
Why are we
here today?

Microservices

"is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. These services are small, highly decoupled and focus on doing a small task, facilitating a modular approach to system-building." - Wikipedia

https://en.wikipedia.org/wiki/Microservices


What makes a microservice
“micro”?

What makes a microservice ”micro”?

Too big of a topic to get into depth today! Read about:

- Domain driven design
- Bounded Contexts
- CQRS models
- Smart endpoints, dumb pipes
- Sam Newman’s book “Building Microservices” O’Reilly Publishing is a great place to start!


Building your API

Basic API technology stack

API Management Challenges

Managing multiple versions and stages of an API is difficult.

Monitoring third-party developers' access is time consuming.

Access authorization is a challenge.

Traffic spikes create an operational burden.

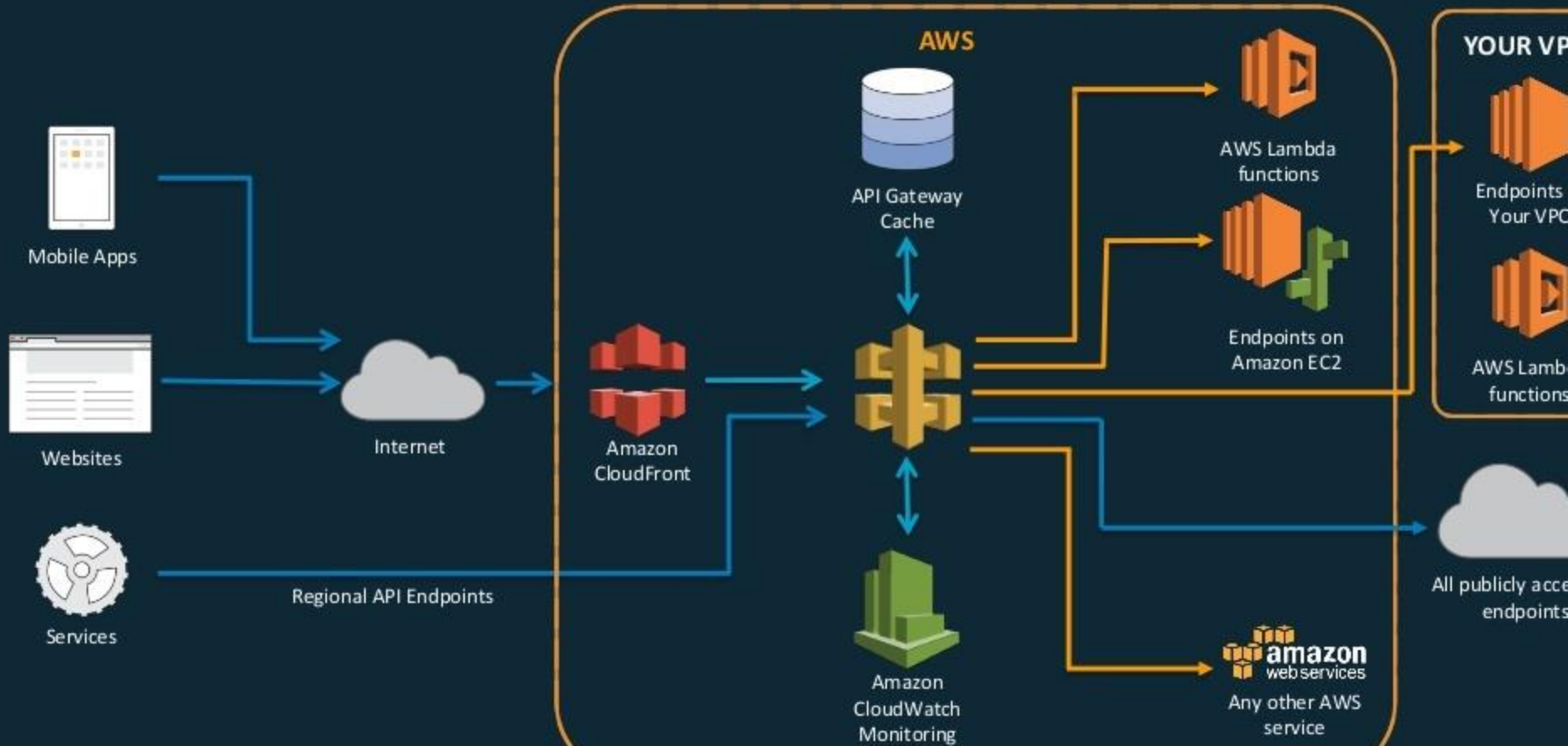
Dealing with increased management overhead

Introducing Amazon API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale:

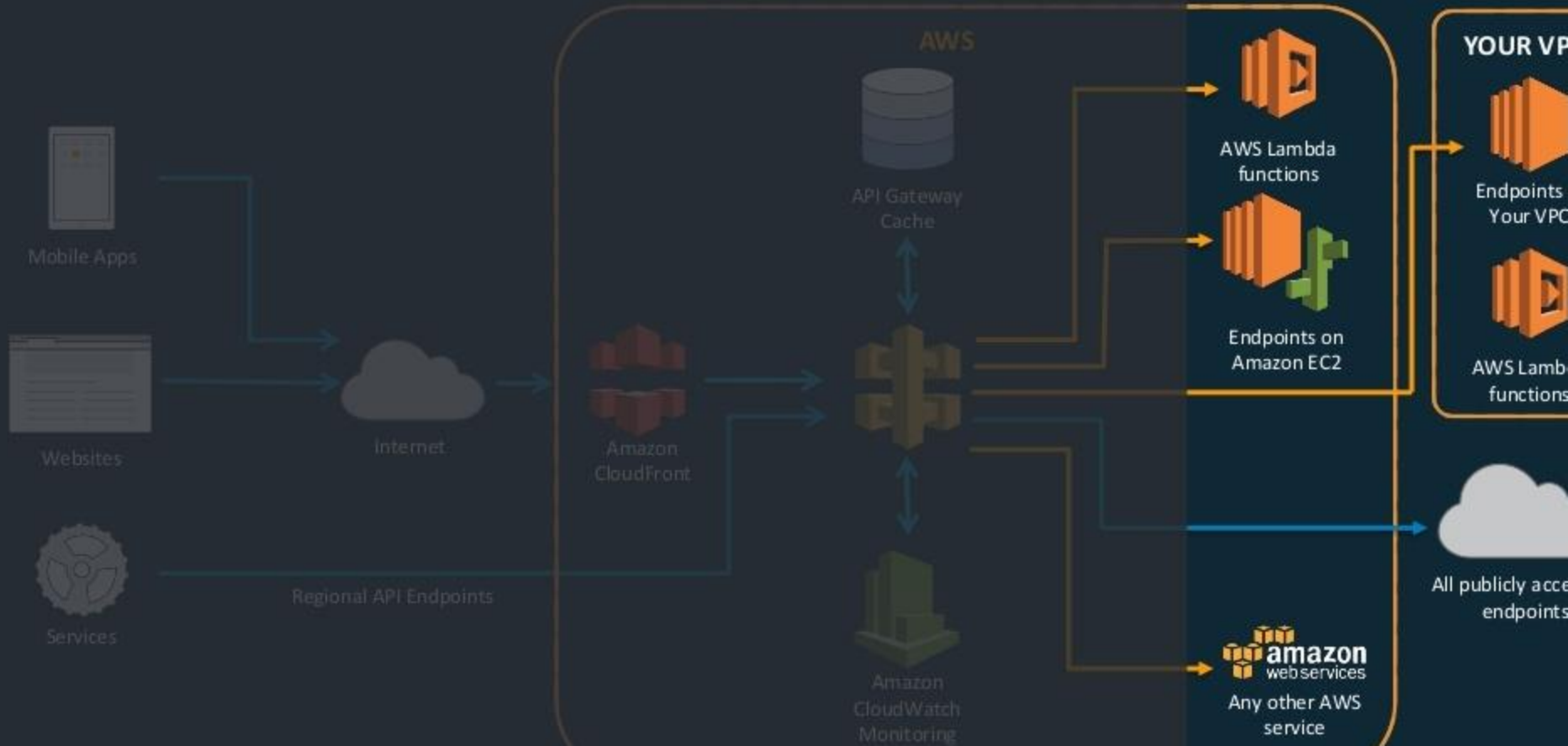
- Host multiple versions and stages of your APIs
- Create and distribute API Keys to developers
- Throttle and monitor requests to protect your backend
- Leverage signature version 4 to authorize access to APIs
- Request / Response data transformation and API mocking
- Reduced latency and DDoS protection through CloudFront
- Optional Managed cache to store API responses
- SDK Generation for Java, JavaScript, Java for Android, Objective-C or Swift for iOS, and Ruby
- Swagger support

API Gateway integrations



Basic API technology stack

API Gateway backend integrations



AWS Compute Services

Amazon
EC2

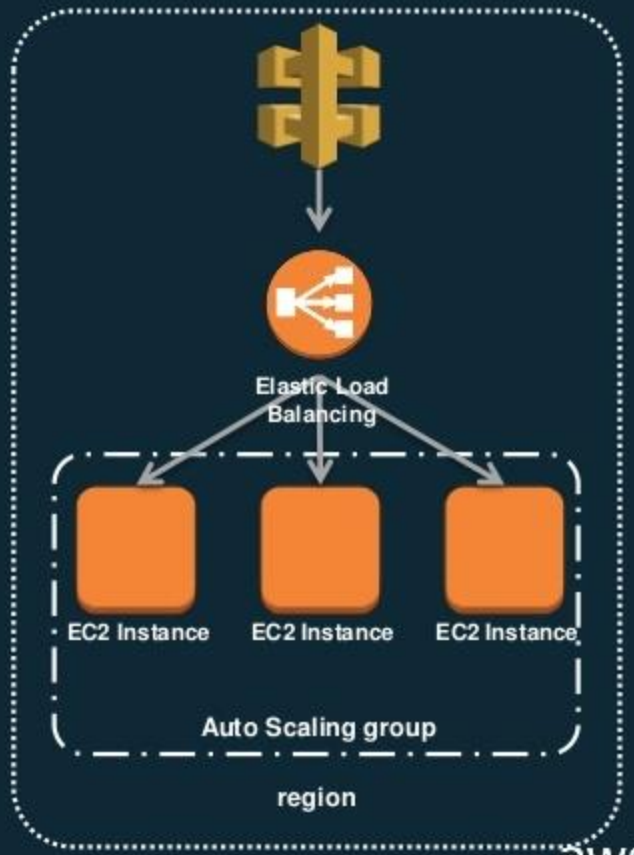
Amazon
Elastic
Container
Service
(ECS)

AWS
Lambda

Deploying Microservices on Amazon EC2

Recommendation:

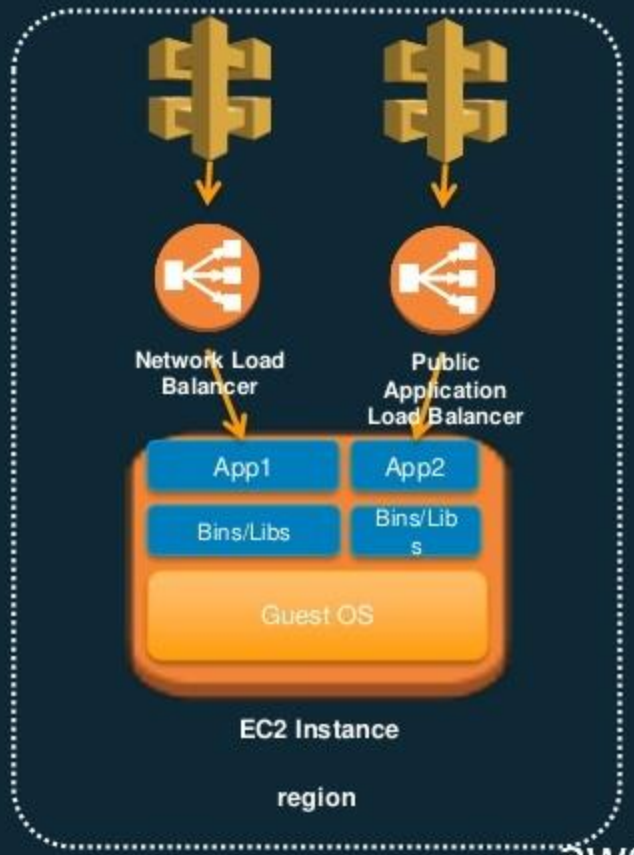
- Single service per host
- Start with small instance sizes
- Leverage Auto Scaling and AWS Elastic Load Balancing/Application Load Balancer/Network Load Balancer(if in VPC)
- Automate the ability to pump out these environments easily
  - Leverage CodeDeploy, CloudFormation, Elastic Beanstalk or Opsworks



Deploying Microservices with ECS

Recommendation

- Put multiple services per host
- Make use of larger hosts with much more CPU/RAM
- Run helper services on the same host as other dependent services
- Leverage Auto Scaling and AWS Elastic Load Balancing/Application Load Balancer/Network Load Balancer(if in VPC)
- Use AWS Fargate for even less administrative overhead!



Serverless means…

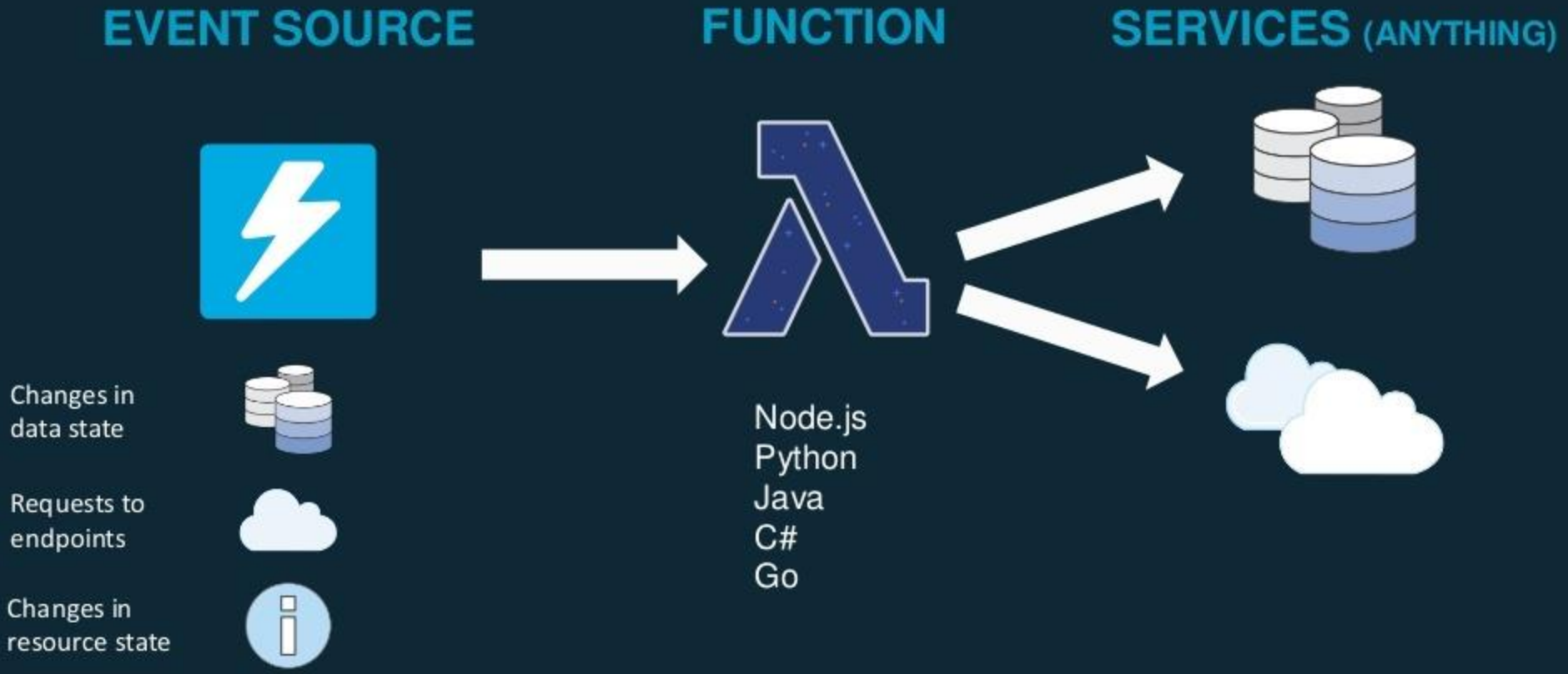
No servers to provision or manage

Scales with usage

Never pay for idle

Availability and fault tolerance built in

Serverless applications



Anatomy of a Lambda function

Handler() function

Function to be executed upon invocation

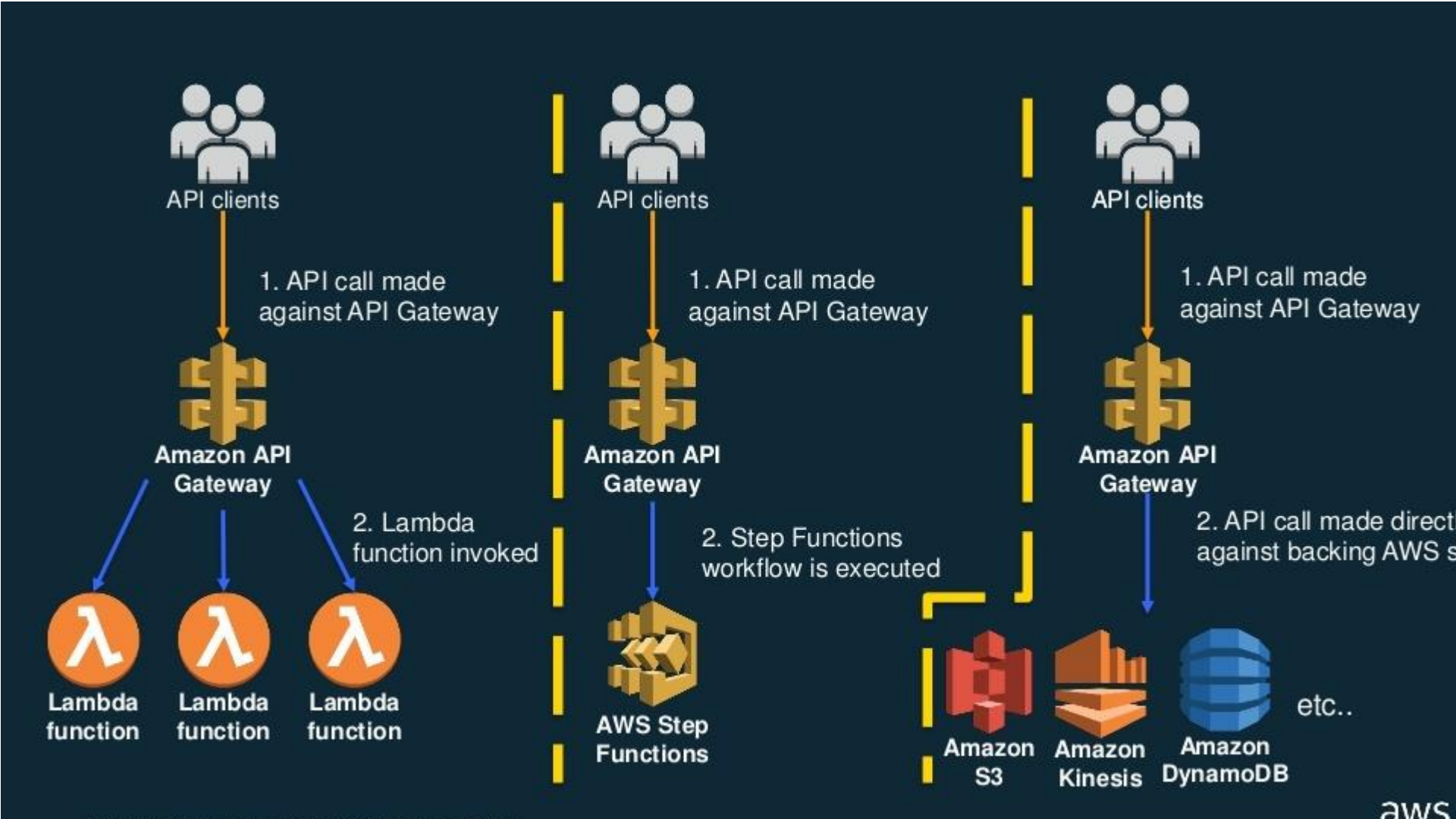
Event object

Data sent during Lambda Function Invocation

Context object

Methods available to interact with runtime information (request ID, log group, etc.)

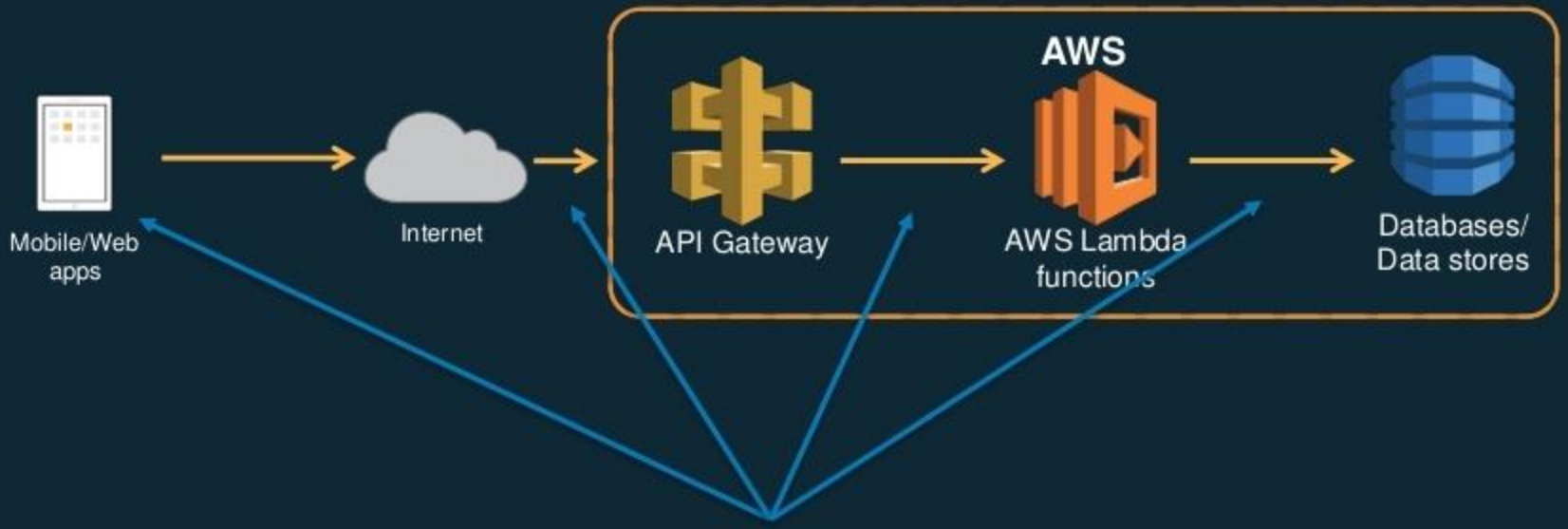
```
public String handleRequest(Book book, Context context) {
    saveBook(book);

    return book.getName() + " saved!";
}
```


API clients
1. API call made against API Gateway
Amazon API Gateway
2. Lambda function invoked
Lambda function
Lambda function
Lambda function
API clients
1. API call made against API Gateway
Amazon API Gateway
2. Step Functions workflow is executed
AWS Step Functions
API clients
1. API call made against API Gateway
Amazon API Gateway
2. API call made direct
against backing AWS s
Amazon S3
Amazon Kinesis
Amazon DynamoDB
etc..

Basic Serverless API technology stack


Secure your API

Basic Serverless API technology stack



Amazon API Gateway Security

Several mechanisms for adding Authz/Authn to our API:

- IAM Permissions
  - Use IAM policies and AWS credentials to grant access
- Custom Authorizers
  - Use Lambda to validate a bearer token(Oauth or SAML as examples) or request parameters and grant access
- Cognito User Pools
  - Create a completely managed user management system

Authentication type comparison

Feature	AWS_IAM	TOKEN	REQUEST	COGNITO
Authentication	X	X	X	X
Authorization	X	X	X	
Signature V4	X			
Cognito User Pools		X	X	X
Third-Party Authentication		X	X	
Multiple Header Support			X	
Additional Costs	NONE	Pay per authorizer invoke	Pay per authorizer invoke	NONE

Cognito User Pools

1

Serverless Authentication and User Management

Add user sign-up and sign-in easily to your mobile and web apps without worrying about server infrastructure

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

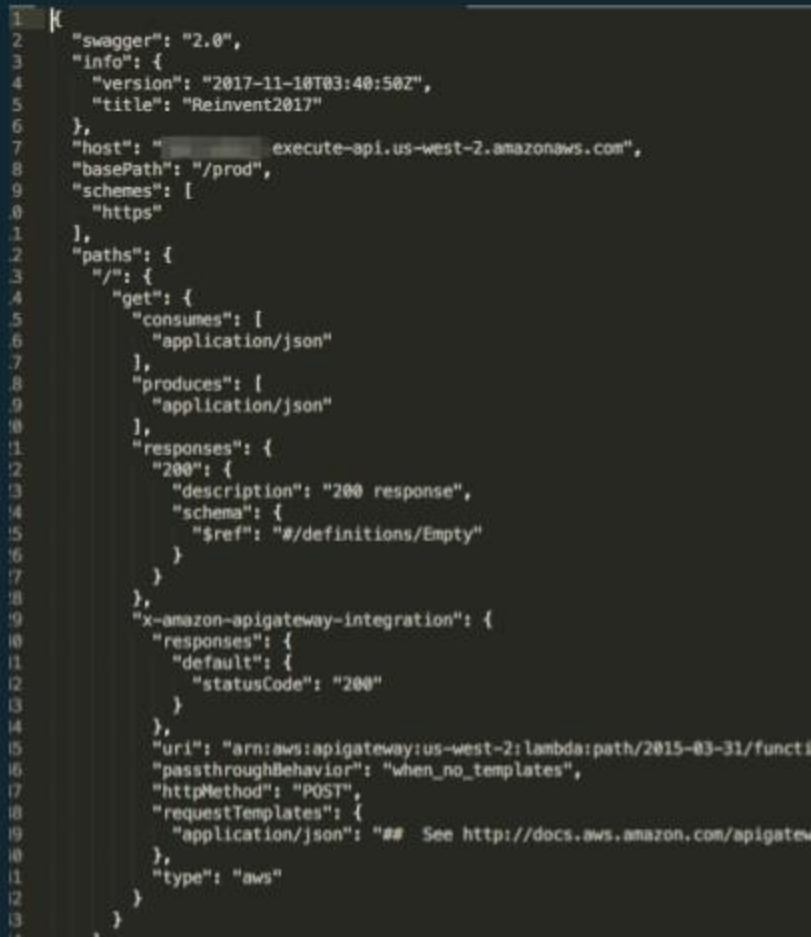
Verify phone numbers and email addresses and offer multi-factor authentication


Where do you ..
START
?

Swagger – now OpenAPI Specification(OAS)

API definition as code:

- Portable API definition
- JSON/YAML
- Import/Export your API
- Amazon API Gateway extensions
- Can be used independently or as part of a CloudFormation template
- Rich 3rd party ecosystem of tools



Frameworks

APEX

aws
Chalice

Claudia.js

SERVERLESS

SPARTA

ClaudiaJS

Node.js framework for deploying projects to AWS Lambda and Amazon API Gateway

- Has sub projects for microservices, chat bots and APIs
- Simplified deployment with a single command
- Use standard NPM packages, no need to learn swagger
- Manage multiple versions

https://claudiajs.com

https://github.com/claudiajs/claudia

app.js:

```
var ApiBuilder = require('claudia-api-builder')
var api = new ApiBuilder();

module.exports = api;

api.get('/hello', function () {
        return 'hello world';
});
```

```
$ claudia create --region us-east-1 --api-module a
```

Chalice

Chalic

Python serverless “microframework” for AWS Lambda and Amazon API Gateway

- A command line tool for creating, deploying, and managing your app
- A familiar and easy to use API for declaring views in python code
- Automatic Amazon IAM policy generation

https://github.com/aws/chalice

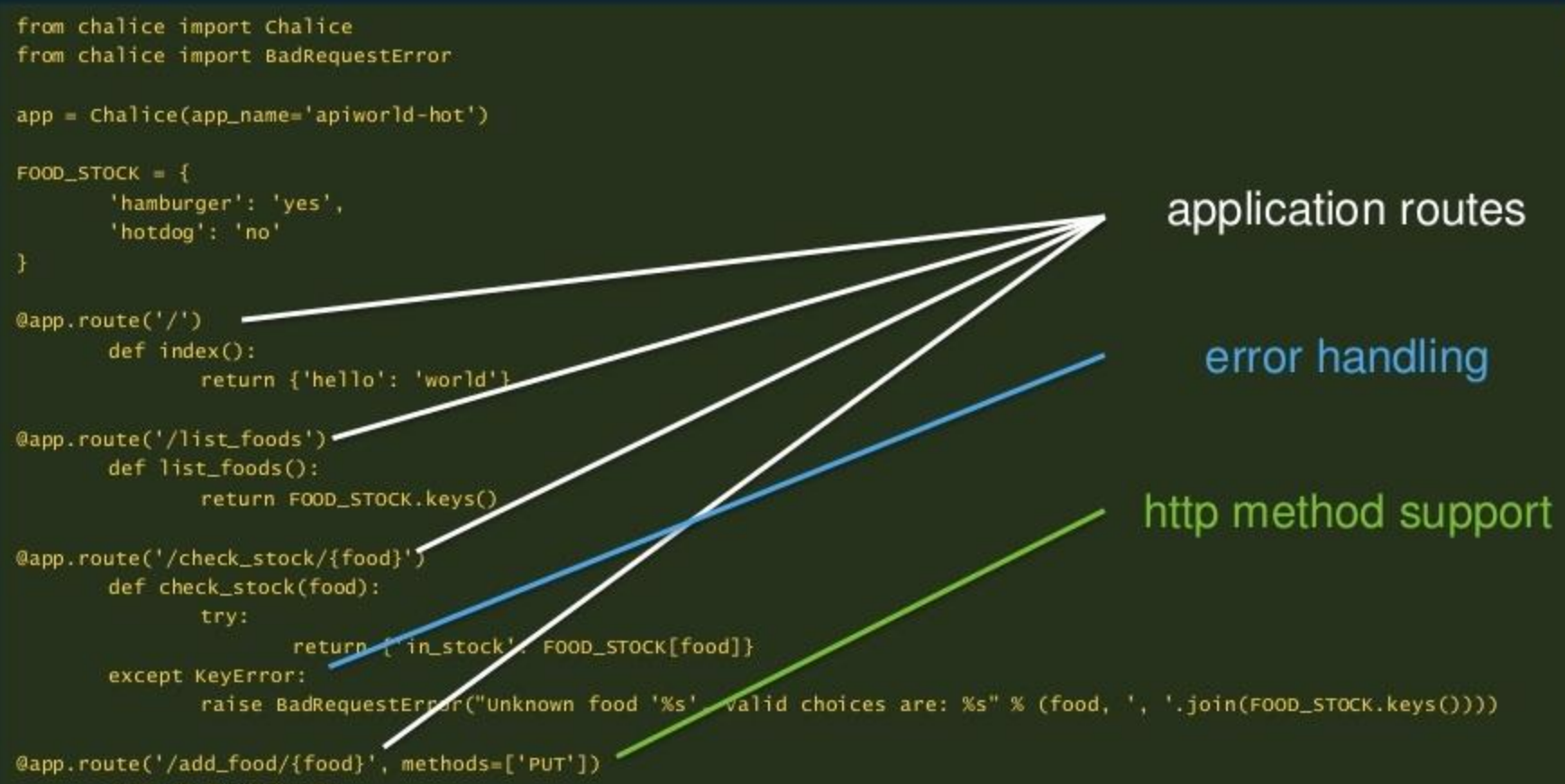
https://chalice.readthedocs.io

app.py:

```
from chalice import Chalice
app = Chalice(app_name="helloworld")

@app.route("/")
def index():
        return {"hello": "world"}
```

$chalice deploy

Chalice – a bit deeper



Chalice – adding Cognito User Pools

Chalic

```
from chalice import Chalice
from chalice import BadRequestError
from chalice import CognitoUserPoolAuthorizer

app = Chalice(app_name='apiworld-hot')

authorizer = CognitoUserPoolAuthorizer( 'MyPool', provider_arns=['arn:aws:cognito:...:userpool/name'])

...
...


@app.route('/list_foods')
        def list_foods():
                return FOOD_STOCK.keys()

@app.route('/check_stock/{food}', methods=['GET'], authorizer=authorizer)
        def check_stock(food):
                try:
                        return {'in_stock': FOOD_STOCK[food]}
        except KeyError:
                raise BadRequestError("Unknown food '%s', valid choices are: %s" % (food, ', '.join(FOOD_STOCK.keys())))

@app.route('/add_food/{food}', methods=['PUT'], authorizer=authorizer)
        def add_food(food):
                return {"value": food}
```

adding authorization

authorization required for certain routes/methods

AWS Serverless Application Model (SAM)

CloudFormation extension optimized for serverless

New serverless resource types: functions, APIs, and tables

Supports anything CloudFormation supports

Open specification (Apache 2.0)

https://github.com/awslabs/serverless-application-model

SAM template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/todo_list.zip
      Handler: index.gethtml
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetHtml:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: ANY

  ListTable:
```

SAM template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/todo_list.zip
      Handler: index.gethtml
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetHtml:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: ANY

  ListTable:
```

Tells CloudFormation this is a S
template it needs to “transform”

Creates a Lambda function with
referenced managed IAM policy
runtime, code at the referenced
location, and handler as defined
Also creates an API Gateway ar
takes care of all
mapping/permissions necessary

Creates a DynamoDB table with

SAM template

26 lines (25 sloc) | 827 Bytes — Raw | Blame | History

```yaml
Transform: 'AWS::Serverless-2016-10-31'
Parameters:
    SamMultipler:
        Description: "SAM multiplier. Make this really big to have a party :)"
        Type: "String"
    OriginUrl:
        Description: "The origin url to allow CORS requests from. This will be the base URL of your static SAM website."
        Type: "String"
Resources:
    GetSAMPartyCount:
        Type: AWS::Serverless::Function
        Properties:
            Handler: index.handler
            Runtime: nodejs4.3
            CodeUri: ./
            Environment:
                Variables:
                    SAM_MULTIPLIER: !Ref SamMultipler
                    ORIGIN_URL: !Ref OriginUrl
            Events:
              GetResource:
                Type: Api
                Properties:
                  Path: /sam
                  Method: get
```

<-THIS BECOMES THIS->

AWS SAM CLI ~~SAM Local~~

Relaunched/GA'd on May 8th!

CLI tool for local building, validating, testing of serverless apps

Works with Lambda functions and "proxy-style" APIs

Response object and function logs available on your local machine

Uses open source docker-lambda images to mimic Lambda's execution environment:

- Emulates timeout, memory limits, runtimes


Deploying your
applications

API Stages

Stages are named links to a deployed version of your API

Recommended for managing API lifecycle

- Dev/test/prod
- Alpha/beta/gamma

Support for parameterized values through *stage variables*

API Gateway Stage Variables

- Stage variables act like environment variables
- Use stage variables to store configuration values
- Stage variables are available in the $context object
- Values are accessible from most fields in API Gateway
  - Lambda function ARN
  - HTTP endpoint
  - Custom authorizer function name
  - Parameter mappings

Stage Variables and Lambda Aliases

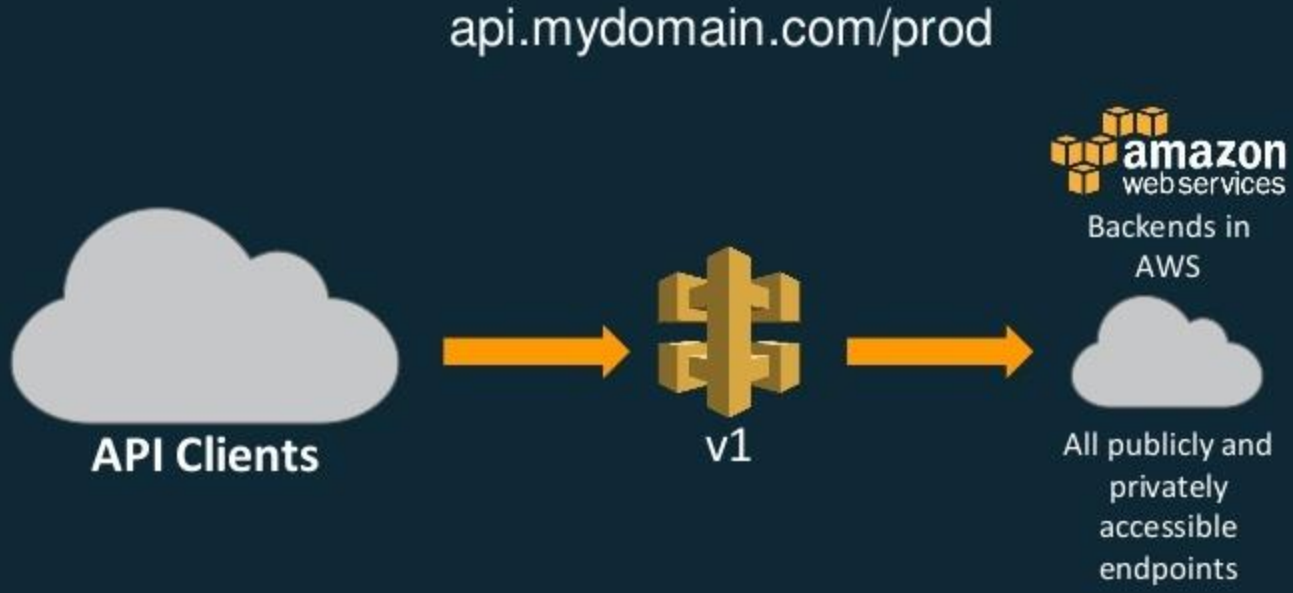
Using Stage Variables in API Gateway together with Lambda function Aliases you can manage a single API configuration and Lambda function for multiple environment stages

Amazon API Gateway Canary Support

Use canary release deployments to gradually roll out new APIs in Amazon API Gateway:

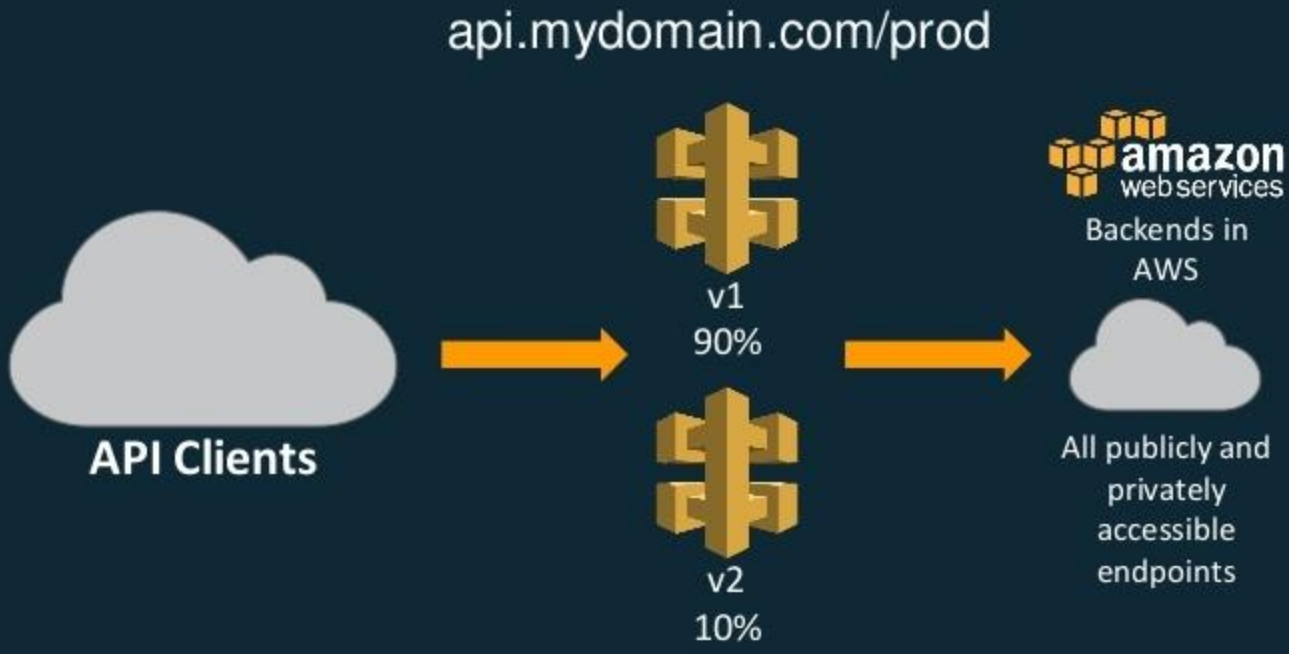
- configure percent of traffic to go to a new stage deployment
- can test stage settings and variables
- API gateway will create additional Amazon CloudWatch Logs group and CloudWatch metrics for the requests handled by the canary deployment API
- To rollback: delete the deployment or set percent of traffic to 0
- Explore new technologies in your API backend:
    - New languages
    - New frameworks
    - Try Lambda in place of other HTTP endpoints!
- Migrate an API from on-premises to AWS via private endpoint integrations in VPC

Amazon API Gateway Canary Support



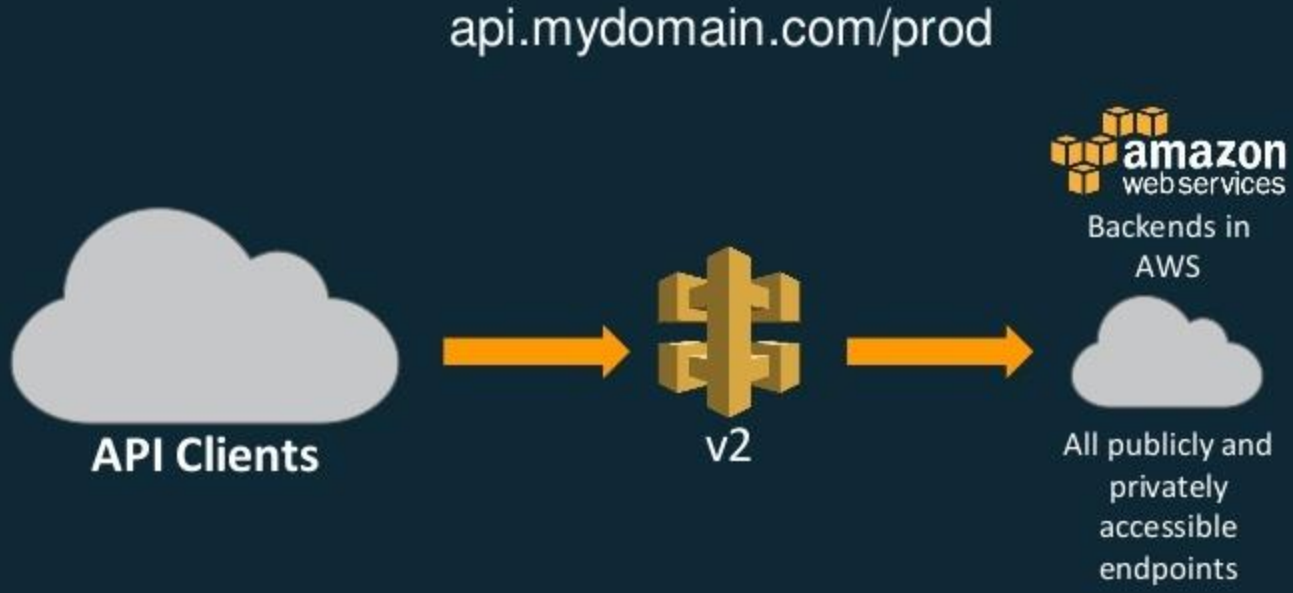
All traffic to currently deployed version

Amazon API Gateway Canary Support

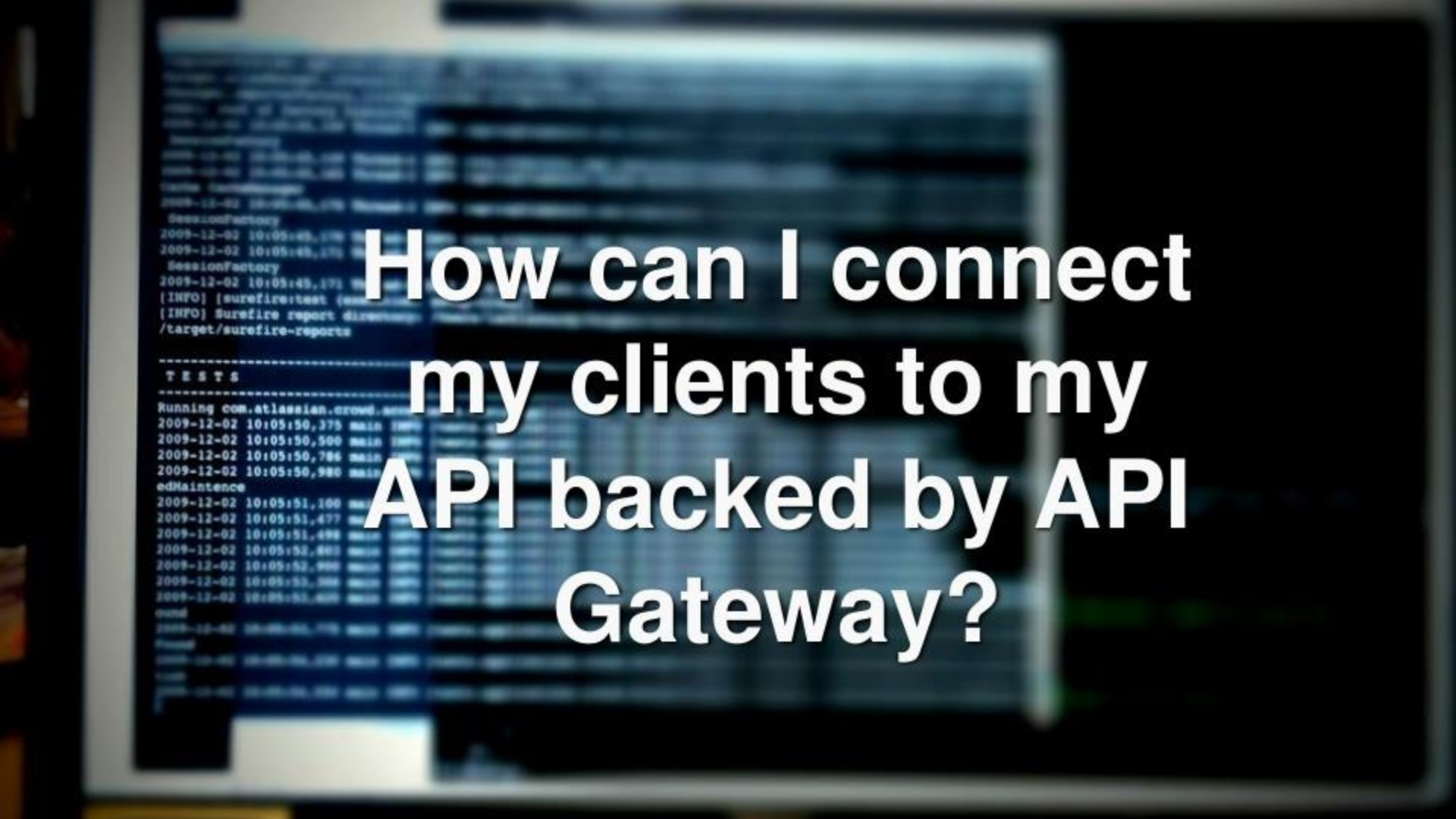


10% traffic to new deployment of stage, rest to previous version

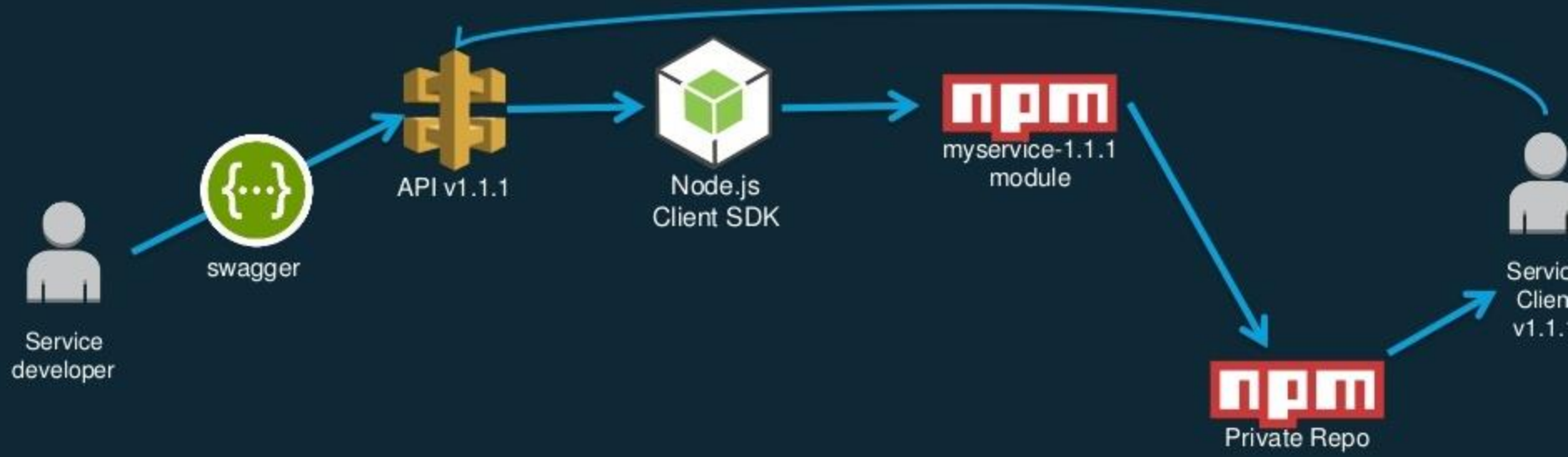
Amazon API Gateway Canary Support



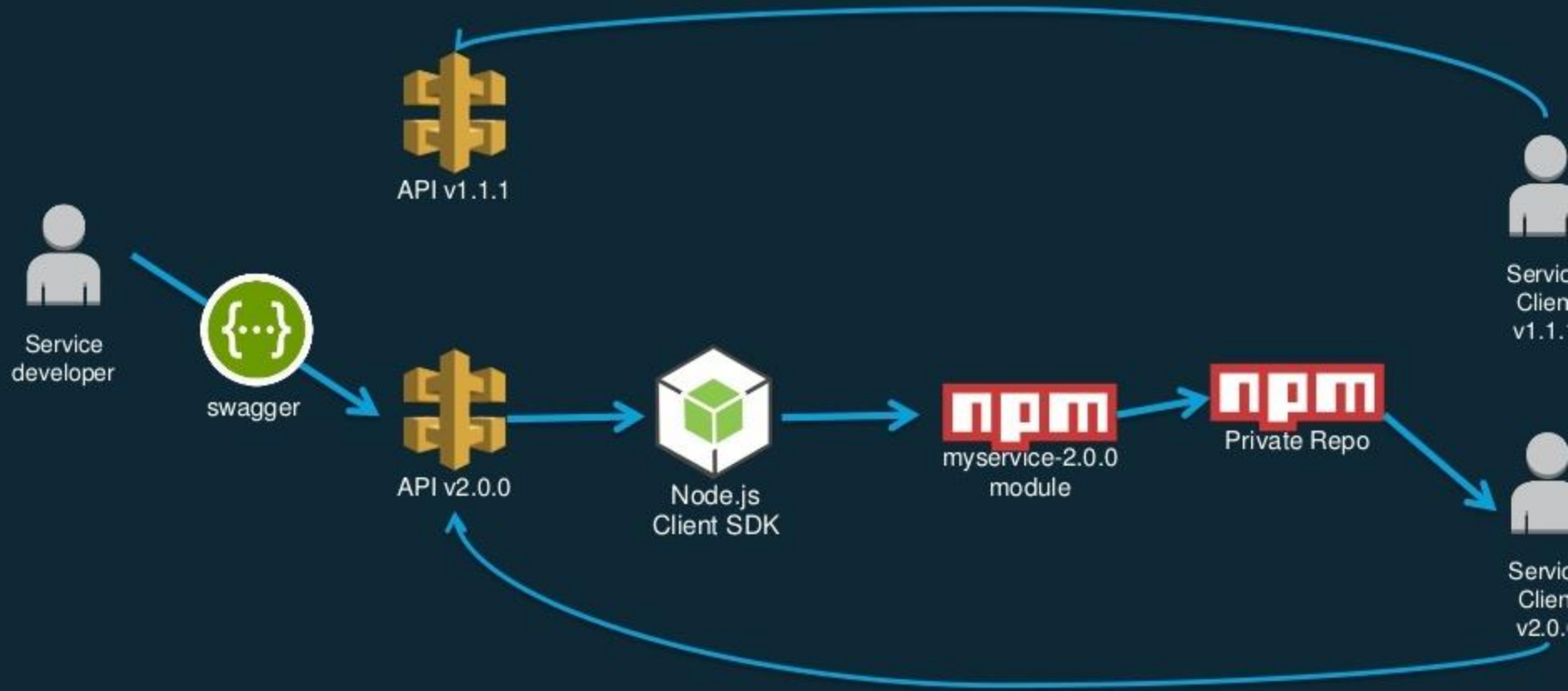
All traffic to new deployed version


How can I connect my clients to my API backed by API Gateway?

SDK work flow:



SDK work flow:



An API based application delivery pipeline:

This pipeline:

- Five Stages
- Builds code artifact
- Three deployed to “Environments”
- Uses SAM/CloudFormation to deploy artifact and other AWS resources
- Has Lambda custom actions for running my own testing functions
- Integrates with a 3rd party tool/service
- Has a manual approval before deploying to production
- Creates a client SDK at the end

FIN, ACK

It's never been easier to build and launch APIs!

Serverless APIs:

- No management of servers
- Pay for what you use and not for idle resources!
- Instantly scale up without turning any knobs or provisioning any resources
- Tooling to get started in minutes with incredibly minimal code needed
- Built in high availability built into multiple places in the application stack
- Authentication and Authorization built into multiple places in the application stack