"
  },
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "get": {
        "tags": [
          "pets"
        ],
        "description": "PetStore HTML web p
        "consumes": [
          "application/json"
        ],
```



INFRASTRUCTURE AS CODE – DON'T DO PAINFUL THINGS BY HAND

- Serverless - https://serverless.com/
  - Build and deploy apps in one go
  - 1.0 release recently
  - First, best support for AWS but cloud-agnostic
- AWS Serverless Application Model (SAM)
  - First-class Cloudformation support for serverless architecture

THE SERVERLESS FRAMEWORK



- Supports (nearly) all AWS Lambda runtimes, Azure, OpenWhisk
- Build/test/deploy from developer machine or CI pipeline
- Really pretty good local testing story

SERVERLESS FRAMEWORK

2. chris@bitsled: ~/src/tech-talks/ijug-chicago/serverless-migration (zsh)

```
 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration npm list -g serverless
/usr/local/lib
└── serverless@1.11.0

 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration serverless create --help
Plugin: Create
create ......................... Create new Serverless service
    --template / -t (required) ......... Template for the service. Available templates: "aws-nodejs"
, "aws-python", "aws-groovy-gradle", "aws-java-maven", "aws-java-gradle", "aws-scala-sbt", "aws-csha
rp", "azure-nodejs", "openwhisk-nodejs" and "plugin"
    --path / -p ......................... The path where the service should be created (e.g. --path m
y-service)
    --name / -n ......................... Name for the service. Overwrites the default name of the cr
eated service.

 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration
```

SERVERLESS FRAMEWORK

2. chris@bitsled: ~/src/tech-talks/ijug-chicago/serverless-migration (zsh)

```
 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration serverless create --template a
ws-java-maven --name serverless-demo
Serverless: Generating boilerplate...

 _______                             __
|   _   .-----.----.--.--.-----.----|  .-----.-----.-----.
|   |___|  -__|   _|  |  |  -__|   _|  |  -__|__ --|__ --|
|____   |_____|__|  \___/|_____|__| |__|_____|_____|_____|
|   |   |             The Serverless Application Framework
|       |                           serverless.com, v1.11.0
 -------'

Serverless: Successfully generated boilerplate for template: "aws-java-maven"
 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration tree
.
├── pom.xml
├── serverless.yml
└── src
    └── main
        ├── java
        │   └── com
        │       └── serverless
        │           ├── ApiGatewayResponse.java
        │           ├── Handler.java
        │           └── Response.java
        └── resources
            └── log4j.properties

6 directories, 6 files
 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration
```

SERVERLESS FRAMEWORK

```java
package com.serverless;

import java.util.Collections;
import java.util.Map;

import org.apache.log4j.Logger;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Handler implements RequestHandler<Map<String, Object>, ApiGatewayResponse> {

    private static final Logger LOG = Logger.getLogger(Handler.class);

    @Override
    public ApiGatewayResponse handleRequest(Map<String, Object> input, Context context) {
        LOG.info("received: " + input);
        Response responseBody = new Response("Go Serverless v1.x! Your function executed successfully!", input);
        return ApiGatewayResponse.builder()
                .setStatusCode(200)
                .setObjectBody(responseBody)
                .setHeaders(Collections.singletonMap("X-Powered-By", "AWS Lambda & serverless"))
                .build();
    }
}

```

SERVERLESS FRAMEWORK – CONFIGURATION

```
80
81      #    Define function environment variables here
82          environment:
83            variable2: value2
84
85      # you can add CloudFormation resource templates here
86      resources:
87        Resources:
88          NewResource:
89            Type: AWS::S3::Bucket
90            Properties:
91              BucketName: my-new-bucket
92      Outputs:
93          NewOutput:
94            Description: "Description for the output"
95            Value: "Some output value"
```

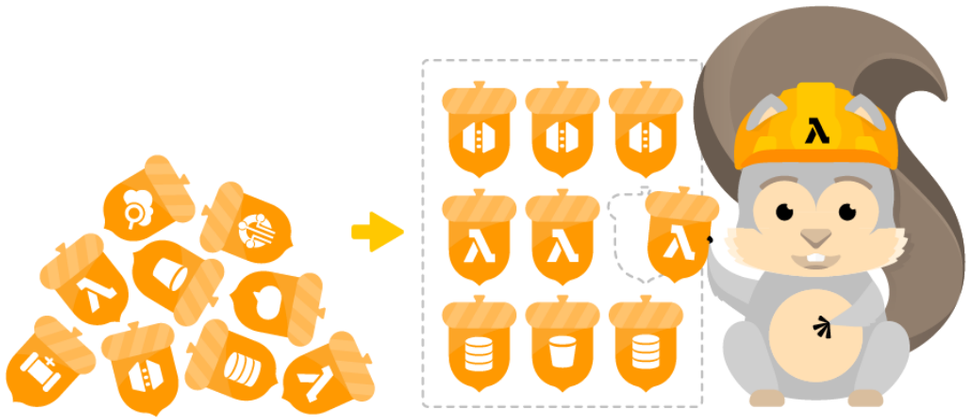
SERVERLESS FRAMEWORK – DEPLOY AND RUN

2. chris@bitsled: ~/src/tech-talks/ijug-chicago/serverless-migration (zsh)

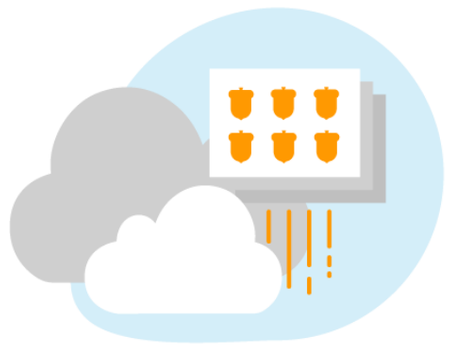
```
Serverless: Stack create finished...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading function .zip files to S3...
Serverless: Uploading service .zip file to S3 (1.98 MB)...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.............
Serverless: Stack update finished...
Service Information
service: serverless-demo
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  None
functions:
  hello: serverless-demo-dev-hello
 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration serverless invoke -f hello
{
    "statusCode": 200,
    "body": "{\"message\":\"Go Serverless v1.x! Your function executed successfully!\",\"input\":{}}
",
    "headers": {
        "X-Powered-By": "AWS Lambda & serverless"
    },
    "isBase64Encoded": false
}
 chris@bitsled ~/src/tech-talks/ijug-chicago/serverless-migration
```

AWS SERVERLESS APPLICATION MODEL

MEET SAM.



USE SAM TO BUILD TEMPLATES THAT DEFINE YOUR SERVERLESS APPLICATIONS.



DEPLOY YOUR SAM TEMPLATE WITH AWS CLOUDFORMATION.

- AWS liked Serverless a lot ….so they made one
- They gave it a cute squirrel mascot

AWS SERVERLESS APPLICATION MODEL (SAM)

- https://github.com/awslabs/serverless-application-model
- Developed by AWS, released November of 2016
- AWS-specific (obviously)
- CloudFormation support for a higher level abstraction over the Lambda/API Gateway/DynamoDB design pattern - AWS::Serverless::Function
- Infrastructure-as-code model for deploying full applications

AWS SAM

- Hop over to VSCode to look at example files

MIGRATING AN EXISTING APP TO SERVERLESS

- Ensure that your boundaries of concerns are delineated by APIs rather than convention or just DI
- Move handling of state to “the edge” - explicitly separated from your business logic
  - Business logic should be, as much as possible, pure functions
- Separate your new mostly-stateless (and some state) functions into individual Lambdas

STEP 1: PUT AN API ON IT

```
package com.hlprmnky.widgco.webstore.shoppingcart;

import com.hlprmnky.widgco.webstore.catalog.*;
import com.hlprmnky.widgco.webstore.userauth.User;

public class ShoppingCartSession {
    private User accountUser;
    private EntityManager dbHandle;

    pubilc ShoppingCartSession(User accountUser, EntityManger dbHandle) throws Exception {
        if(!accountUser.isValid()) {
            throw new UserSessionExpiredException;
        }
        this.session = session;
        this.dbHandle = dbHandle;
    }

    public void addToCart(ItemId id, int requestedQty) {
        UserShoppingCart cart = getShoppingCart(dbHandle.findById, ShoppingCart.class);
        Item item = dbHandle.findById(id, CatalogItem.class);
        if(item) {
            cart.add(item, requestedQty);
            // handle decrement of on hand qty here
            // etc.
        }
    }
}
```

STEP 1: PUT AN API ON IT

```
1   package com.hlprmnky.widgco.webstore.shoppingcart;
2
3   import com.hlprmnky.widgco.service.CatalogService;
4   import com.hlprmnky.widgco.service.UserService;
5
6   public class ShoppingCartSession {
7       private UserService userService;
8       private CatalogService catalogService;
9
10      pubilc ShoppingCartSession(UserService userService, CatalogService catalogService) {
11          this.userService = userService;
12          this.catalogService = catalogService;
13      }
14
15      public void addToCart(UserId userId, ItemId itemId, int requestedQty) {
16          if(userService.valid(userId)) {
17              Item item = catalogService.findById(id);
18              if(item && item.onHandQty >= requestedQty) {
19                  userService.addToCart(userId, item, requestedQty);
20                  catalogService.reduceOnHandQty(item, requestedQty);
21              }
22          }
23      }
24  }
25
```

STEP 2: STATE TO THE EDGES

```
public ShoppingCart addToCart(UserId userId, ItemId itemId, int requestedQty) {
    if(userService.valid(userId)) {
        Item item = catalogService.findById(id);
        if(item && item.onHandQty >= requestedQty) {
            ShoppingCart userCart = userService.getShoppingCart(userId);
            userCart.add(item, requestedQty);
            catalogService.reduceOnHandQty(item, requestedQty);
            return userCart;
        } else {
            throw new InsufficientInventoryException;
        }
    }
    return null;
}
```

STEP 2: STATE TO THE EDGES – OR EVEN:

```
16      /*
17       * Note that someone else validates the user and fetches their cart here
18       */
19      public ShoppingCart addToCart(ShoppingCart userCart, ItemId itemId, int requestedQty) {
20          Item item = catalogService.findById(id);
21          if(item && item.onHandQty >= requestedQty) {
22              userCart.add(item, requestedQty);
23              catalogService.reduceOnHandQty(item, requestedQty);
24              userCart.add(item, requestedQty);
25              return userCart;
26          } else {
27              throw new InsufficientInventoryException;
28          }
29      }
```

STEP 3: CARVE OUT FUNCTIONS INTO LAMBDA

- If you're already using or adopting a microservices architecture, you're ahead of the game here
- Individual API methods need to be broken out of classes into functions
- Tested Lambda functions get put behind API Gateway endpoints
- Blue/green or just point the ELB for '/catalogService' at API Gateway for some or all users

WHITHER STATE?

- In functions whose explicit only job is updating state
- persistUserCart(UserId, ShoppingCart)
  - not addToCart(…) { userService.persist(cart); }
- Makes testing of most functions that make up your business logic simple
- Can still store state in RDS, DynamoDB, on-prem Oracle, whatever

WHITHER LOGS?

- Splunk has native Lambda support for HTTPEventCollector
- SumoLogic has similar native support for CloudWatch, Kinesis, S3 file logging of events
- Can also write log statements out to CloudWatch
- Consider writing *events* to Kinesis instead of *log messages* to some other service

QUESTIONS?

THANKS FOR WATCHING