
AWS
re:Invent

OPN302

# The serverless LAMP stack

Benjamin Smith

Senior Developer Advocate, AWS Serverless

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

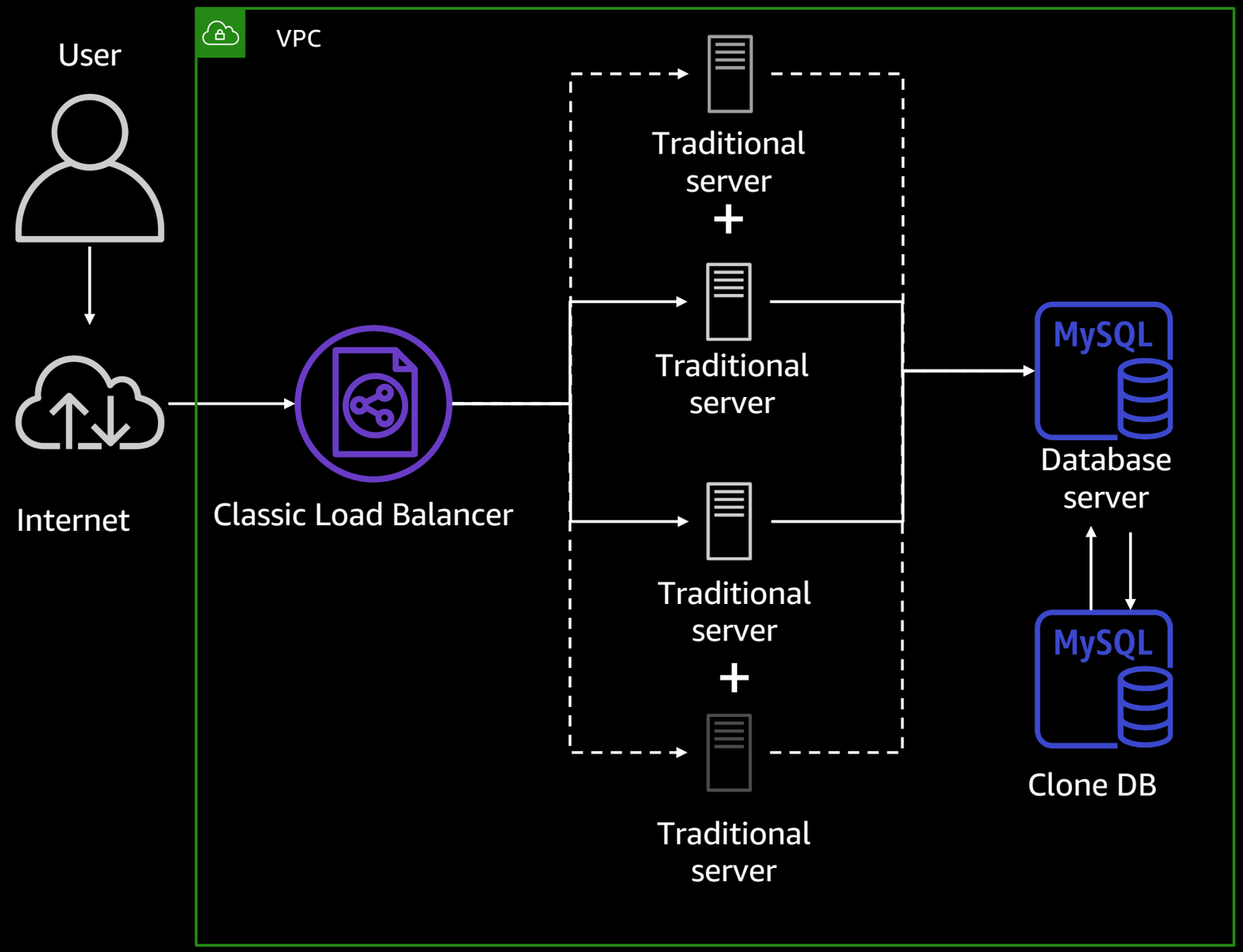
# The challenges with traditional PHP applications



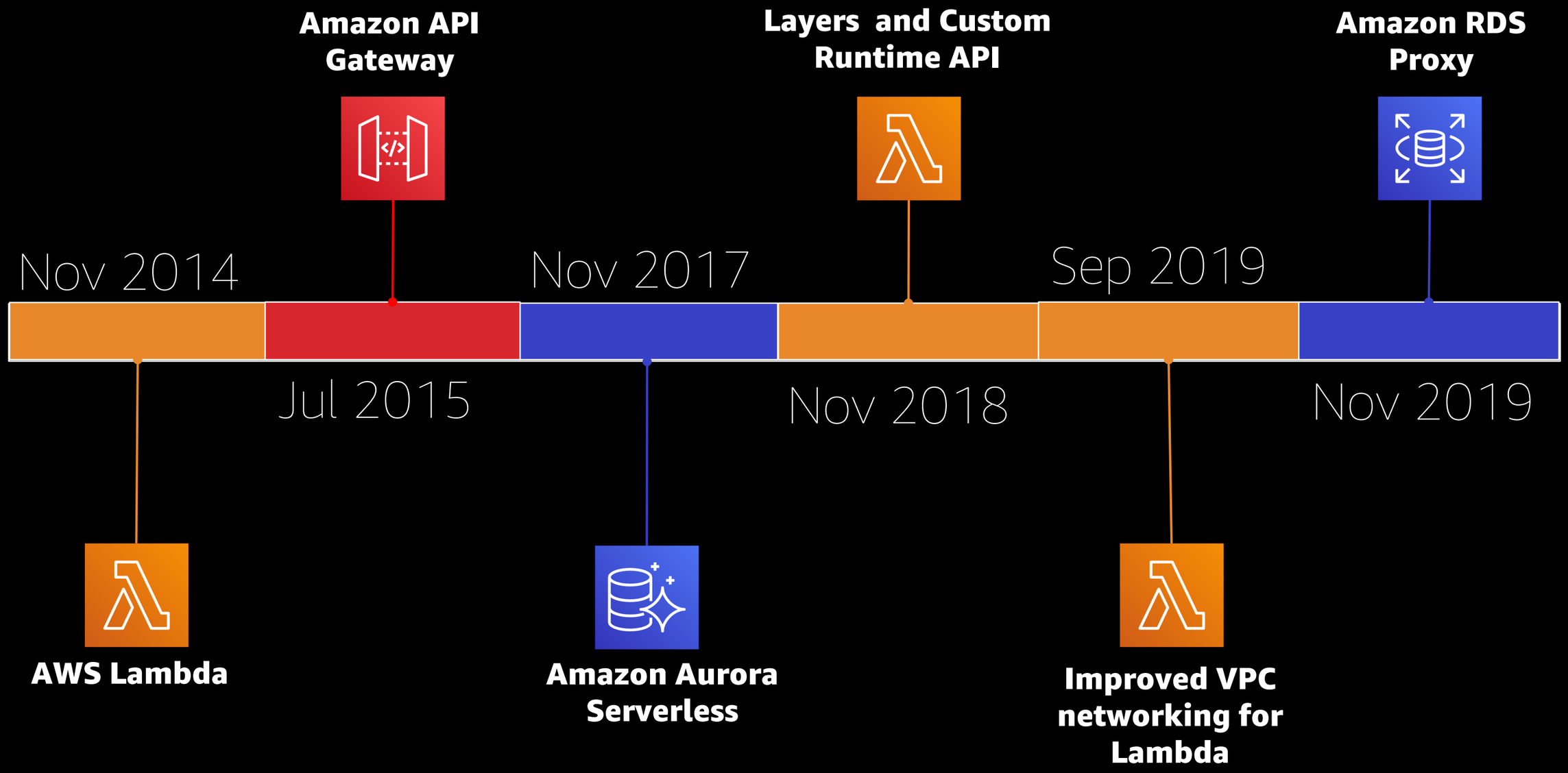


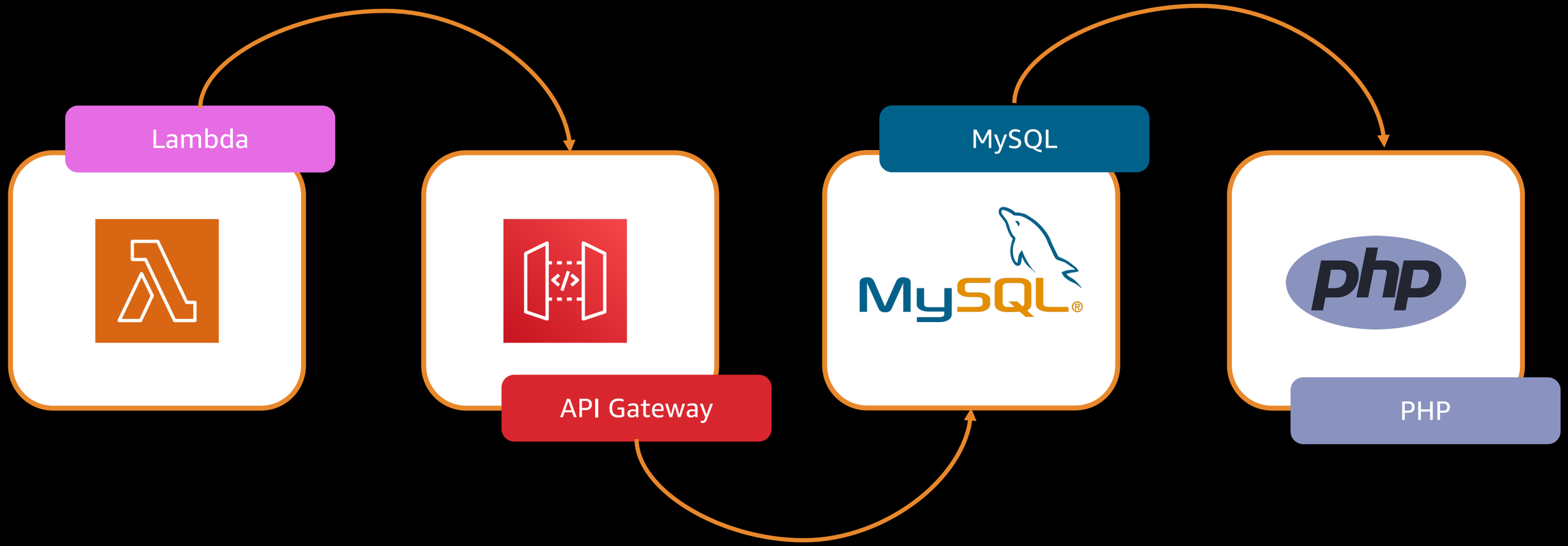
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# Traditional LAMP doesn't scale well

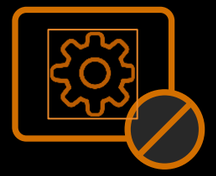


# Why now?


Amazon API Gateway
Layers and Custom Runtime API
Amazon RDS Proxy
Nov 2014
Jul 2015
Nov 2017
Nov 2018
Sep 2019
Nov 2019
AWS Lambda
Amazon Aurora Serverless
Improved VPC networking for Lambda


Lambda
API Gateway
MySQL
MySQL
php
PHP

# What is serverless?

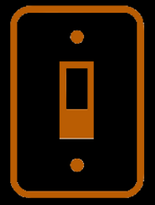


No infrastructure provisioning, no management



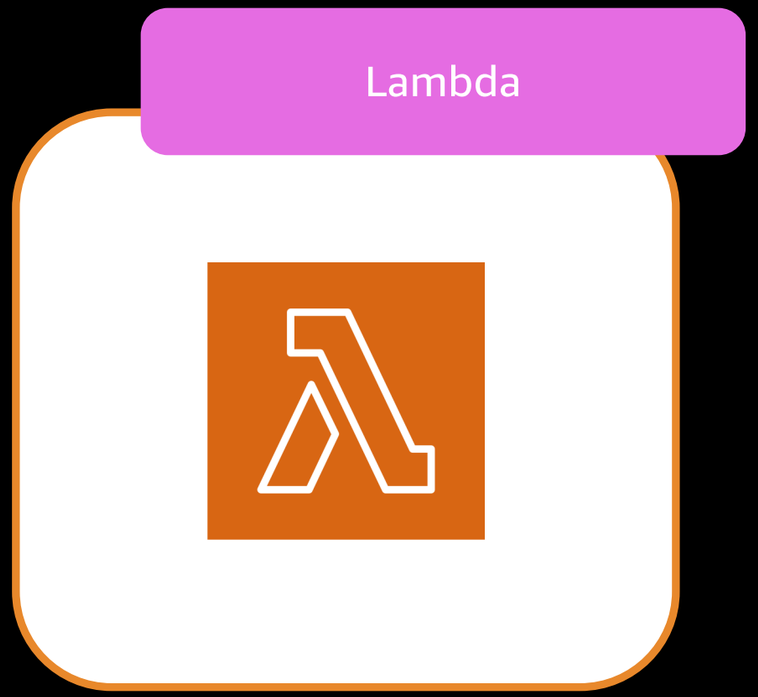
Automatic scaling

Pay for value



Highly available and secure






# Lambda

A

M

P

# Serverless applications

Event source

Function

Services (anything)







Changes in data state



Requests to endpoints



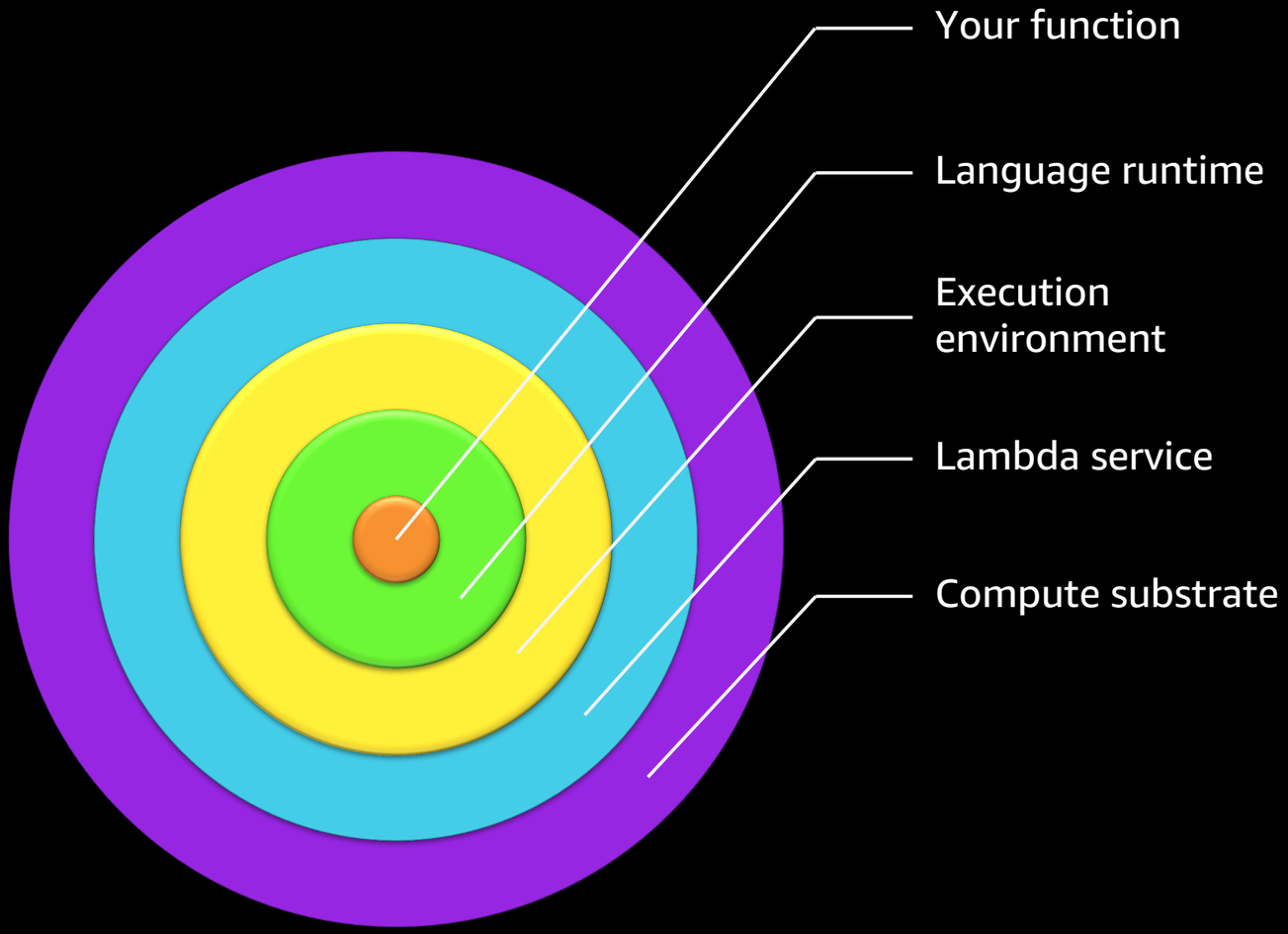
Changes in resource state

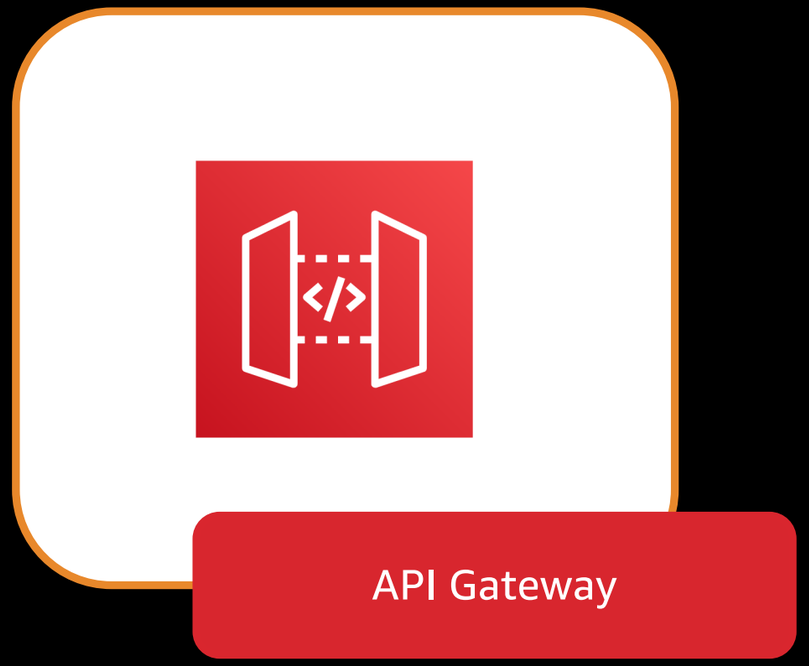


Node.js
Python
Java
Go
Ruby
.Net (C#/Powershell)
Custom Runtime API



# Anatomy of an AWS Lambda function






L
API Gateway
M
P

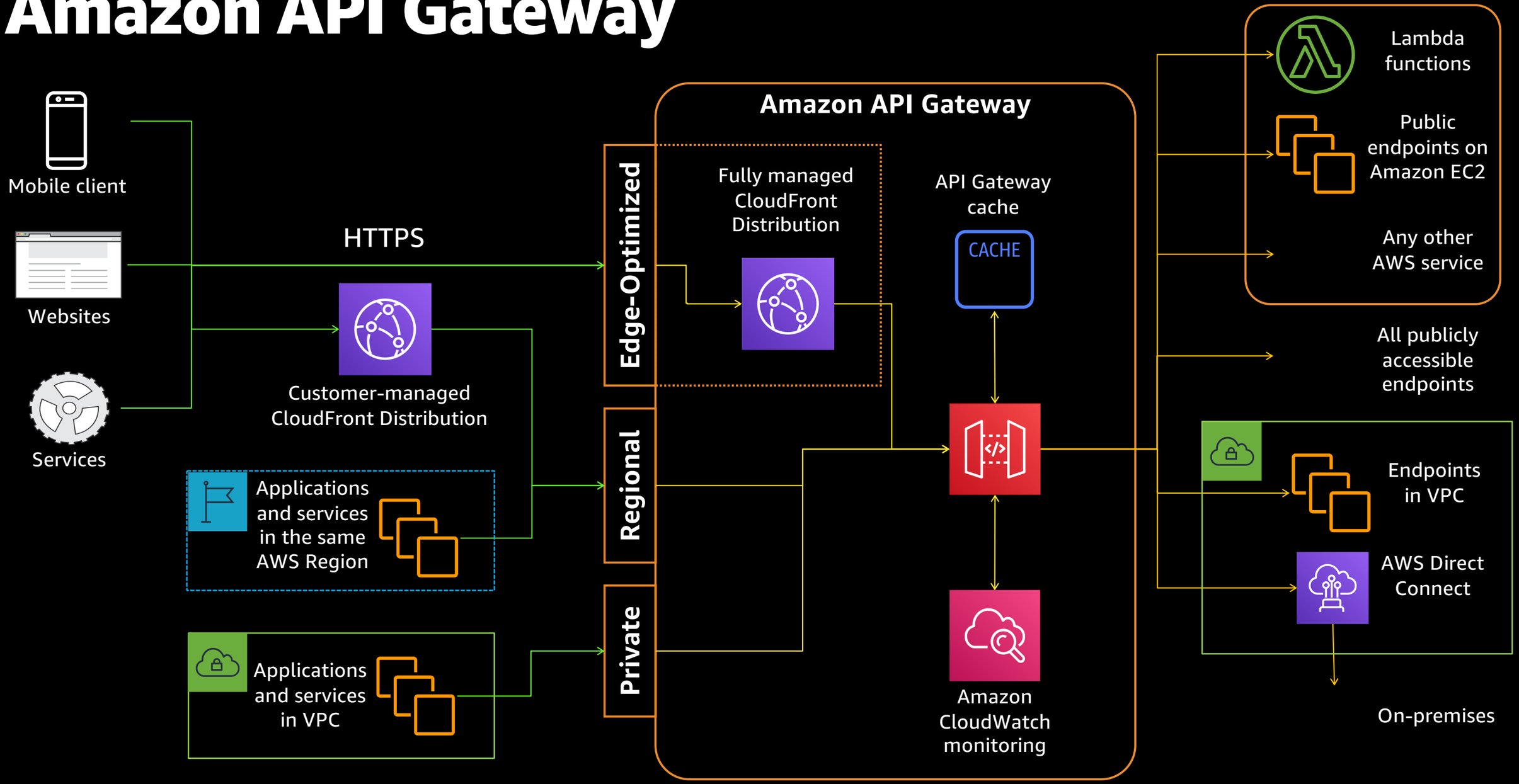
# APIs are the front door to your backend services



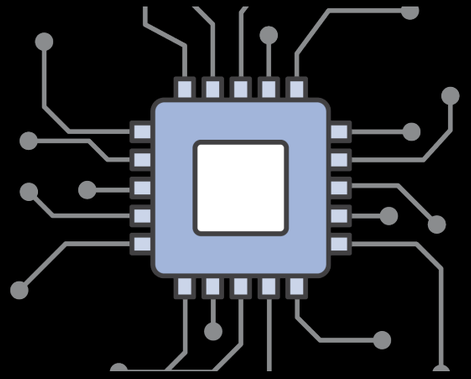


© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# Amazon API Gateway



# Amazon API Gateway



Create a unified API front-end for multiple microservices



DDoS protection and throttling for your backend applications



Authenticate and authorize requests to a backend



Throttle, meter, and monetize API usage by third-party developers

MySQL




L
A
MySQL + Amazon Aurora
Amazon RDS Proxy
P

# Amazon Aurora

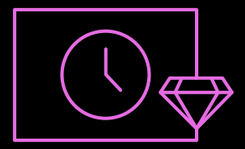
- MySQL- and PostgreSQL-compatible relational database built for the cloud
- Performance and availability of commercial-grade databases at 1/10$^{th}$ the cost

---



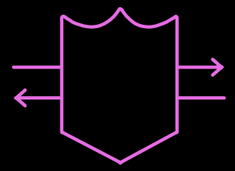
**Performance and scalability**

5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to 15 read replicas



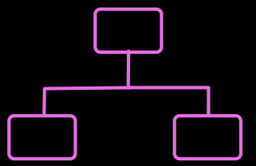
**Availability and durability**

Fault-tolerant, self-healing storage; 6 copies of data across 3 Availability Zones; nearly continuous backup to S3



**Highly secure**

Network isolation, encryption at rest/transit



**Fully managed**

Managed by RDS: No hardware provisioning, software patching, setup, configuration, or backups

---

# Amazon Aurora

- MySQL- and PostgreSQL-compatible relational database built for the cloud
- Performance and availability of commercial-grade databases at 1/10th the cost

70,369 Gigabytes

**and scalability**

5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to 15 read replicas

**and durability**

Fault-tolerant, self-healing storage; 6 copies of data across 3 Availability Zones; nearly continuous backup to S3

**secure**

Network isolation, encryption at rest/transit

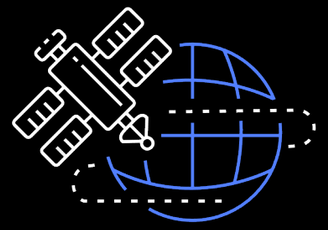
**managed**

Managed by RDS: No hardware provisioning, software patching, setup, configuration, or backups

# Amazon RDS Proxy

- Fully managed, highly available database proxy feature for Amazon RDS
- Pools and shares database connections to make applications more scalable, resilient to database failures, and secure

---



Pool and share DB connections for improved application scaling

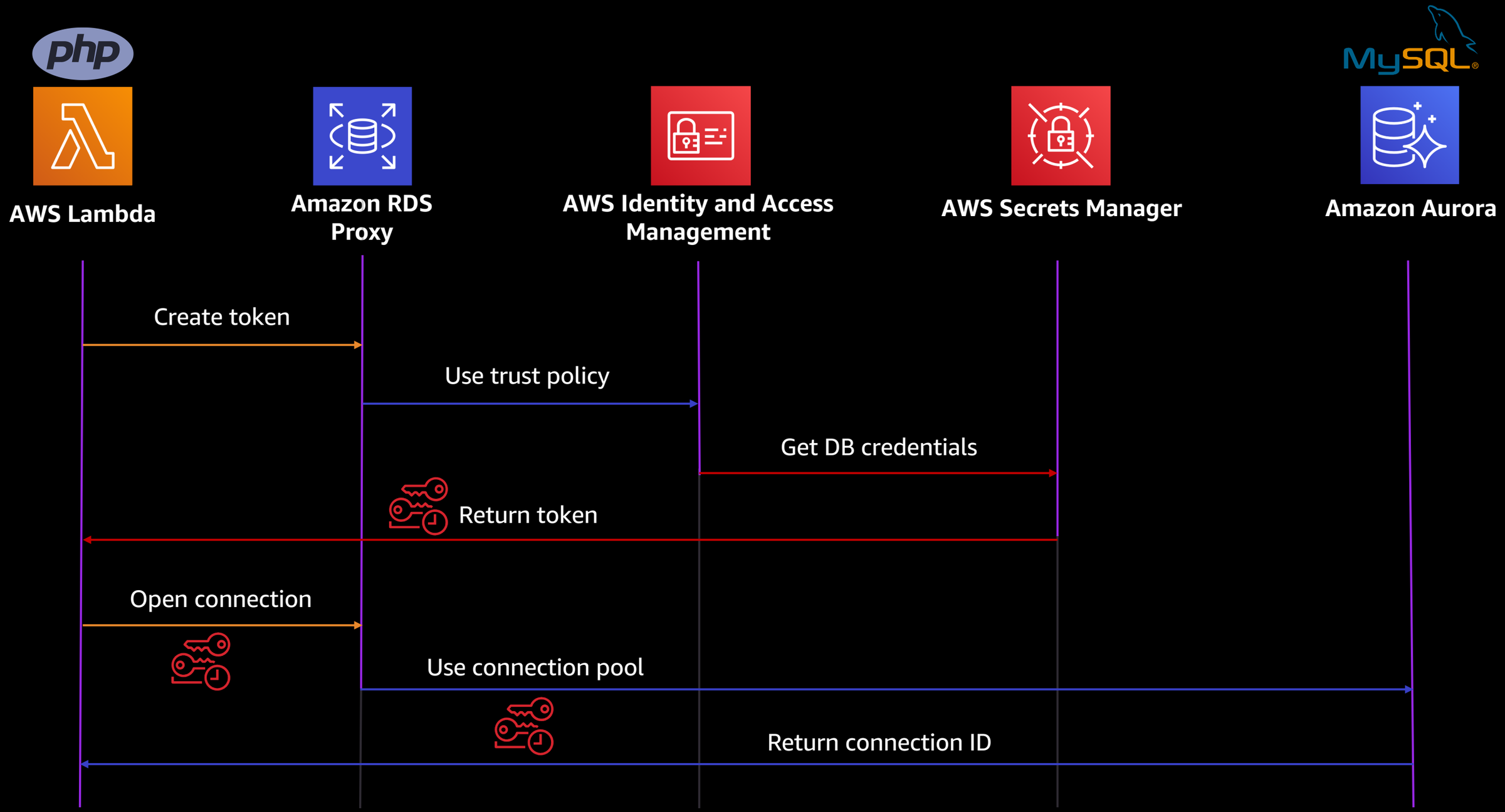


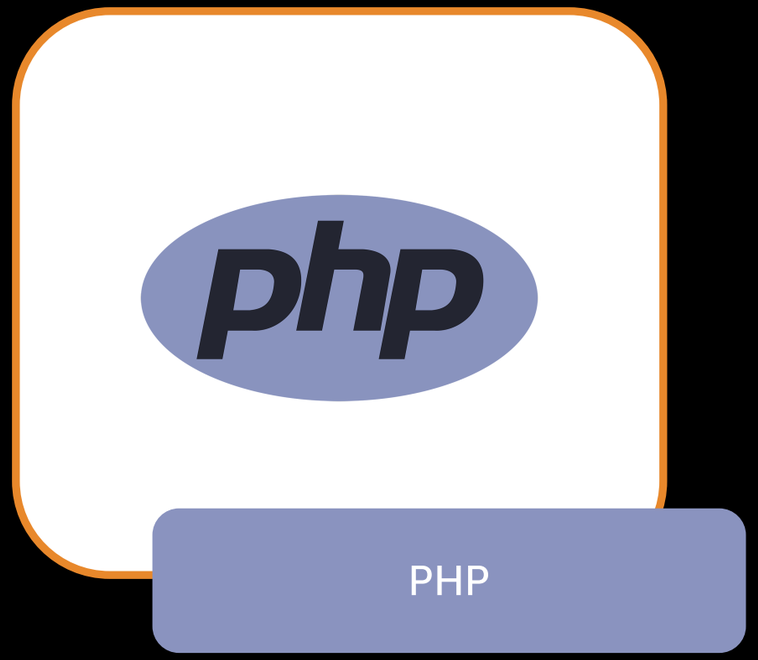
Increase application availability and reduce DB failover times



Manage application data security with DB access controls

---


AWS Lambda
Amazon RDS Proxy
AWS Identity and Access Management
AWS Secrets Manager
Amazon Aurora
Create token
Use trust policy
Get DB credentials
Return token
Open connection
Use connection pool
Return connection ID




L
A
M
PHP on AWS Lambda

# Approximately 80% of the web runs on PHP

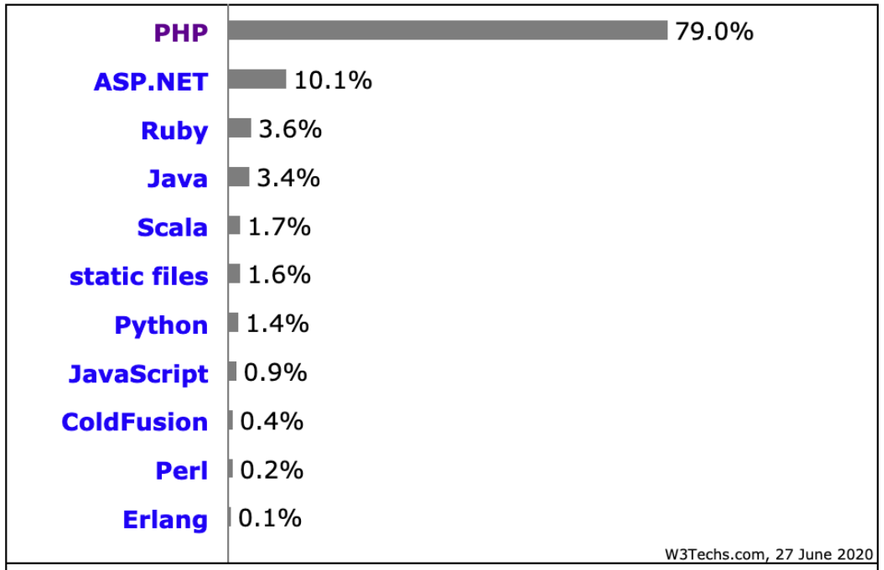
## Usage statistics of server-side programming languages for websites

This diagram shows the percentages of websites using various server-side programming languages. See technologies overview for explanations on the methodologies used in the surveys. Our reports are updated daily.

Request an extensive server-side programming languages market report.

**Learn more**

How to read the diagram:
PHP is used by 79% of all the websites whose server-side programming language we know.




Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

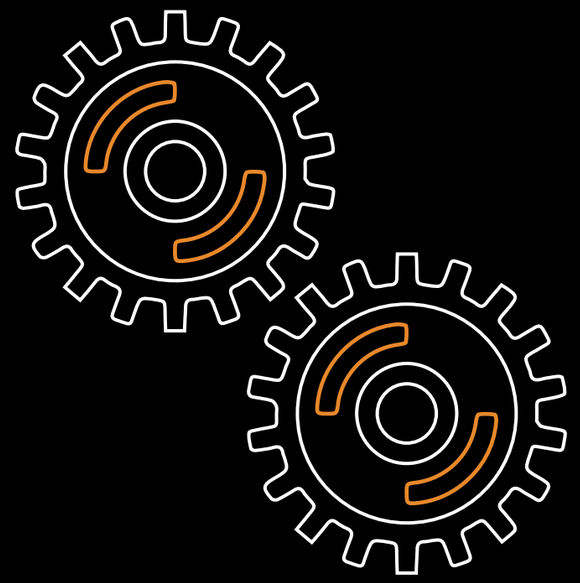
Source: https://w3techs.com/technologies/overview/programming_language

# AWS Lambda custom runtimes

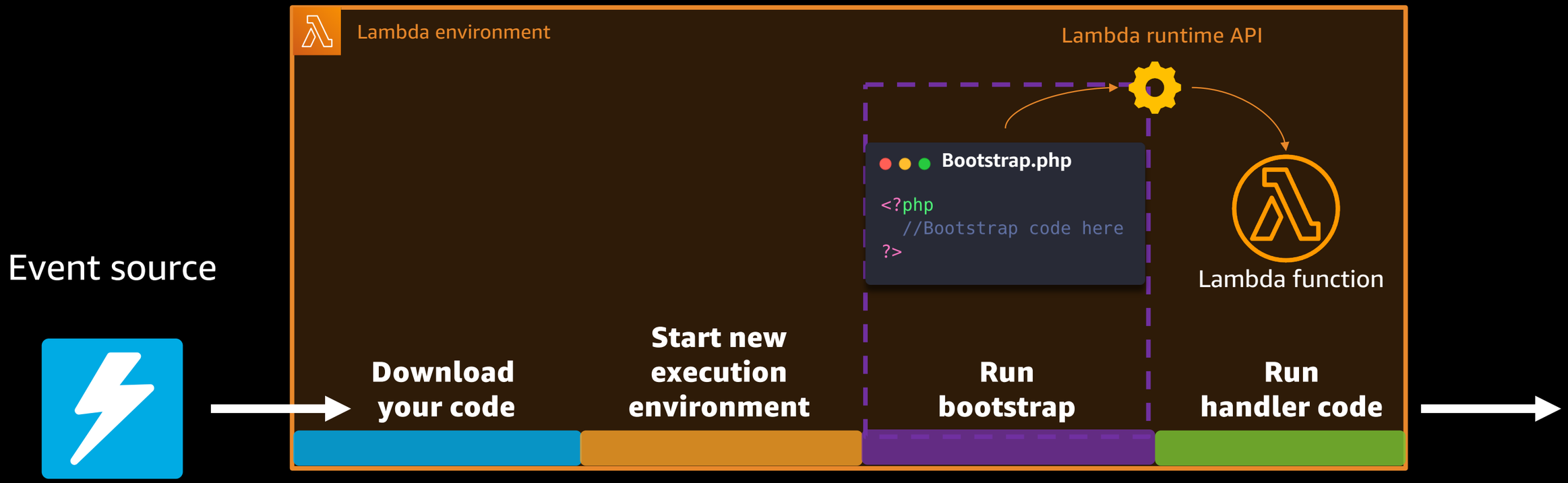
Bring any Linux-compatible language runtime

Powered by the **Runtime API**

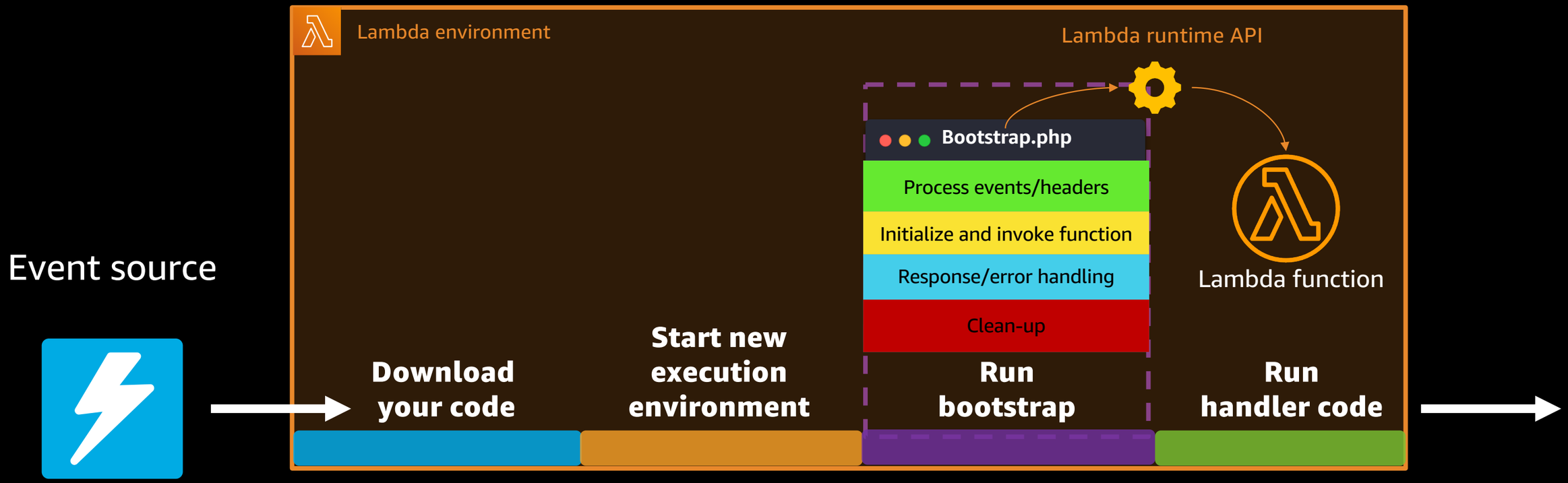
Custom runtimes distributed as layers



# The bootstrap executable acts as a bridge between the runtime HTTP API and the function to be invoked



# Bootstrap needs to manage response/error handling, context, creation, and function execution

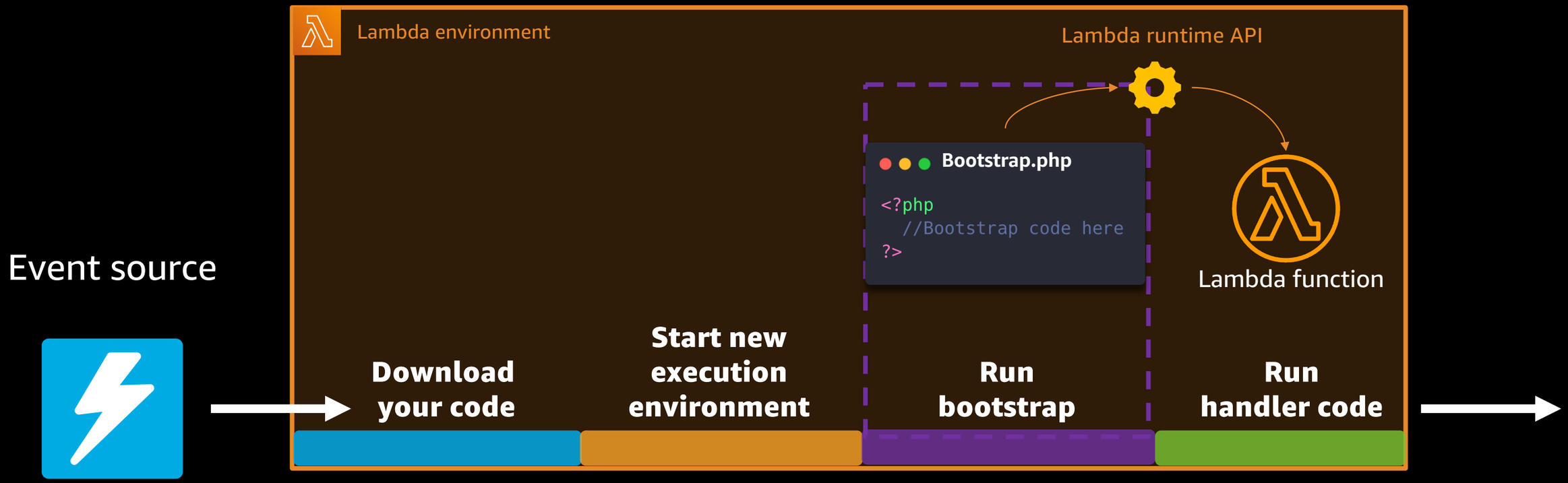


# Information on the bootstrap file and the function handler are shared as environment variables

`$_ENV['AWS_LAMBDA_RUNTIME_API']` ← The host and port of the **runtime API**

`$_ENV['_HANDLER']` ← The handler location configured on the function

`$_ENV['LAMBDA_TASK_ROOT']` ← The directory that contains the function code



# Bootstrap file

```php
#!/opt/bin/php
<?php

// This invokes Composer's autoloader so that we'll be able to use Guzzle and any other 3rd party libraries we
need.
require __DIR__ . '/vendor/autoload.php';

// This is the request processing loop. Barring unrecoverable failure, this loop runs until the environment shuts
down.
do {
    // Ask the runtime API for a request to handle.
    $request = getNextRequest();

    // Obtain the function name from the _HANDLER environment variable and ensure the functions code is available.
    $handlerFunction = array_slice(explode('.', $_ENV['_HANDLER']), -1)[0];
    require_once $_ENV['LAMBDA_TASK_ROOT'] . '/src/' . $handlerFunction . '.php';

    // Execute the desired function and obtain the response.
    $response = $handlerFunction($request['payload']);

    // Submit the response back to the runtime API.
    sendResponse($request['invocationId'], $response);
} while (true);

function getNextRequest()
{
    $client = new \GuzzleHttp\Client();
    $response = $client->get('http://' . $_ENV['AWS_LAMBDA_RUNTIME_API'] . '/2018-06-01/runtime/invocation/next');

    return [
      'invocationId' => $response->getHeader('Lambda-Runtime-Aws-Request-Id')[0],
      'payload' => json_decode((string) $response->getBody(), true),
```

Process **next** request

**Handler function** name from **$_ENV**

**Return** response/error

Interact with **runtime API**

# PHP versions, modules, and package dependencies



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# PHP versions and extensions

```
mkdir -p ~/environment/php-7-bin
curl -sL https://github.com/php/php-src/archive/php-7.3.0.tar.gz \
| tar -xvz
cd php-src-php-7.3.0
```

**Compile** the required version of **PHP** in an **Amazon Linux environment**

```
./buildconf --force
./configure --prefix=/home/ec2-user/environment/php-7-bin/ \
--with-openssl=/usr/local/ssl --with-curl --with-zlib --with-mysqli
make install
```

**Include any extensions** with the PHP configuration

**Package** the **PHP binary** and **bootstrap** file together into a file

```
zip -r runtime.zip bin bootstrap
```

**Publish** the file to a **layer**

```
aws lambda publish-layer-version \
    --layer-name PHP-example-runtime \
    --zip-file fileb://runtime.zip \
    --region eu-west-1
```

Add the layer to your Lambda function

```
Resources:
  Hello:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Runtime: provided
      Layers:
        - 'arn:aws:lambda:us-east-1:209497400698:layer:php-73:7'
```

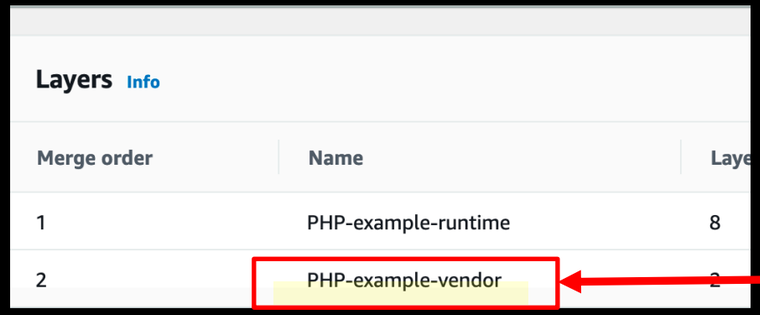
# Libraries and dependencies

```
curl -sS https://getcomposer.org/installer | ./bin/php \
./bin/php composer.phar \
require guzzlehttp/guzzle aws/aws-sdk-php \
zip -r vendor.zip vendor/
```

Install **Composer**; install any **additional libraries**; package the dependencies into a binary

```
aws lambda publish-layer-version \
    --layer-name PHP-example-vendor \
    --zip-file fileb://vendor.zip \
    --region eu-west-1
```

Use the **AWS CLI** to publish a **layer** from the binary




Add the vendor layer to your Lambda function

# Libraries and dependencies in your lambda code

```php
<?php

use Aws\Exception\AwsException;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;


function index($data){
  // Use the default credential provider
  $provider = CredentialProvider::defaultProvider();
  $RdsAuthGenerator = new Aws\Rds\AuthTokenGenerator($provider);
  $token = $RdsAuthGenerator->createToken(getenv('proxyHost').":3306",getenv('region'), getenv('username'));

  $mysqli = mysqli_init();
  $mysqli->real_connect(getenv('proxyHost'), getenv('username'), $token, 'mysql',3306 , NULL, MYSQLI_CLIENT_SSL);

  if ($mysqli->connect_errno) {
      //code to handle and return error
  }
  $mysqli -> close();
  return json_encode($res);
}
```

**Use statement** imports classes before the function handler

Use the imported library inside the function handler

# Libraries and dependencies in your lambda code

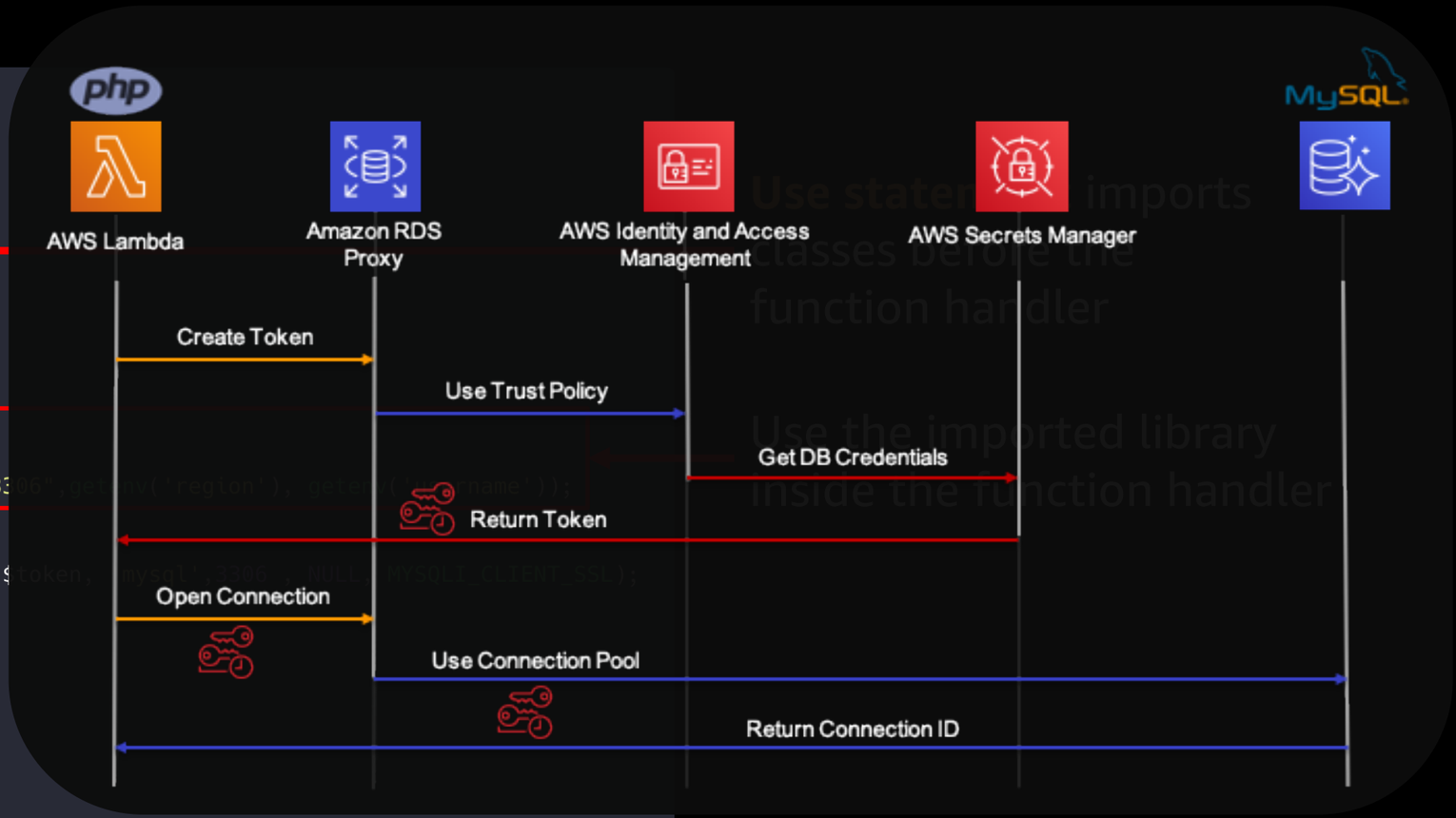
```php
<?php

use Aws\Exception\AwsException;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;


function index($data){
  // Use the default credential provider
  $provider = CredentialProvider::defaultProvider();
  $RdsAuthGenerator = new Aws\Rds\AuthTokenGenerator($provider);
  $token = $RdsAuthGenerator->createToken(getenv('proxyHost').":3

  $mysqli = mysqli_init();
  $mysqli->real_connect(getenv('proxyHost'), getenv('username'), $

  if ($mysqli->connect_errno) {
      //code to handle and return error
  }
  $mysqli -> close();
  return json_encode($res);
}
```



# Open-source runtimes for PHP




**Open source**, production-ready PHP runtimes for AWS Lambda



```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  Hello:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Runtime: provided
      Layers:
          - 'arn:aws:lambda:us-east-1:209497400698:layer:php-73:7'
```

**Deploy** as a layer with AWS Serverless Application Model (**AWS SAM**) or Serverless Framework

# Bref's PHP-FPM layer

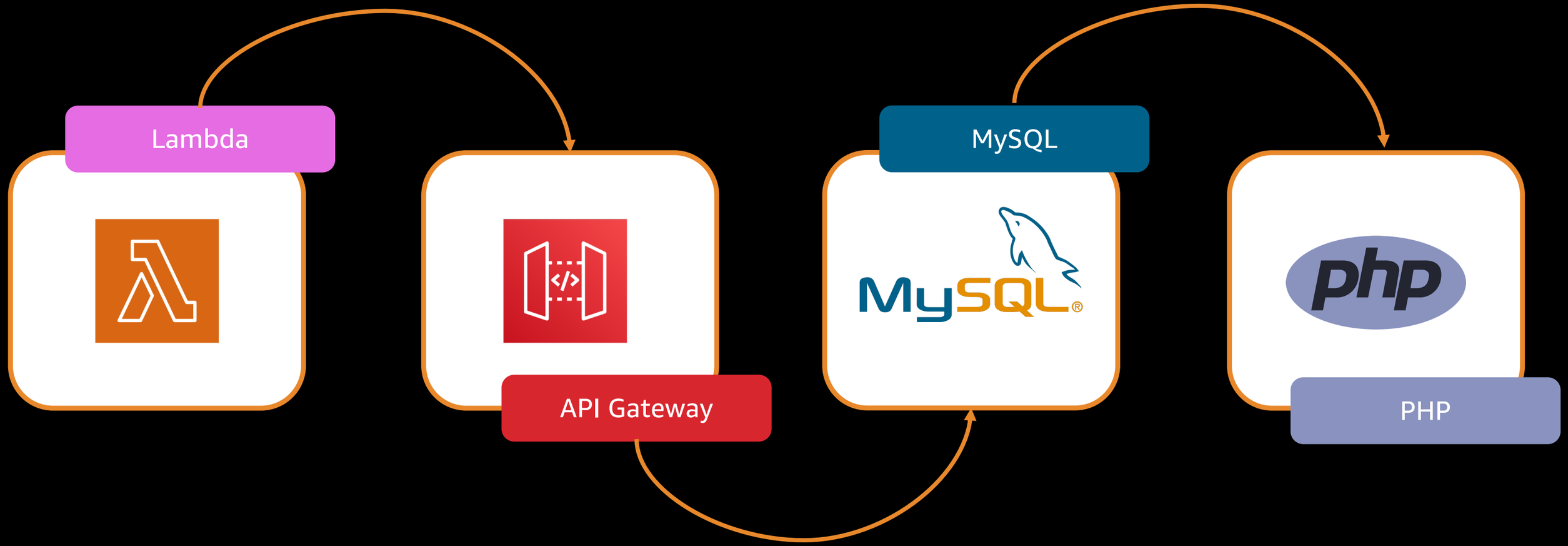
PHP-FPM is a server implementing the FastCGI protocol, developed by the PHP core team

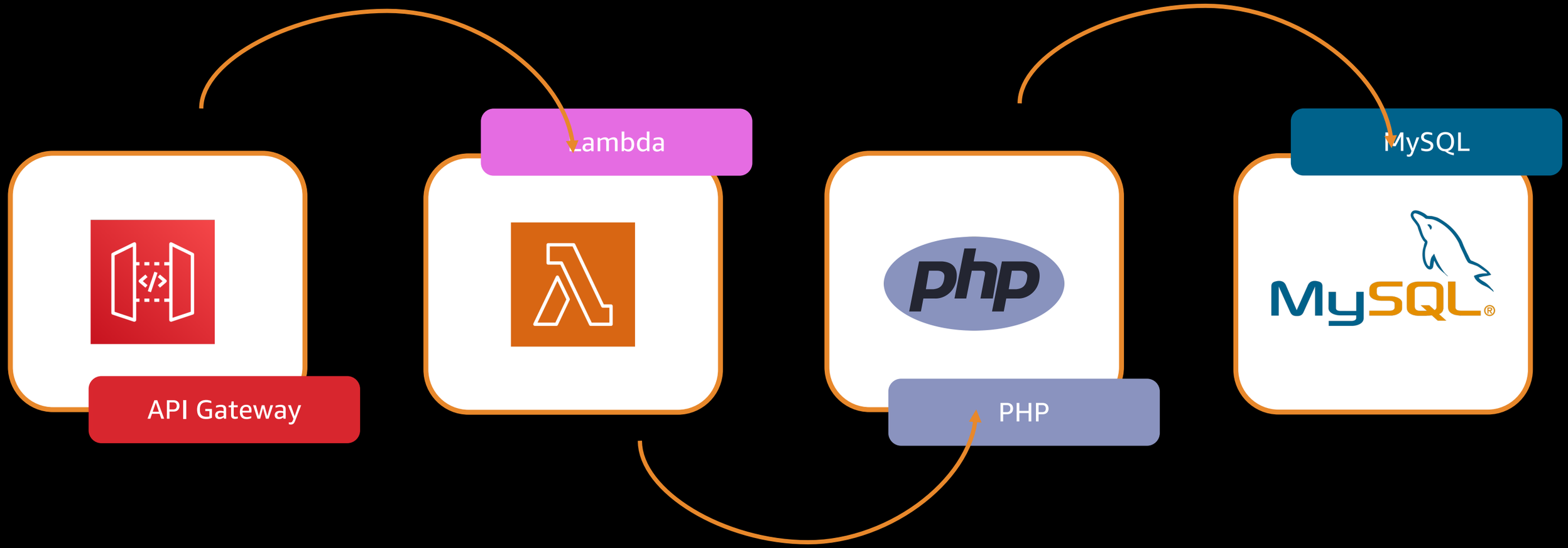
It is traditionally used with HTTP servers like Apache or NGINX to manage inbound HTTP requests for PHP applications at high loads

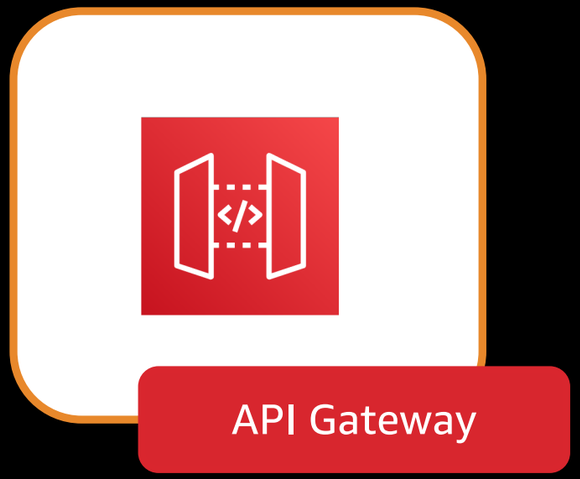


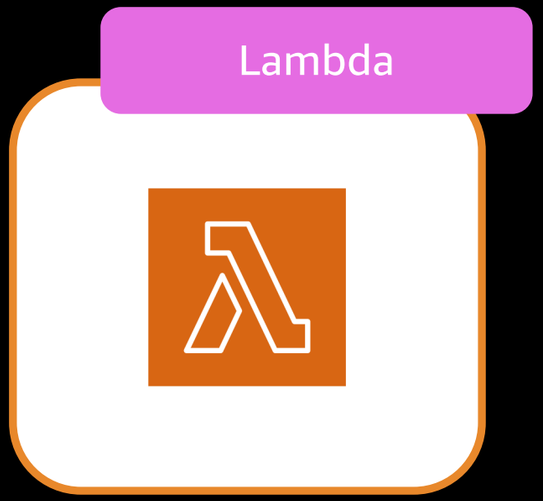

Bref's implementation of PHP-FPM handles:

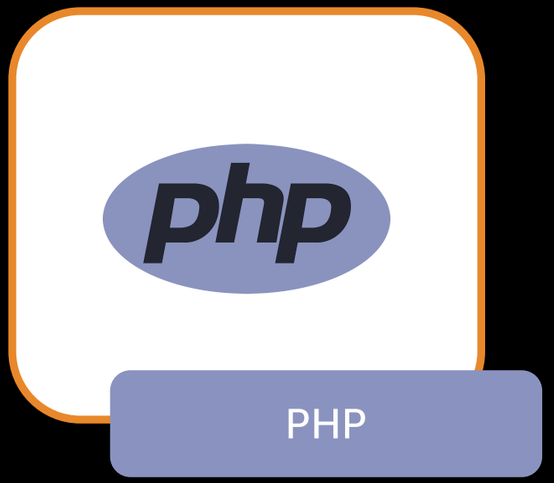
- $_GET
- Single process per HTTP request
- Returning HTTP responses
- Memory cache optimizations
- Database persistent connections
- $_GLOBALS
- OPcode cache optimizations
- $_REQUEST
- $_POST
- $_ENV
- $_SERVER


Lambda
API Gateway
MySQL
MySQL
php
PHP

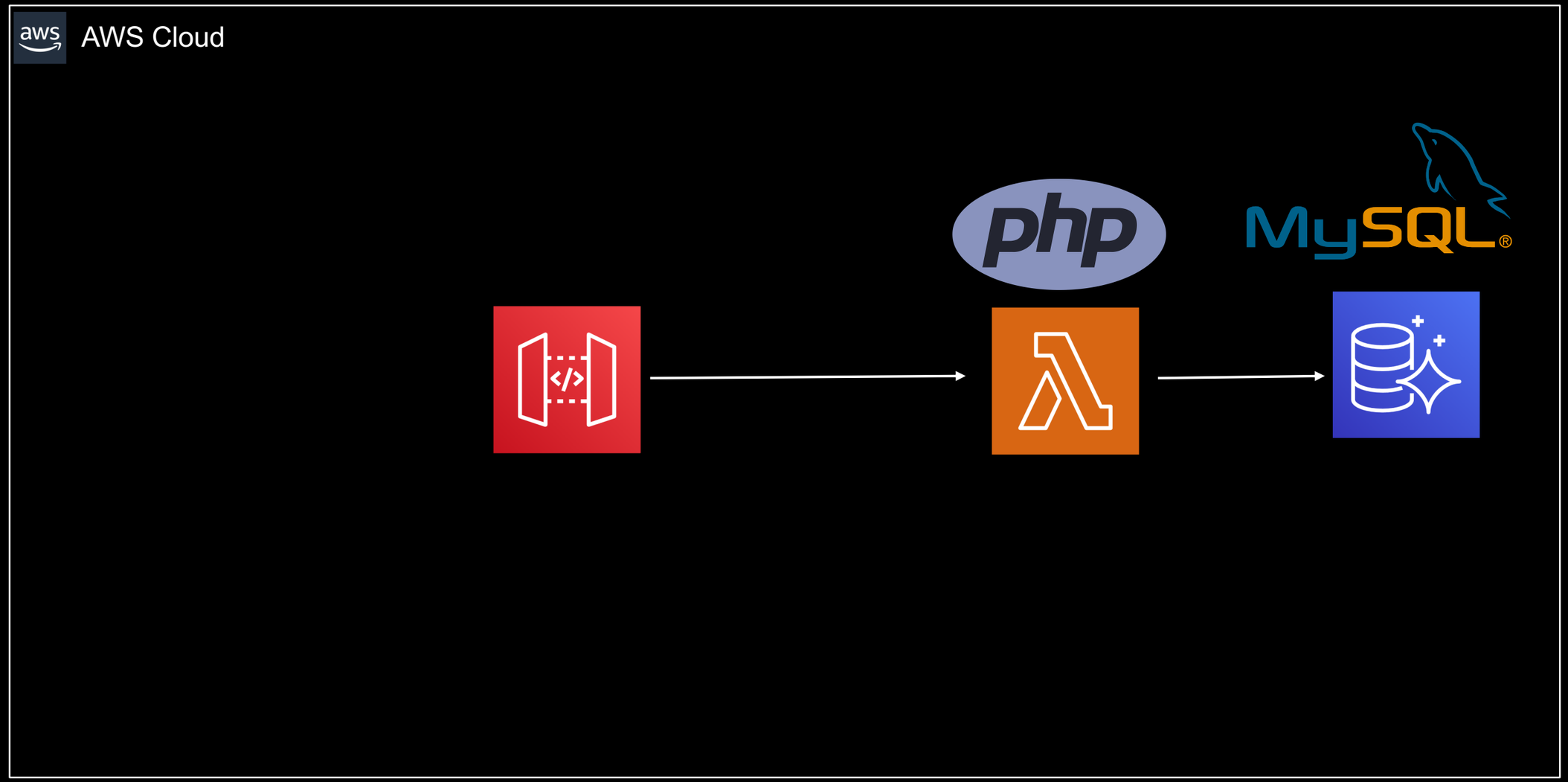

Lambda
MySQL
API Gateway
PHP


API Gateway

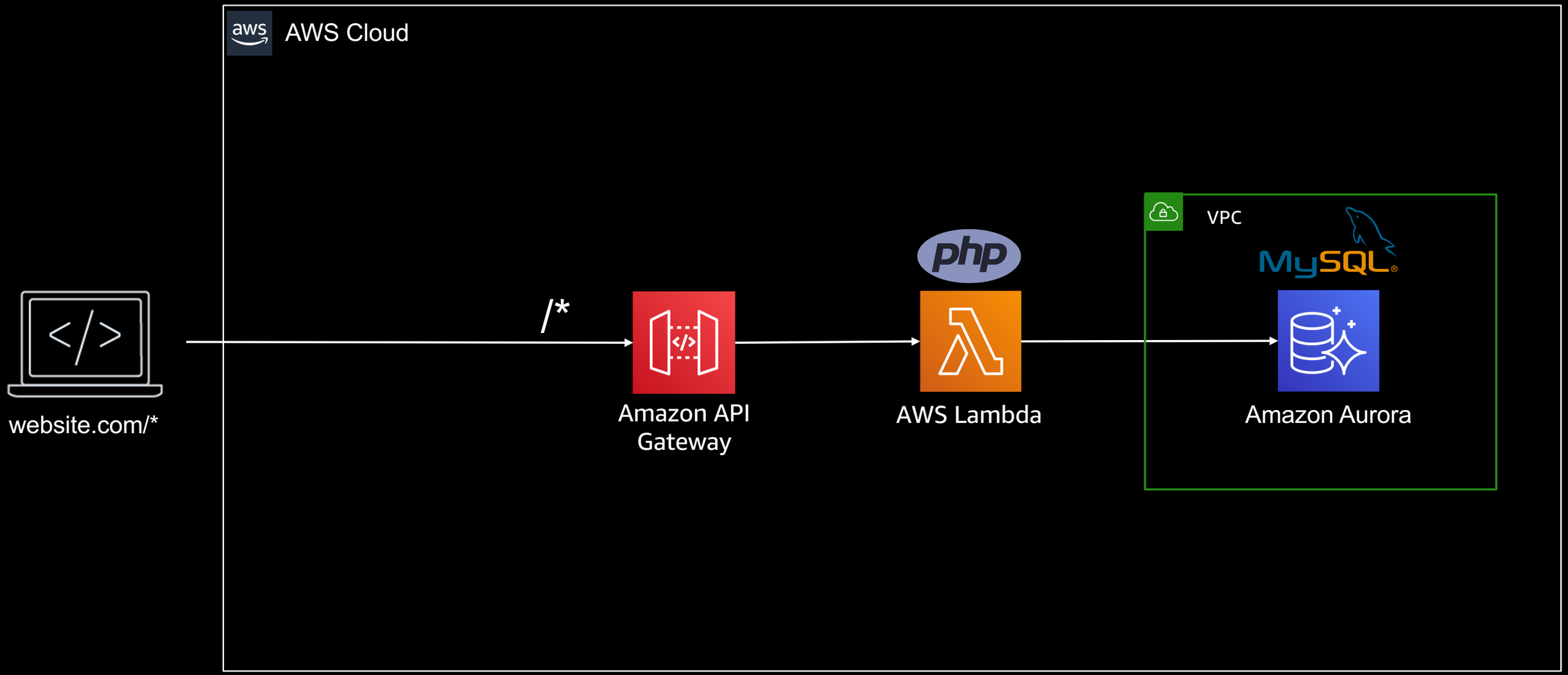

Lambda


php
PHP

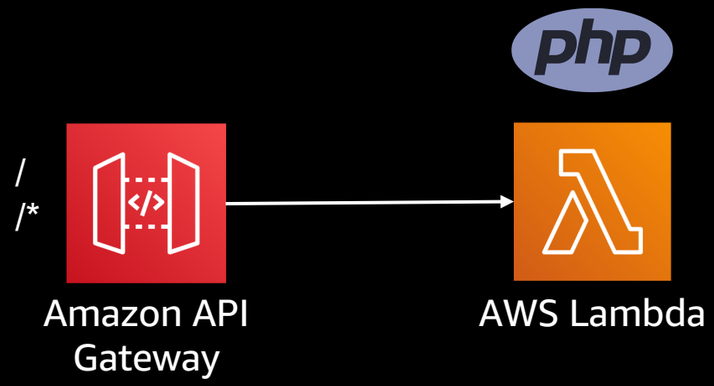

MySQL
MySQL®


AWS Cloud
php
MySQL

# API Gateway to catch all requests



# API Gateway routing




## API Gateway rule config

```
events:
  -   http: 'ANY /'
  -   http: 'ANY /{proxy+}'
```

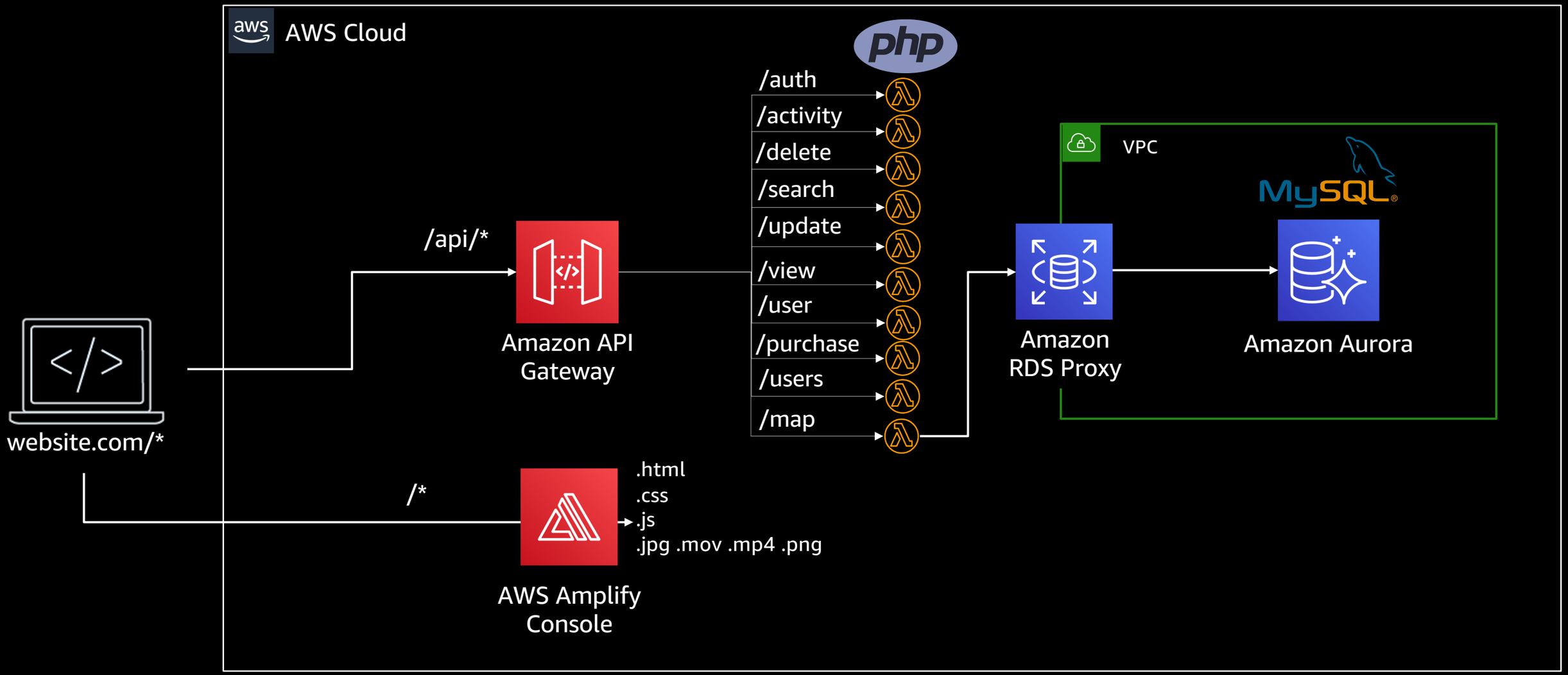
## apache2 config file

```
Alias /yourdir /var/www/html/yourdir/public/
<Directory "/var/www/html/yourdir/public">
        AllowOverride All
        Order allow,deny
        allow from all
</Directory>
```

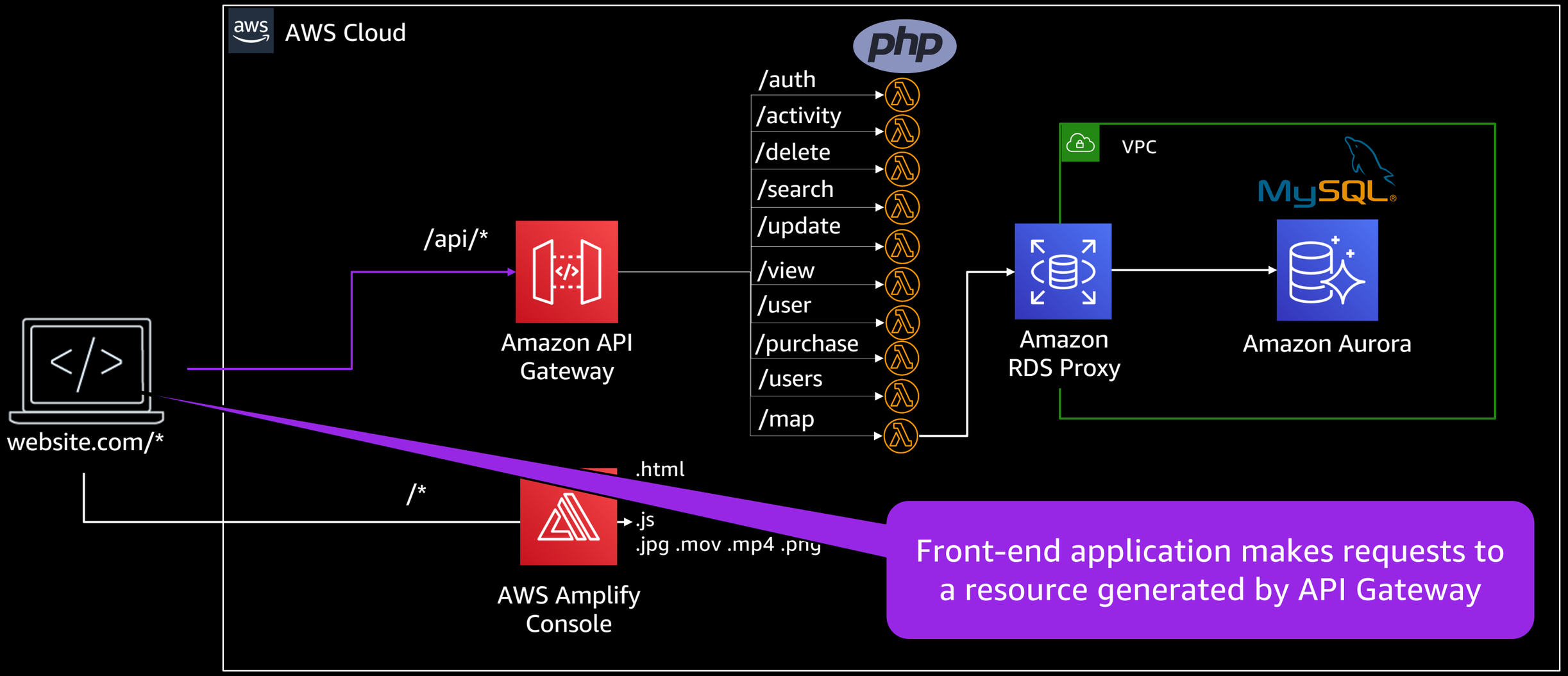
## public/.htaccess

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.php [L]
</IfModule>
```

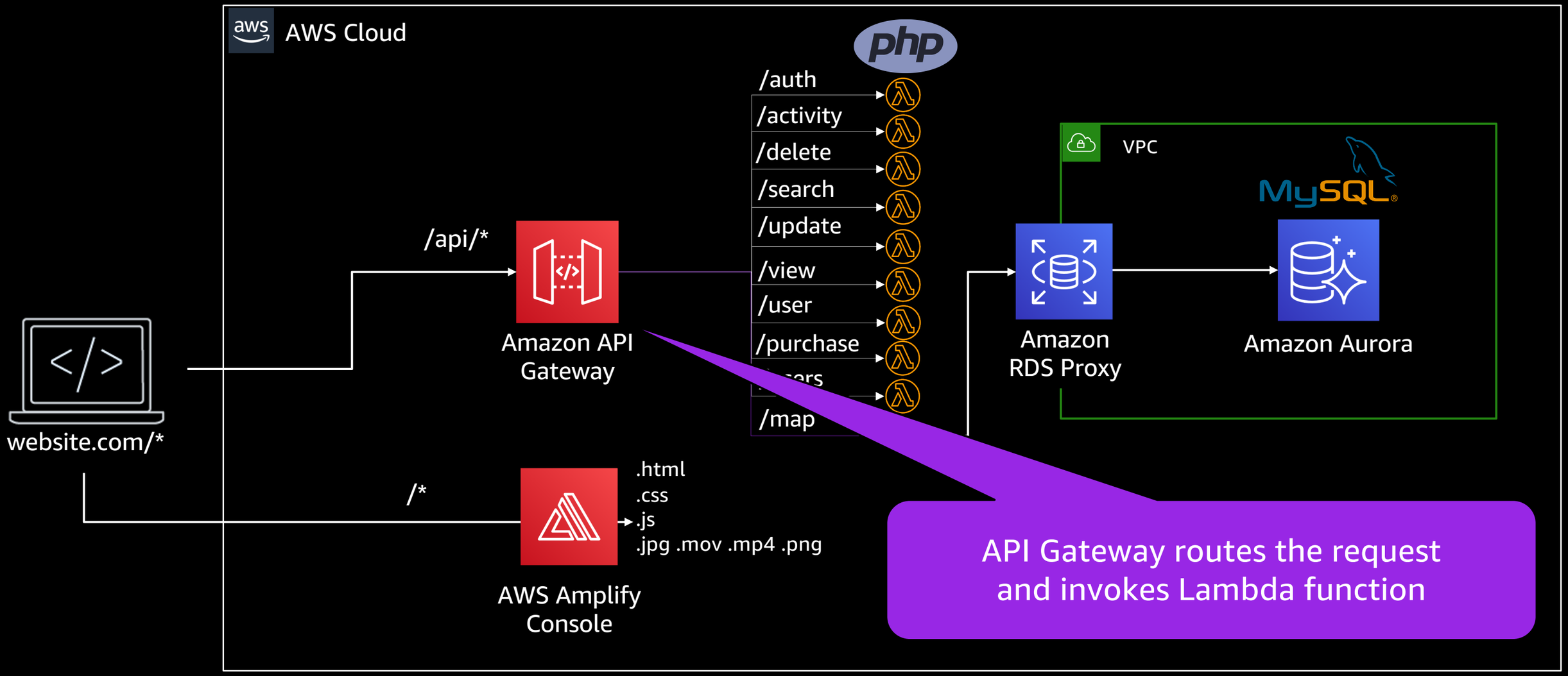
# Event-driven microservices



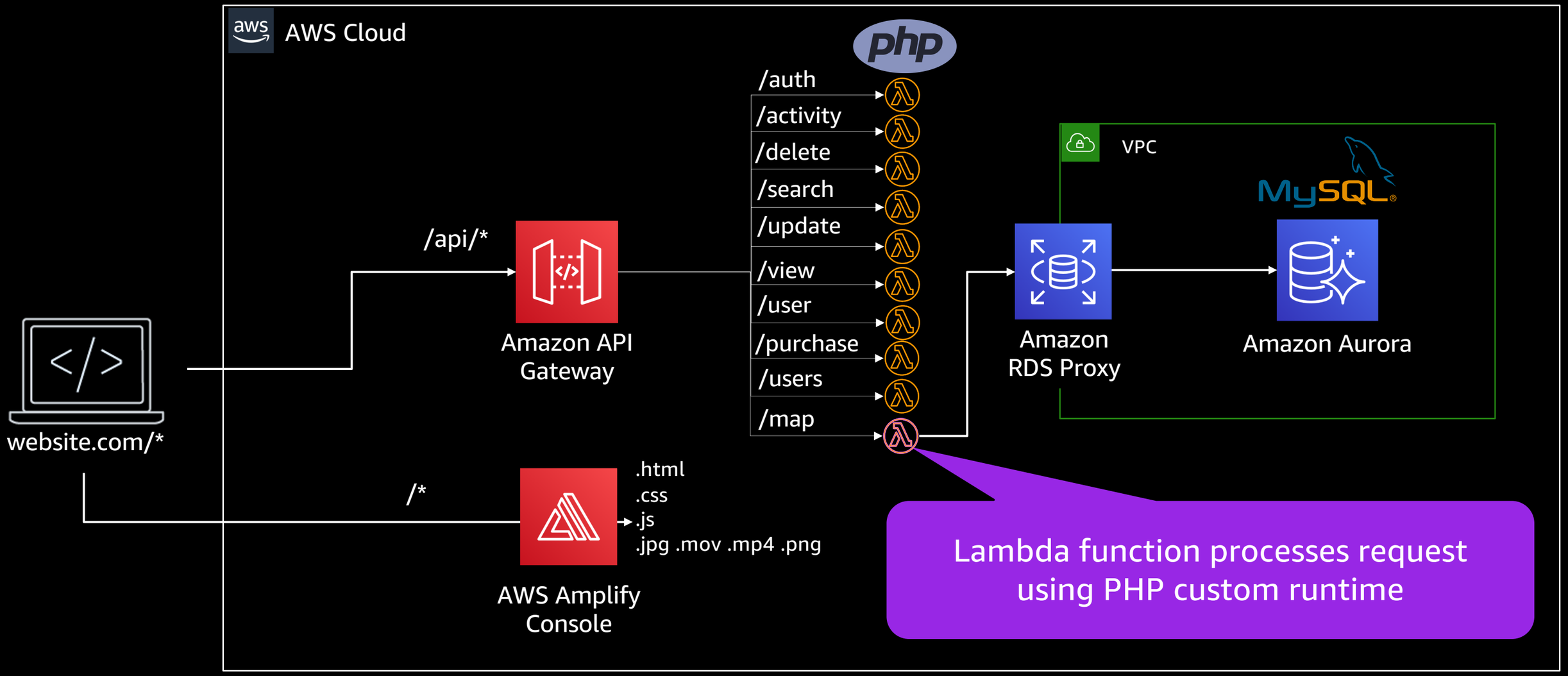
# Event-driven microservices



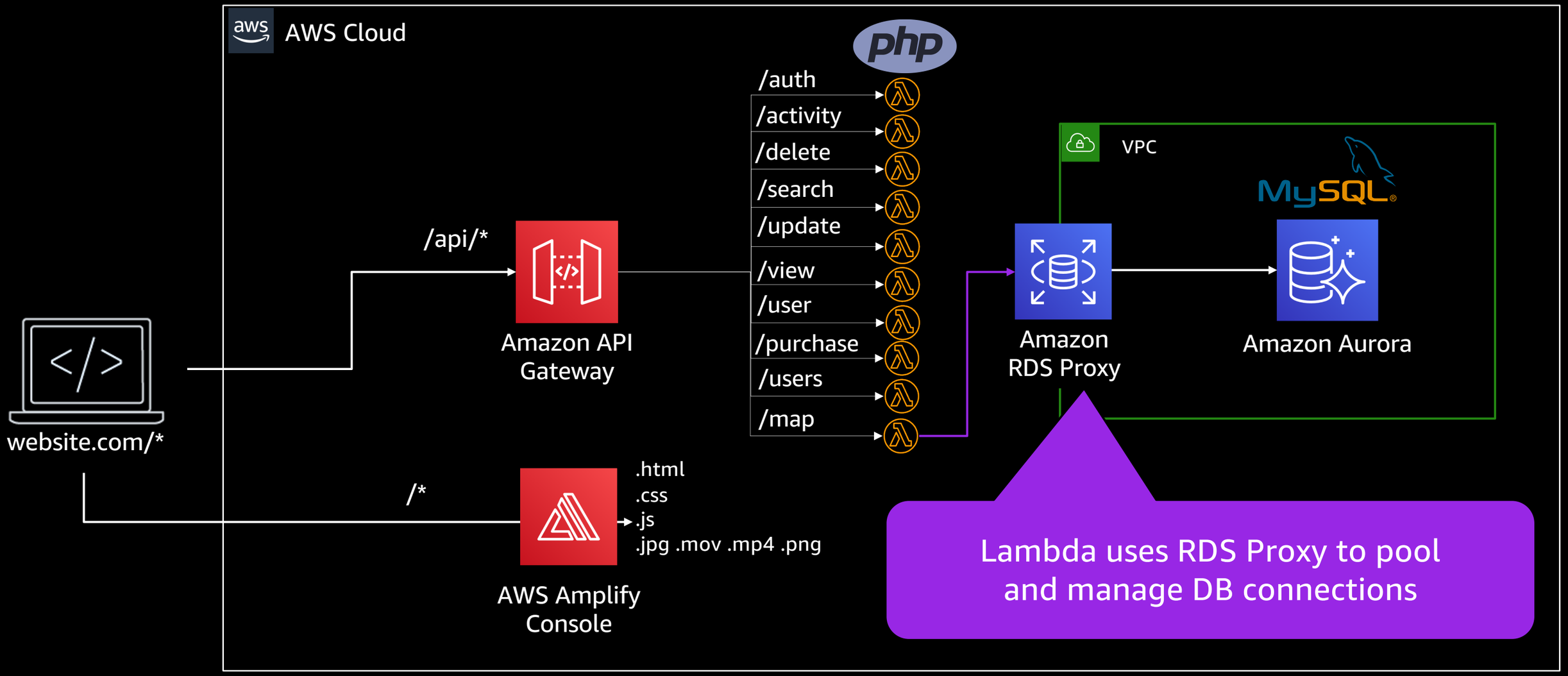
# Event-driven microservices



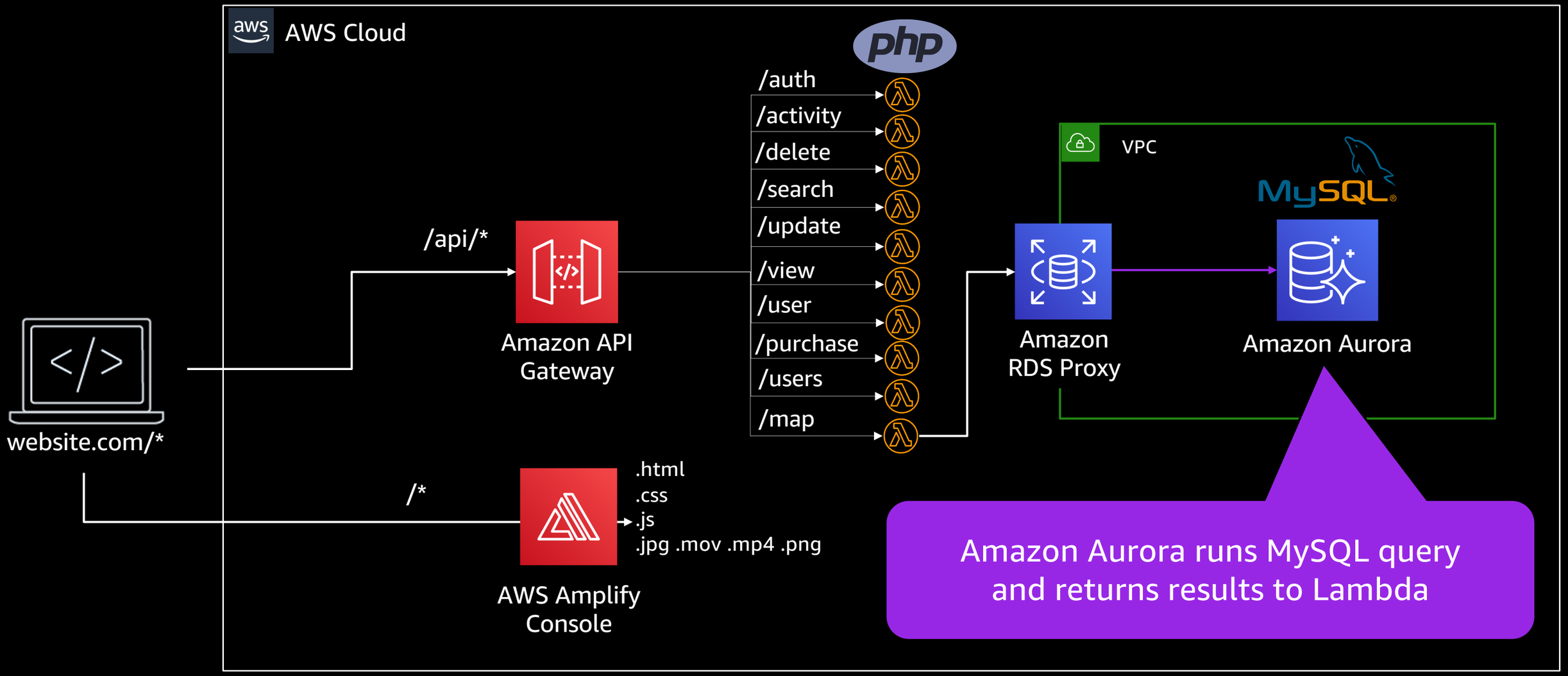
# Event-driven microservices



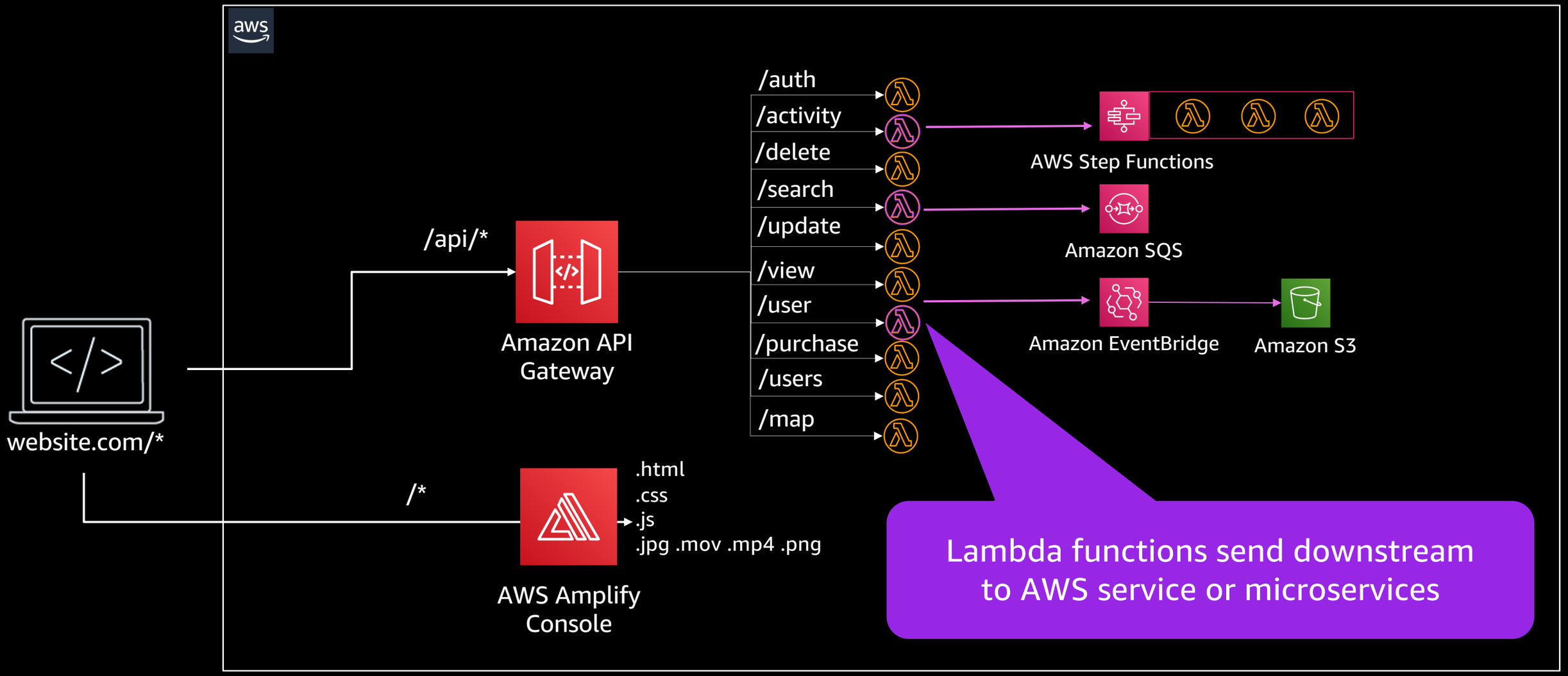
# Event-driven microservices



# Event-driven microservices



# Event-driven microservices

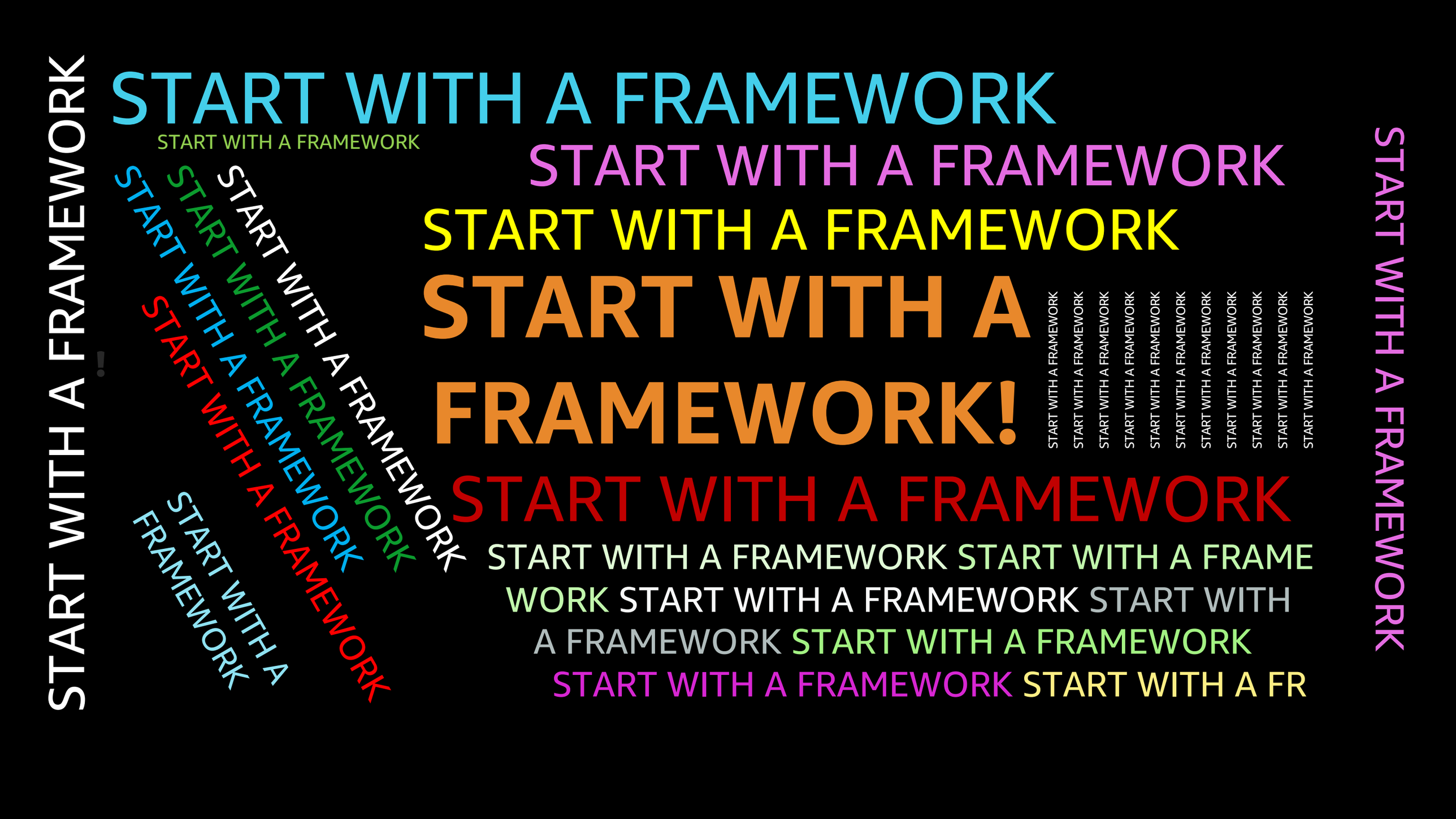


# Getting started

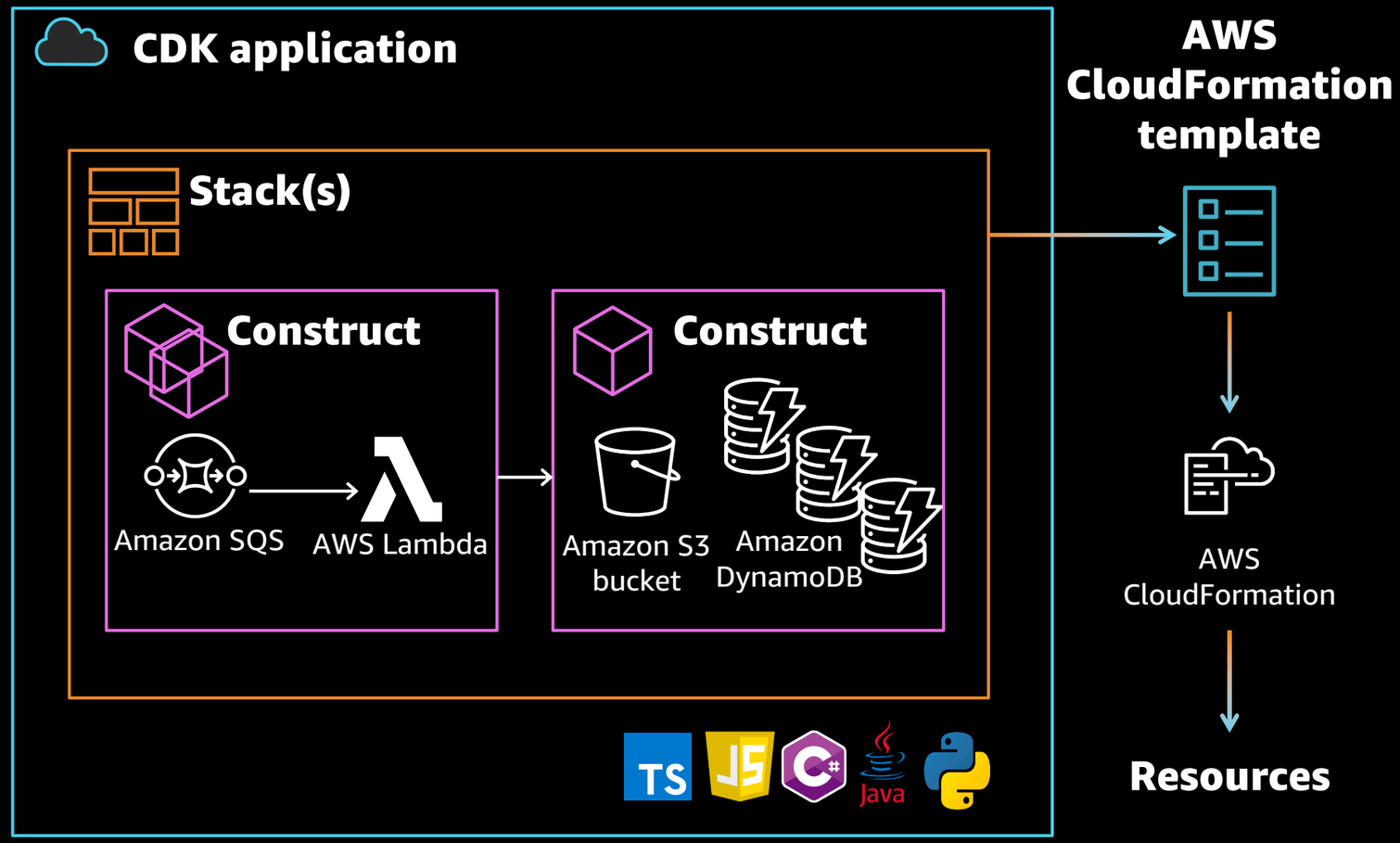


AWS re:Invent

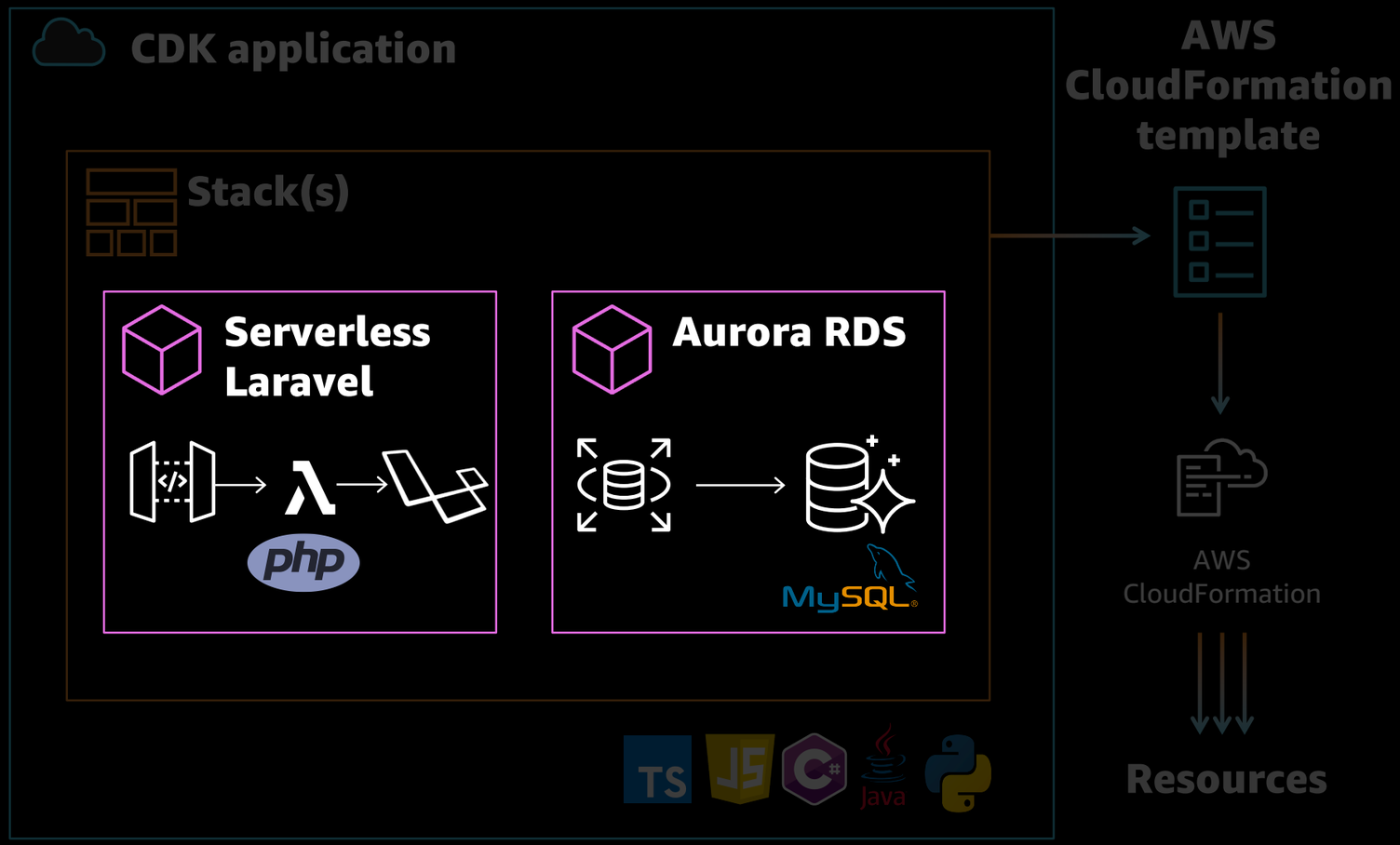
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.


START WITH A FRAMEWORK!
START WITH A FRAMEWORK
START WITH A FRAMEWORK
START WITH A FRAMEWORK

# AWS Cloud Development Kit (AWS CDK)



# Serverless LAMP stack construct library



# Setting up application code base

INSTALL LARAVEL AND OPEN-SOURCE BREF PACKAGES

```
$ mkdir serverless-lamp && cd serverless-lamp

$ mkdir cdk codebase
```

Create directories for AWS CDK and Laravel project

```
$ docker run --rm -ti \
--volume $PWD:/app \
composer create-project --prefer-dist laravel/laravel ./codebase
```

Create the new Laravel project with Docker

```
$ cd codebase
$ docker run --rm -ti \
--volume $PWD:/app \
composer require bref/bref bref/laravel-bridge
```

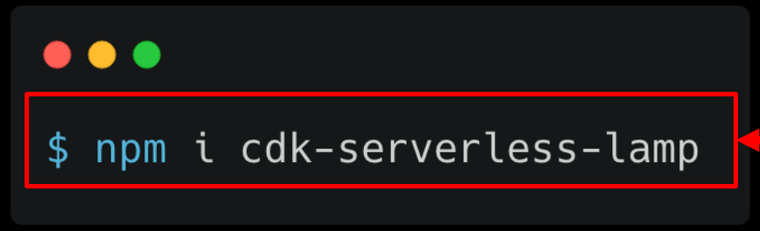
Install bref/bref and bref/laravel-bridge packages in the vendor directories

# Initializing CDK construct

DEFINE CLOUD RESOURCES WITH CDK CONSTRUCTS LIBRARIES



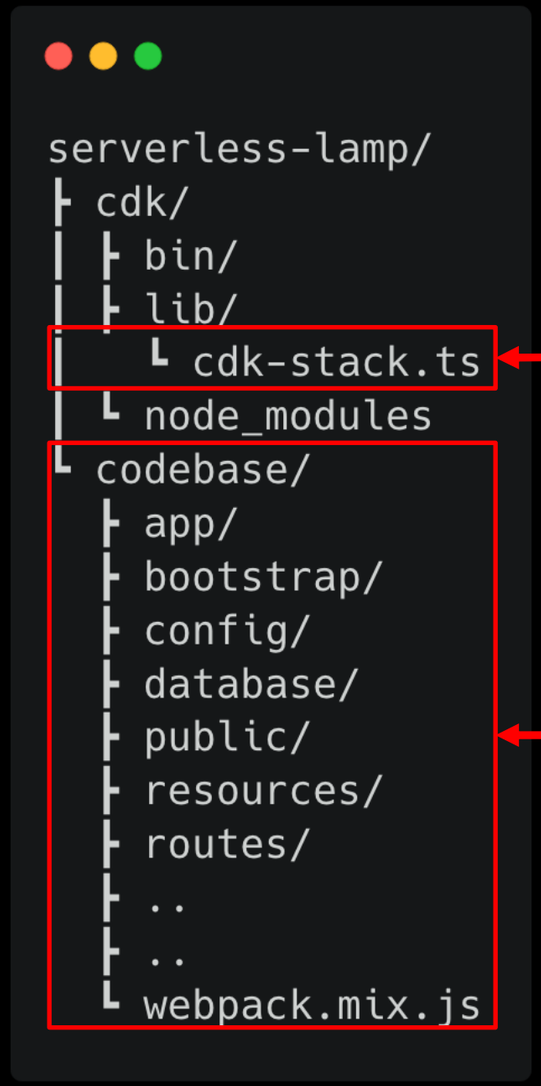

Initialize an AWS CDK project with TypeScript




Install the cdk-severless-lamp npm module

# Initializing CDK for serverless LAMP stack

THE FOLLOWING FILE STRUCTURE IS CREATED




CDK template file to define cloud resources

PHP Laravel application

# What do I do next?

### Articles

- "Authenticating AWS Cognito with Laravel" - *Marco Aurélio Deleu*
- "AWS Lambda With PHP Using Bref And Serverless Framework" - *Adam Culp*
- "Bleeding Edge PHP on ZEIT Now" - *Milan Felix Šulc*
- "Compiling wkhtmltopdf for use inside an AWS Lambda function with Bref is easier than you'd think" - *Neal Brooks*
- "Deploying Laravel Artisan on AWS Lambda" - *Marco Aurélio Deleu*
- "Deploy Static Frontend + PHP Files Using ZEIT Now" - *Milan Felix Šulc*
- "Going serverless with PHP" - *Kpicaza*
- "Introducing the new Serverless LAMP stack" - *Benjamin Smith*
- "PHPUnit with Bref" - *Marco Aurélio Deleu*
- "Responding to SMS in PHP using AWS Lambda and Bref" - *Gary Hockin*
- "Run a serverless Laravel app with queue workers on AWS Lambda using Bref" - *Marco Marassi*
- "Serverless for PHP Developers" - *Gene Surov*
- "Serverless PHP using AWS Lambda" - *Jacques Fu*
- "Symfony Messenger on AWS Lambda" - *Tobias Nyholm*
- "Une introduction au cloud et aux lambdas en PHP" - *Loïck Piera*
- "Using AWS Lambda as a private Laravel API" - *Marco Aurélio Deleu*
- "What is AWS Lambda, and how Laravel Vapor uses it" - *Mohamed Said*

### Case studies

- "From LAMP to serverless: case study of externals.io" - *Matthieu Napoli*
- "How Serverless Saved Us For $2" - *Tristan Bessoussa*
- "Retour d'expérience : Serverless et PHP" - *Gaultier Boniface*
- "Serverless case study: PrettyCI and Laravel Queues" - *Matthieu Napoli*
- "Serverless case study: returntrue.win" - *Matthieu Napoli*
- "Using NewRelic with Bref on AWS Lambda" - *Tobias Nyholm*
- "We rebuilt Fathom Analytics from the ground up and moved to Laravel Vapor" - *Jack Ellis*

### Talks / Videos

- "A journey toward serverless on a ship called PHP" - *Matthieu Napoli*
- "Creating new AWS Lambda PHP Mezzio application with Bref - Beachcasts LIVE!" - *Adam Culp*
- "Getting started with PHP on AWS Lambda" - *Thomas Bley*
- "PHP Layer for AWS Lambda" - *Gaultier Boniface*
- "Serverless PHP applications with Bref - Matthiew Napoli - PHP UK 2020" - *Matthieu Napoli*
- "Serverless PHP? EC2 vs Elasticbeanstalk vs Lambda" - *Kai Hendry*
- "Symfony aws"- *Alessandro Minoccheri*

### Tools

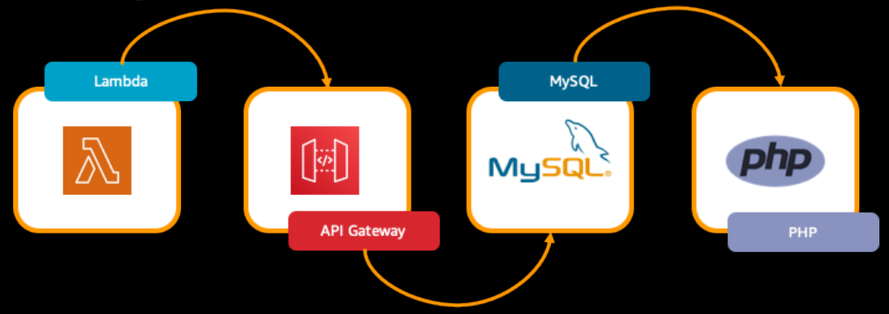
- "All you need to log with Bref on AWS Lambda" - *brefphp community*
- "Bref PHP extensions separately available as Lambda layers" - *brefphp community*
- "Bref, the Laravel way" - *Signature Tech Studio*
- "Bridge to use Symfony Messenger on AWS Lambda with Bref" - *brefphp community*
- "Driver for AWS Aurora serverless rds data api / doctrine database abstraction layer" - *Marco Pfeiffer*
- "Examples of PHP applications built with Bref" - *brefphp community*
- "Package to use Laravel on AWS Lambda with Bref" - *brefphp community*
- "Small library to write good Lambda hooks" - *Tobias Nyholm*
- "Vercel PHP runtime" - *Milan Felix Šulc*

### Events

- June 25/25 2020, Dutch PHP Conference Online Serverless PHP applications with Bref - *Matthieu Napoli*
- July 1 2020, ServerlessDays Paris - Remote Session #3 The new LAMP stack is serverless - *Benjamin Smith*
- September 24/25 2020, Serverless Days Zurich, The new LAMP Stack for 2020, - *Benjamin Smith*

## AWS Serverless PHP code examples on GitHub

https://s12d.com/LAMP




## Community-curated resources

https://s12d.com/LAMPresources




## Open-source PHP layer for Lambda

https://s12d.com/bref

# Recap

The traditional LAMP Stack doesn’t scale well

With serverless technologies, PHP developers can

- Build scalable web applications without managing infrastructure
- Build decoupled microservices for backend applications
- Use mature, existing PHP frameworks
- Move from prototype to production at speed
- Compile their own custom runtime or use an open-source layer

You can build **serverless applications** with PHP

# Thank you!

Benjamin Smith

Twitter: @Benjamin_l_s



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



# Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.