

Introduction to Serverless PHP

Rob Allen

February 2019

Slides: <https://akrabat.com/5703>

Platform options

Physical servers (Dell/HP)

Platform options

Virtual machines (EC2)

Physical servers (Dell/HP)

Platform options

Containers (Kubernetes)

Virtual machines (EC2)

Physical servers (Dell/HP)

Platform options

Platform (CloudFoundry)

Containers (Kubernetes)

Virtual machines (EC2)

Physical servers (Dell/HP)

Platform options

Serverless (OpenWhisk)

Platform (CloudFoundry)

Containers (Kubernetes)

Virtual machines (EC2)

Physical servers (Dell/HP)

Platform options



Serverless (OpenWhisk)

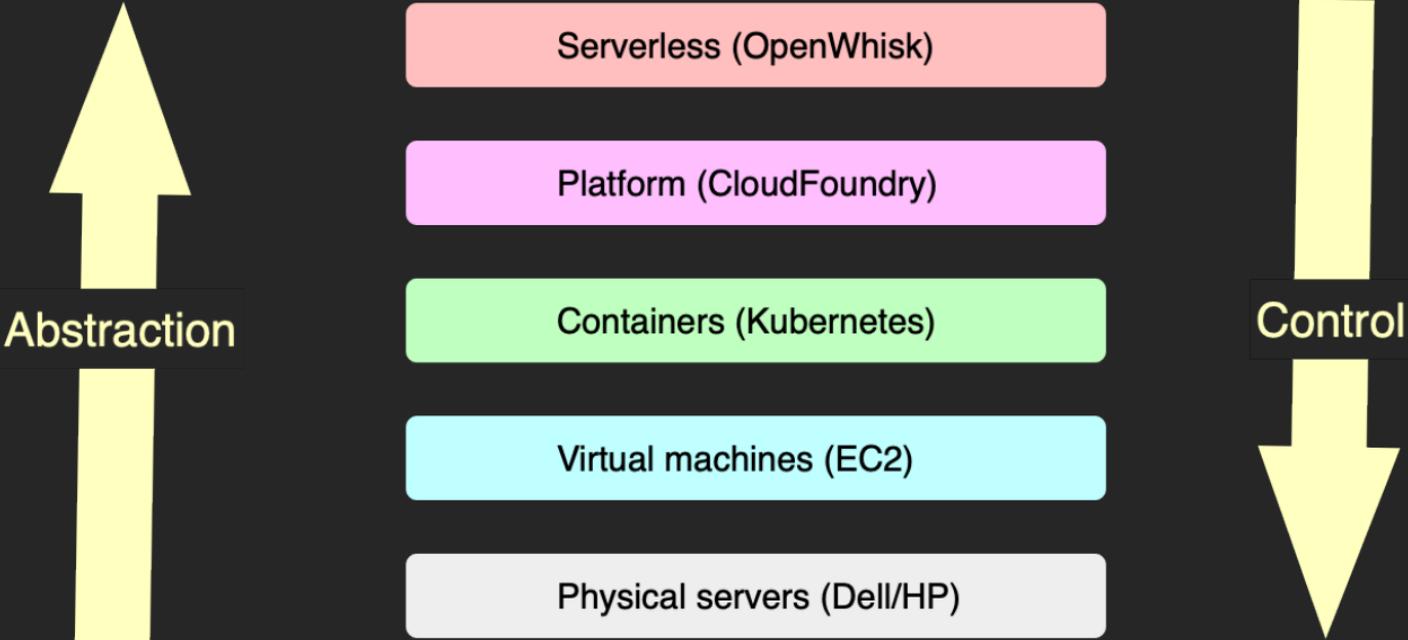
Platform (CloudFoundry)

Containers (Kubernetes)

Virtual machines (EC2)

Physical servers (Dell/HP)

Platform options



Serverless

Serverless is all about composing software systems from a collection of cloud services.

With serverless, you can lean on off-the-shelf cloud services resources for your application architecture, focus on business logic and application needs.

Nate Taggart, CEO Stackery

FaaS

- Your code
- Deployed to the cloud
- Runs when needed
- Scaled automatically
- Pay only for execution

Where are the servers?





**Not Your
PROBLEM!**



Use-cases



Use-cases

Synchronous

Service is invoked and provides immediate response
(HTTP requests: APIs, chat bots)



Use-cases

Synchronous

Service is invoked and provides immediate response
(HTTP requests: APIs, chat bots)

Asynchronous

Push a message which drives an action later
(web hooks, timed events, database changes)



Benefits

Benefits

- No need to maintain infrastructure

Benefits

- No need to maintain infrastructure
- Concentrate on application code

Benefits

- No need to maintain infrastructure
- Concentrate on application code
- Pay only for what you use, when you use it

Benefits

- No need to maintain infrastructure
- Concentrate on application code
- Pay only for what you use, when you use it
- Language agnostic

Challenges

Challenges

- Start up latency

Challenges

- Start up latency
- Time limit

Challenges

- Start up latency
- Time limit
- State is external

Challenges

- Start up latency
- Time limit
- State is external
- Different way of thinking

It's about value



Beau @BeauVrolyk · 30m

Replying to @akrabat @kelseyhightower

1) "Serverless" is a point on the path to true app isolation. Apps want to just run, their authors don't care about infrastructure at all.



Beau @BeauVrolyk · 29m

Replying to @akrabat @kelseyhightower

2) The App author should not need to know, anymore than a Journalist knows about printing presses or what the voltage of the power used.



1



1



2



Beau @BeauVrolyk · 25m

Replying to @akrabat @kelseyhightower

3) We are relearning what was known in the time-share days. Pricing needs to be based on something customers value, not infa. items like VMs



When should you use serverless?

- Responding to web hooks

When should you use serverless?

- Responding to web hooks
- PWA/Static site contact form, et al.



When should you use serverless?

- Responding to web hooks
- PWA/Static site contact form, et al.
- Additional features without extending current platform

When should you use serverless?

- Responding to web hooks
- PWA/Static site contact form, et al.
- Additional features without extending current platform
- Variable traffic levels

When should you use serverless?

- Responding to web hooks
- PWA/Static site contact form, et al.
- Additional features without extending current platform
- Variable traffic levels
- When you want your costs to scale with traffic

Serverless platforms



Google Cloud Platform



IBM Cloud



Serverless platforms with PHP support



Google Cloud Platform



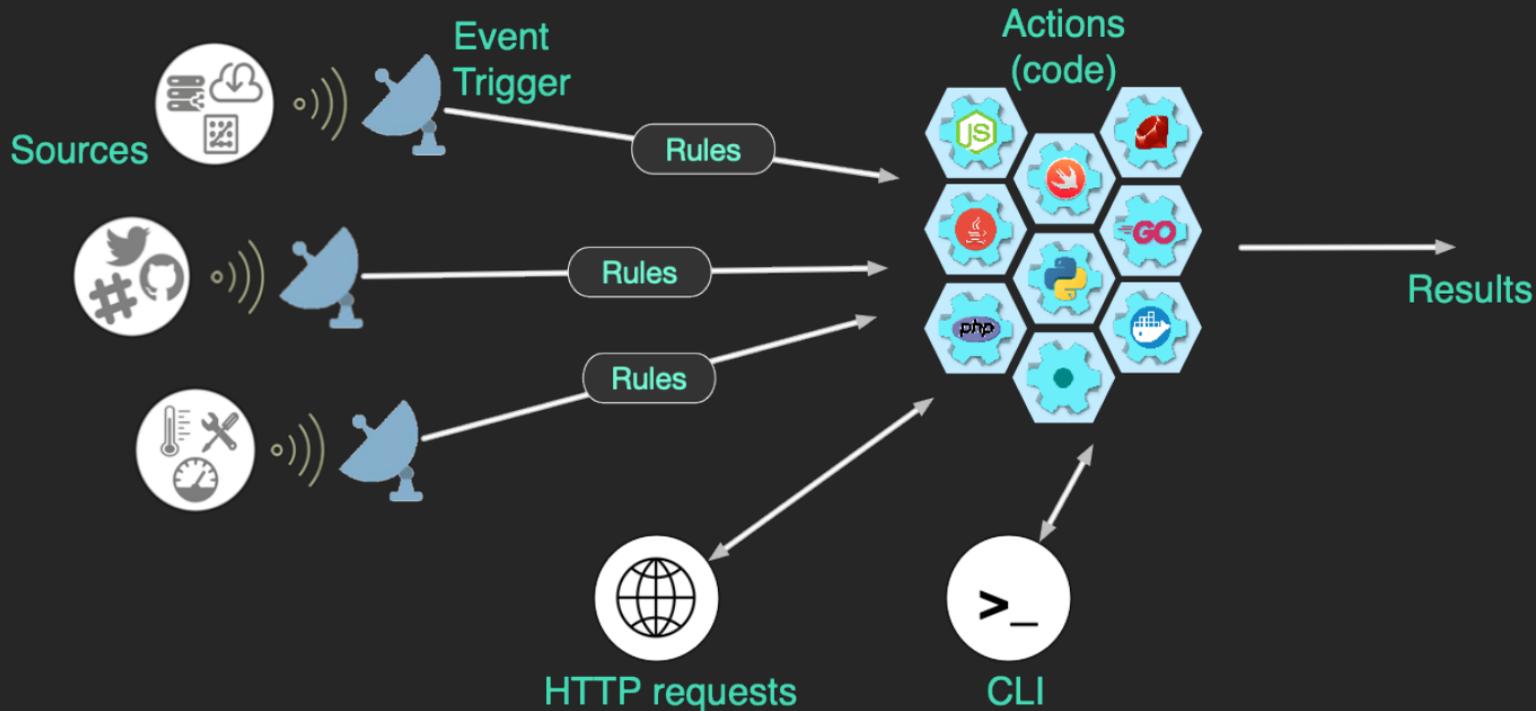


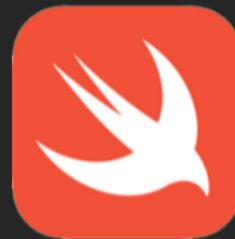
APACHE

OpenWhisk



Concepts







Java



Hello world in PHP

```
1 <?php
2 function main(array $args) : array
3 {
4     $name = $args["name"] ?? "World";
5
6     return [ "msg" => "Hello $name" ];
7 }
```

Hello world in PHP

Entry point

Event parameters

```
1 <?php
2 function main(array $args) : array
3 {
4     $name = $args["name"] ?? "World";
5
6     return [ "msg" => "Hello $name" ];
7 }
```

Service result

Upload your action

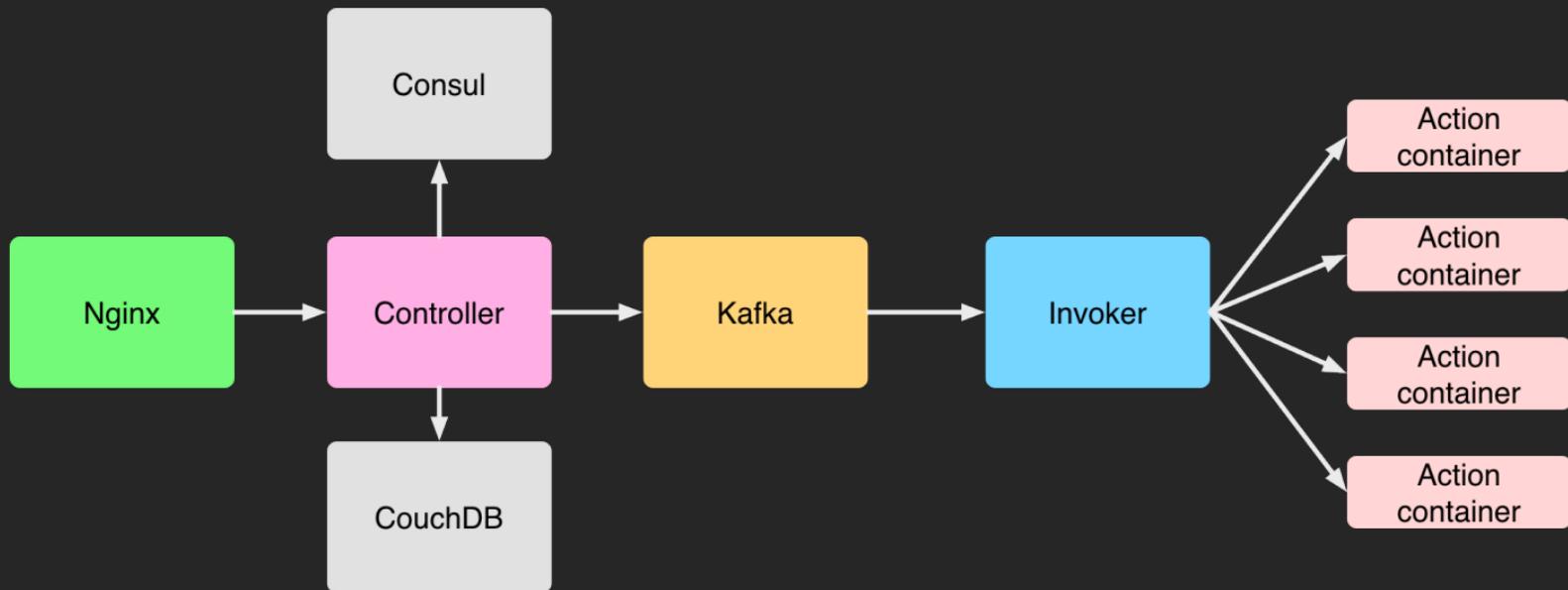
```
$ wsk action update hello hello.php  
ok: updated action hello
```

Run your action

```
$ wsk action invoke hello --result  
{  
  "msg": "Hello World"  
}
```

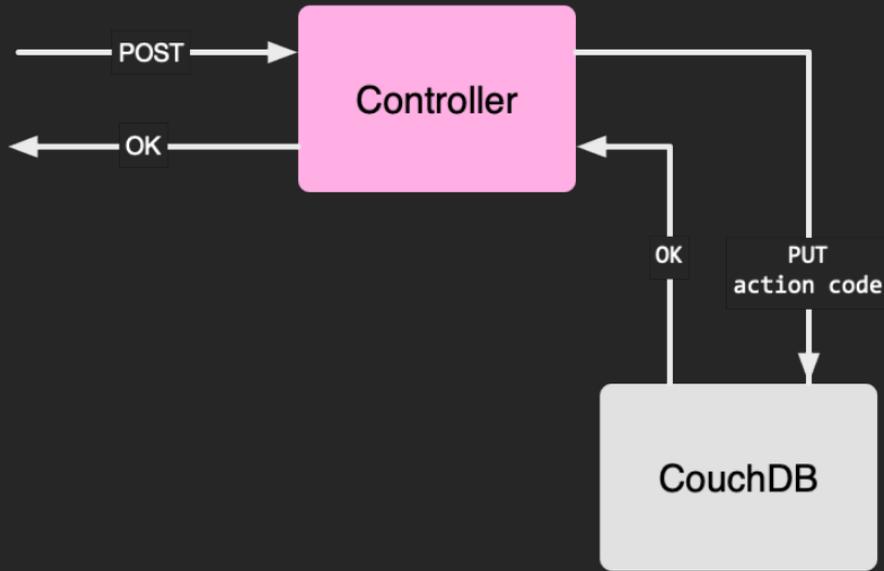
Segue: How did it do this?

OpenWhisk's architecture



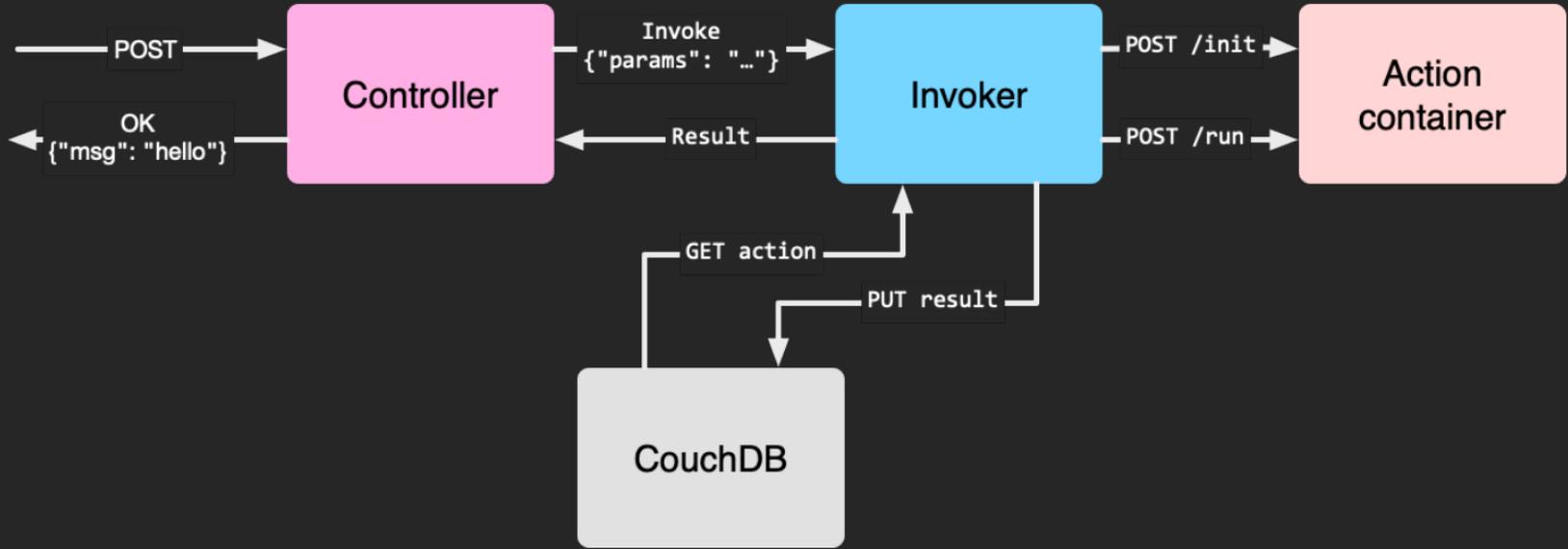
Create an action

```
$ wsk action create hello hello.php
```



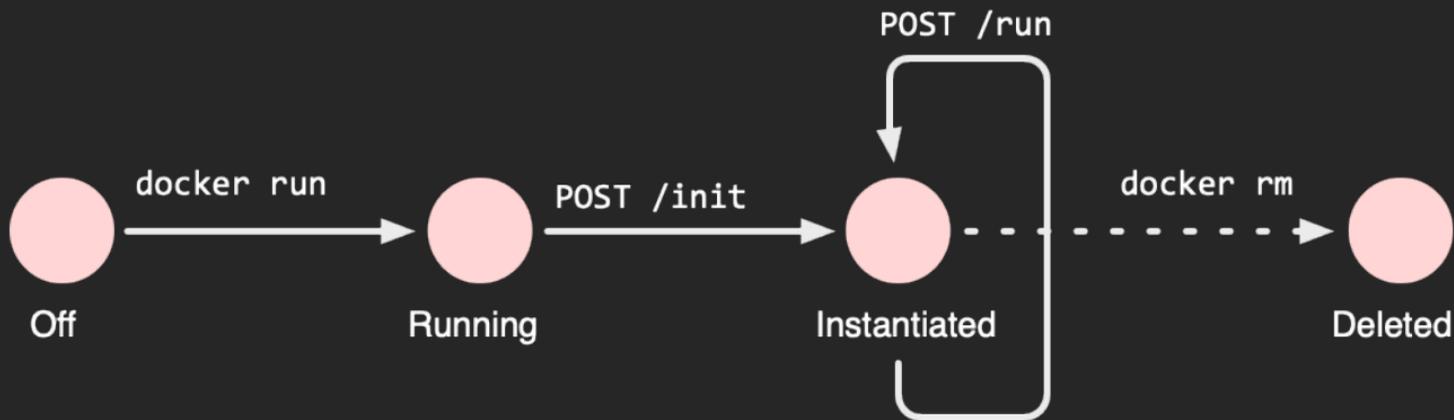
Invoke an action

```
$ wsk action invoke hello -r
```



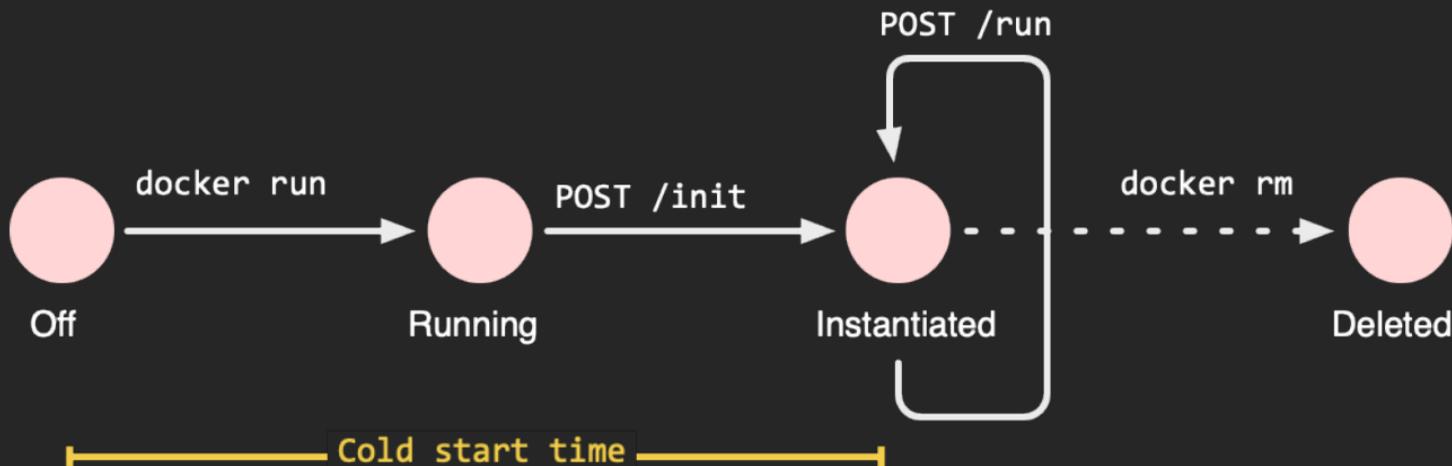
Action container lifecycle

- Hosts the user-written code
- Controlled via two end points: `/init` & `/run`



Action container lifecycle

- Hosts the user-written code
- Controlled via two end points: `/init` & `/run`



End Segue

Public web access

Add the `--web` flag:

```
$ wsk action update hello hello.php --web true
```

Public web access

Add the `--web` flag:

```
$ wsk action update hello hello.php --web true
$ curl https://openwhisk.ng.bluemix.net/api/v1/web/ \
    19FT_demo/default/hello.json
{
  "msg": "Hello World"
}
```

API Gateway

When you want to do more with HTTP endpoints

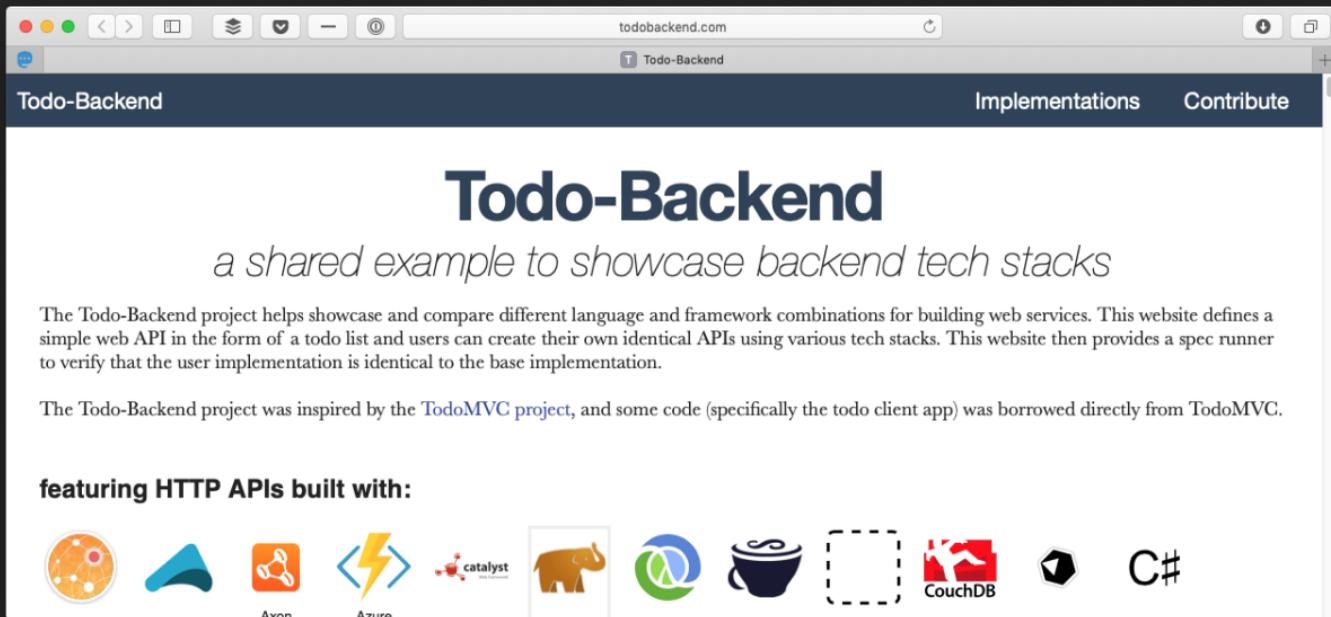
- Route endpoint methods to actions (Open API Spec support)
- Custom domains
- Rate limiting
- Security (API keys, OAuth, CORS)
- Analytics

API Gateway



```
$ wsk api create /myapp /hello GET hello  
ok: created API /myapp/hello GET for action /_/hello
```

Todo-Backend



The screenshot shows a web browser window with the URL `todobackend.com`. The page has a dark blue header with the text "Todo-Backend" on the left and "Implementations" and "Contribute" on the right. The main content area features the title "Todo-Backend" in a large, bold, dark blue font, followed by the subtitle *a shared example to showcase backend tech stacks* in a smaller, italicized font. Below this, there are two paragraphs of text. The first paragraph explains the project's purpose: to showcase and compare different language and framework combinations for building web services. The second paragraph mentions that the project was inspired by the [TodoMVC project](#) and that some code was borrowed from it. At the bottom, there is a section titled "featuring HTTP APIs built with:" followed by a row of ten icons representing various technologies: a circular orange icon with dots, a blue triangle, an orange square with a white circle, a blue lightning bolt, a red square with white dots and the word "catalyst", a brown elephant, a green circle with a white triangle, a blue coffee cup, a dashed square, a red square with a white figure and the text "CouchDB", a black hexagon, and the text "C#".

Todo-Backend Implementations Contribute

Todo-Backend

a shared example to showcase backend tech stacks

The Todo-Backend project helps showcase and compare different language and framework combinations for building web services. This website defines a simple web API in the form of a todo list and users can create their own identical APIs using various tech stacks. This website then provides a spec runner to verify that the user implementation is identical to the base implementation.

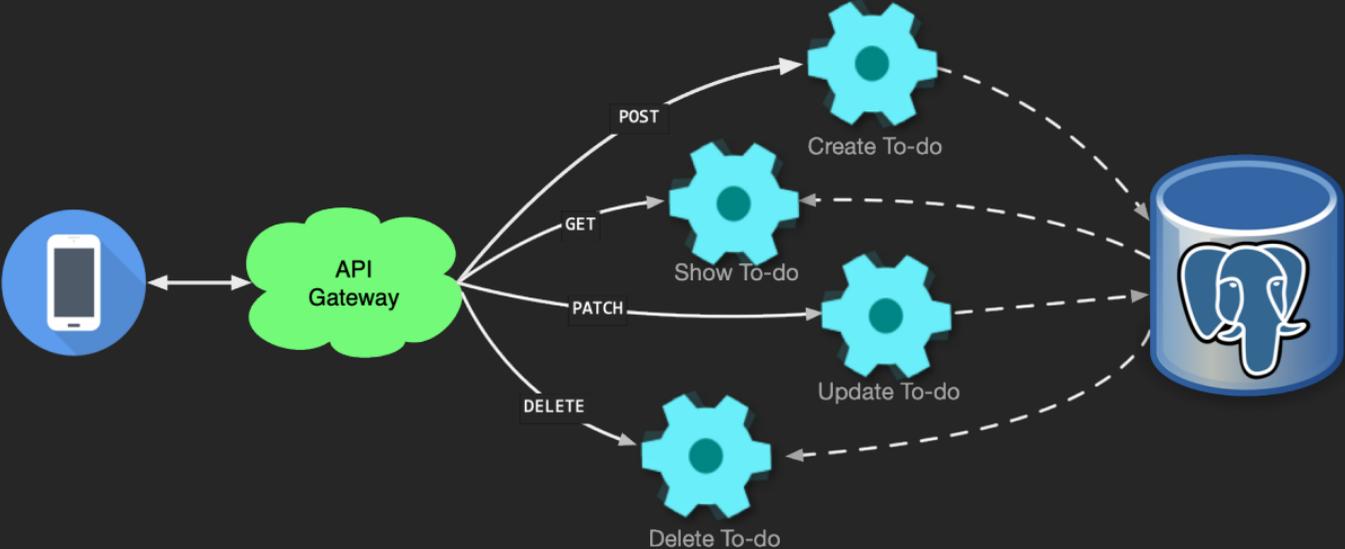
The Todo-Backend project was inspired by the [TodoMVC project](#), and some code (specifically the todo client app) was borrowed directly from TodoMVC.

featuring HTTP APIs built with:



Todo-Backend

Add, edit and delete to-do items using OpenWhisk (on IBM) with PostgreSQL



Serverless Framework

```
application manifest: serverless.yml
```

```
service: ow-todo-backend
```

```
provider:
```

```
  name: openwhisk
```

```
  runtime: php
```

```
plugins:
```

```
  - serverless-openwhisk
```



Configure action

```
functions:  
  edit-todo:  
    handler: "src/actions/editTodo.main"  
    name: "todo-backend/edit-todo"
```

Configure action

```
functions:  
  edit-todo:  
    handler: "src/actions/editTodo.main"  
    name: "todo-backend/edit-todo"
```

Configure action

```
functions:  
  edit-todo:  
    handler: "src/actions/editTodo.main"  
    name: "todo-backend/edit-todo"
```

Configure API Gateway

```
functions:
  edit-todo:
    handler: "src/actions/editTodo.main"
    name: "todo-backend/edit-todo"
  events:
    - http:
        path: /todos/{id}
        method: patch
```

Configure API Gateway

```
functions:
  edit-todo:
    handler: "src/actions/editTodo.main"
    name: "todo-backend/edit-todo"
    events:
      - http:
          path: /todos/{id}
          method: patch
```

Project files

```
.  
├── src/  
├── vendor/  
├── composer.json  
├── composer.lock  
└── serverless.yml
```

Project files



editTodo.php

```
function main(array $args) : array
{
    try {
        $parts = explode("/", $args['__ow_path']);
        $id = (int)array_pop($parts);

        $data = json_decode(base64_decode($args['__ow_body']), true);
        if (!is_array($data)) {
            throw new InvalidArgumentException('Missing body', 400);
        }

        $container = new AppContainer($args);
        $mapper = $container[TodoMapper::class];

        $todo = $mapper->loadById($id);
        $mapper->update($todo, $data);

        $transformer = $container[TodoTransformer::class];
        $resource = new Item($todo, $transformer, 'todos');
        $fractal = $container[Manager::class];

        return [
            'statusCode' => 200,
            'body' => $fractal->createData($resource)->toArray(),
        ];
    } catch (\Throwable $e) {
        var_dump((string)$e);
        $code = $e->getCode() < 400 ? $e->getCode(): 500;
        return [
            'statusCode' => $code,
            'body' => ['error' => $e->getMessage()];
        ]
    }
}
```



editTodo.php: Error handling

```
function main(array $args) : array
{
    try {
        // do stuff
    } catch (\Throwable $e) {
        var_dump((string)$e);
        $code = $e->getCode() < 400 ? $e->getCode() : 500;
        return [
            'statusCode' => $code,
            'body' => ['error' => $e->getMessage()]];
    }
}
```

editTodo.php: Error handling

```
function main(array $args) : array
{
    try {
        // do stuff
    } catch (\Throwable $e) {
        var_dump((string)$e);
        $code = $e->getCode() < 400 ? $e->getCode() : 500;
        return [
            'statusCode' => $code,
            'body' => ['error' => $e->getMessage()]];
    }
}
```

editTodo.php: Error handling

```
function main(array $args) : array
{
    try {
        // do stuff
    } catch (\Throwable $e) {
        var_dump((string)$e);
        $code = $e->getCode() < 400 ? $e->getCode() : 500;
        return [
            'statusCode' => $code,
            'body' => ['error' => $e->getMessage()]];
    }
}
```



editTodo.php: Error handling

```
function main(array $args) : array
{
    try {
        // do stuff
    } catch (\Throwable $e) {
        var_dump((string)$e);
        $code = $e->getCode() < 400 ? $e->getCode() : 500;
        return [
            'statusCode' => $code,
            'body' => ['error' => $e->getMessage()]];
    }
}
```



editTodo.php: Grab input

```
$parts = explode("/", $args['__ow_path']);  
$id = (int)array_pop($parts);
```



editTodo.php: Grab input

```
$parts = explode("/", $args['__ow_path']);  
$id = (int)array_pop($parts);  
  
$body = base64_decode($args['__ow_body']);  
$data = json_decode($body, true);  
if (!is_array($data)) {  
    throw new Exception('Missing body', 400);  
}
```

editTodo.php: Grab input

```
$parts = explode("/", $args['__ow_path']);  
$id = (int)array_pop($parts);  
  
$body = base64_decode($args['__ow_body']);  
$data = json_decode($body, true);  
if (!is_array($data)) {  
    throw new Exception('Missing body', 400);  
}
```

editTodo.php: Do the work

```
$container = new AppContainer($args);  
$mapper = $container[TodoMapper::class];
```

```
$todo = $mapper->loadById($id);  
$mapper->update($todo, $data);
```

editTodo.php: Do the work

```
$container = new AppContainer($args);  
$mapper = $container[TodoMapper::class];
```

```
$todo = $mapper->loadById($id);  
$mapper->update($todo, $data);
```

editTodo.php: Do the work

```
$container = new AppContainer($args);  
$mapper = $container[TodoMapper::class];
```

```
$todo = $mapper->loadById($id);  
$mapper->update($todo, $data);
```

editTodo.php: Present results

```
$transformer = $container[TodoTransformer::class];  
$resource = new Item($todo, $transformer, 'todos');  
$fractal = $container[Manager::class];  
  
$output = $fractal->createData($resource);  
  
return [  
    'statusCode' => 200,  
    'body' => $output->toArray(),  
];
```

editTodo.php: Present results

```
$transformer = $container[TodoTransformer::class];  
$resource = new Item($todo, $transformer, 'todos');  
$fractal = $container[Manager::class];  
  
$output = $fractal->createData($resource);  
  
return [  
    'statusCode' => 200,  
    'body' => $output->toArray(),  
];
```

editTodo.php: Present results

```
$transformer = $container[TodoTransformer::class];  
$resource = new Item($todo, $transformer, 'todos');  
$fractal = $container[Manager::class];  
  
$output = $fractal->createData($resource);  
  
return [  
    'statusCode' => 200,  
    'body' => $output->toArray(),  
];
```

editTodo.php: Present results

```
$transformer = $container[TodoTransformer::class];  
$resource = new Item($todo, $transformer, 'todos');  
$fractal = $container[Manager::class];  
  
$output = $fractal->createData($resource);  
  
return [  
    'statusCode' => 200,  
    'body' => $output->toArray(),  
];
```

Deploy

```
$ serverless deploy
Serverless: Packaging service...
Serverless: Compiling Functions...
Serverless: Compiling Packages...
Serverless: Compiling API Gateway definitions...
Serverless: Compiling Rules...
Serverless: Compiling Triggers & Feeds...
Serverless: Compiling Service Bindings...
Serverless: Deploying Packages...
Serverless: Deploying Functions...
Serverless: Deploying API Gateway definitions...
[...]
```




AWS Lambda with PHP

Only *sensibly* possible since November 2018 with the introduction of *layers*

AWS Lambda with PHP

Only *sensibly* possible since November 2018 with the introduction of *layers*

Process:

1. Create a layer containing:
 1. the PHP executable
 2. a bootstrap script
2. Write the PHP function!

Full details: akrabat.com/lambdaphp

Bootstrap file

```
while (true) {  
    // get next event  
    $eventPayload = $lambdaRuntime->getEventPayload();  
  
    // execute handler  
    $data = $function($eventPayload);  
  
    // send result  
    $lambdaRuntime->sendResult($data);  
}
```



Manage using Serverless Framework

```
service: hello-lambdaphp
```

```
provider:
```

```
  name: aws
```

```
  runtime: provided
```

```
  memorySize: 128
```

```
layers:
```

```
  php:
```

```
    path: layer/php
```

Manage using Serverless Framework

```
service: hello-lambdaphp
```

```
provider:
```

```
  name: aws
```

```
  runtime: provided
```

```
  memorySize: 128
```

```
layers:
```

```
  php:
```

```
    path: layer/php
```

Manage using Serverless Framework

```
service: hello-lambdaphp
```

```
provider:
```

```
  name: aws
```

```
  runtime: provided
```

```
  memorySize: 128
```

```
layers:
```

```
  php:
```

```
    path: layer/php
```

Hello World in Lambda

```
function hello($eventData)
{
    $data = json_decode($eventData, true, 512,
        JSON_THROW_ON_ERROR);

    $name = $data['name'] ?? 'World';

    return json_encode(['msg' => "Hello $name"]);
}
```

Hello World in Lambda

```
function hello($eventData)
{
    $data = json_decode($eventData, true, 512,
        JSON_THROW_ON_ERROR);

    $name = $data['name'] ?? 'World';

    return ['msg' => "Hello $name"];
}
```

Hello World in Lambda

```
function hello($eventData)
{
    $data = json_decode($eventData, true, 512,
        JSON_THROW_ON_ERROR);

    $name = $data['name'] ?? 'World';

    return ['msg' => "Hello $name"];
}
```

Hello World in Lambda

```
function hello($eventData)
{
    $data = json_decode($eventData, true, 512,
        JSON_THROW_ON_ERROR);

    $name = $data['name'] ?? 'World';

    return ['msg' => "Hello $name"];
}
```

Manage using Serverless Framework

```
functions:  
  hello:  
    handler: handler.hello  
    layers:  
      - {Ref: PhpLambdaLayer}
```



Deploy using Serverless Framework

```
$ sls deploy
```

```
Serverless: Packaging service...
```

```
Serverless: Uploading CloudFormation file to S3...
```

```
...
```

```
functions:
```

```
  hello: hello-lambda-php-dev-hello
```

```
layers:
```

```
  php: arn:aws:lambda:eu-west-2:661969457706:layer:php:
```

```
Serverless: Removing old service artifacts from S3...
```

Invoke using Serverless Framework

```
$ sls invoke -f hello --data='{ "name": "Rob" }'  
{  
  "msg": "Hello Rob"  
}
```

Bref

Community maintained PHP runtime for AWS Lambda

```
<?php
```

```
require __DIR__.' /vendor/autoload.php';
```

```
lambda(function (array $event) {  
    $name = $event['name'] ?? 'World';  
    return ['msg' => "Hello $name"];  
});
```

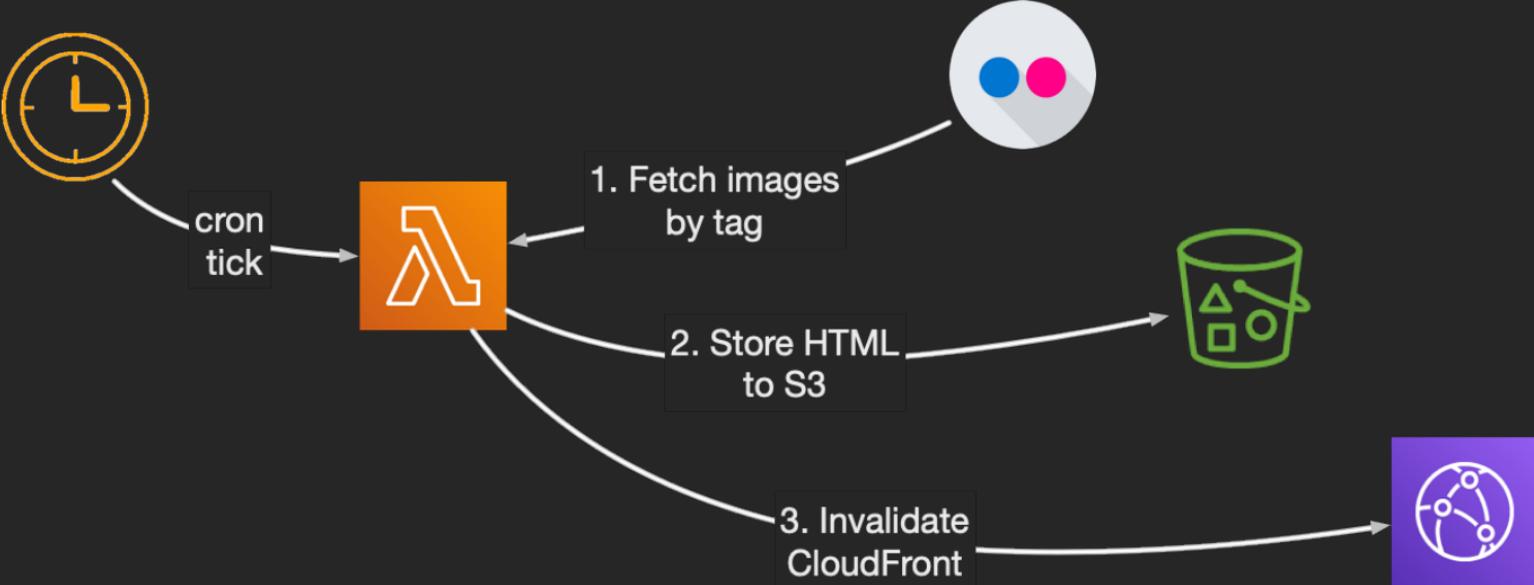
Project 365
My photo-a-day website

Project 365

Static website to display my photo-a-day picture for each day of the year.

- Hosted on S3
- CloudFront CDN
- Lambda/PHP function

Lambda/PHP function



Serverless configuration

```
functions:
  update:
    handler: src/actions/update.main
    layers:
      - {Ref: PhpLambdaLayer}
    environment:
      FLICKR_API_KEY: ${self:custom.FLICKR_API_KEY}
      FLICKR_USER_ID: ${self:custom.FLICKR_USER_ID}
    events:
      - schedule:
          name: project365-build
          rate: cron(0 */2 * * ? *)
```



Serverless configuration

```
functions:
```

```
  update:
```

```
    handler: src/actions/update.main
```

```
    layers:
```

```
      - {Ref: PhpLambdaLayer}
```

```
    environment:
```

```
      FLICKR_API_KEY: ${self:custom.FLICKR_API_KEY}
```

```
      FLICKR_USER_ID: ${self:custom.FLICKR_USER_ID}
```

```
    events:
```

```
      - schedule:
```

```
        name: project365-build
```

```
        rate: cron(0 */2 * * ? *)
```

Serverless configuration

```
functions:
  update:
    handler: src/actions/update.main
    layers:
      - {Ref: PhpLambdaLayer}
    environment:
      FLICKR_API_KEY: ${self:custom.FLICKR_API_KEY}
      FLICKR_USER_ID: ${self:custom.FLICKR_USER_ID}
    events:
      - schedule:
          name: project365-build
          rate: cron(0 */2 * * ? *)
```

Serverless configuration

```
functions:
  update:
    handler: src/actions/update.main
    layers:
      - {Ref: PhpLambdaLayer}
  environment:
    FLICKR_API_KEY: ${self:custom.FLICKR_API_KEY}
    FLICKR_USER_ID: ${self:custom.FLICKR_USER_ID}
  events:
    - schedule:
        name: project365-build
        rate: cron(0 */2 * * ? *)
```

Serverless configuration

```
functions:
  update:
    handler: src/actions/update.main
    layers:
      - {Ref: PhpLambdaLayer}
    environment:
      FLICKR_API_KEY: ${self:custom.FLICKR_API_KEY}
      FLICKR_USER_ID: ${self:custom.FLICKR_USER_ID}
  events:
    - schedule:
        name: project365-build
        rate: cron(0 */2 * * ? *)
```

main()

```
function main(array $eventData) : array
{
    $apiKey = getEnvVar('P365_FLICKR_API_KEY');
    $userId = getEnvVar('P365_FLICKR_USER_ID');
    $year = $eventData['year'] ?? date('Y');

    $pageCreator = new PhotoPageCreator($apiKey);
    $html = $pageCreator->update($year, $userId);
    $uploader = new Uploader($cloudFrontId);
    $uploader->uploadOne($year, $html, $s3Bucket);
    $uploader->invalidateCache(['/' . $year]);
}
```

main()

```
function main(array $eventData) : array
{
    $apiKey = getEnvVar('P365_FLICKR_API_KEY');
    $userId = getEnvVar('P365_FLICKR_USER_ID');
    $year = $eventData['year'] ?? date('Y');

    $pageCreator = new PhotoPageCreator($apiKey);
    $html = $pageCreator->update($year, $userId);
    $uploader = new Uploader($cloudFrontId);
    $uploader->uploadOne($year, $html, $s3Bucket);
    $uploader->invalidateCache(['/' . $year]);
}
```

main()

```
function main(array $eventData) : array
{
    $apiKey = getEnvVar('P365_FLICKR_API_KEY');
    $userId = getEnvVar('P365_FLICKR_USER_ID');
    $year = $eventData['year'] ?? date('Y');

    $pageCreator = new PhotoPageCreator($apiKey);
    $html = $pageCreator->update($year, $userId);
    $uploader = new Uploader($cloudFrontId);
    $uploader->uploadOne($year, $html, $s3Bucket);
    $uploader->invalidateCache(['/' . $year]);
}
```

main()

```
function main(array $eventData) : array
{
    $apiKey = getEnvVar('P365_FLICKR_API_KEY');
    $userId = getEnvVar('P365_FLICKR_USER_ID');
    $year = $eventData['year'] ?? date('Y');

    $pageCreator = new PhotoPageCreator($apiKey);
    $html = $pageCreator->update($year, $userId);
    $uploader = new Uploader($cloudFrontId);
    $uploader->uploadOne($year, $html, $s3Bucket);
    $uploader->invalidateCache(['/' . $year]);
}
```

main()

```
function main(array $eventData) : array
{
    $apiKey = getEnvVar('P365_FLICKR_API_KEY');
    $userId = getEnvVar('P365_FLICKR_USER_ID');
    $year = $eventData['year'] ?? date('Y');

    $pageCreator = new PhotoPageCreator($apiKey);
    $html = $pageCreator->update($year, $userId);
    $uploader = new Uploader($cloudFrontId);
    $uploader->uploadOne($year, $html, $s3Bucket);
    $uploader->invalidateCache(['/' . $year]);
}
```

Fetch photos from Flickr

```
$url = '?' . http_build_query([
    'api_key' => $this->flickrApiKey,
    'user_id' => $flickrUserId,
    'extras' => 'url_z, date_taken, owner_name',
    'method' => 'flickr.photos.search',
    'tags' => $year,
]);

$response = $this->client->get($url);
$data = json_decode($response->getBody(), true);
return $data['photos'];
```

Fetch photos from Flickr

```
$url = '?' . http_build_query([
    'api_key' => $this->flickrApiKey,
    'user_id' => $flickrUserId,
    'extras' => 'url_z, date_taken, owner_name',
    'method' => 'flickr.photos.search',
    'tags' => $year,
]);

$response = $this->client->get($url);
$data = json_decode($response->getBody(), true);
return $data['photos'];
```

Fetch photos from Flickr

```
$url = '?' . http_build_query([
    'api_key' => $this->flickrApiKey,
    'user_id' => $flickrUserId,
    'extras' => 'url_z, date_taken, owner_name',
    'method' => 'flickr.photos.search',
    'tags' => $year,
]);

$response = $this->client->get($url);
$data = json_decode($response->getBody(), true);
return $data['photos'];
```

Upload to S3

```
$s3 = new S3Client([  
    'version' => 'latest',  
    'region'  => getenv('AWS_DEFAULT_REGION')  
]);
```

```
$s3->putObject([  
    'Bucket' => $bucketName,  
    'ACL'    => 'public-read',  
    'Key'    => "$year.html",  
    'Body'   => $data,  
    'ContentType' => 'text/html',  
]);
```

Upload to S3

```
$s3 = new S3Client([  
    'version' => 'latest',  
    'region'  => getenv('AWS_DEFAULT_REGION')  
]);
```

```
$s3->putObject([  
    'Bucket' => $bucketName,  
    'ACL'    => 'public-read',  
    'Key'    => "$year.html",  
    'Body'   => $data,  
    'ContentType' => 'text/html',  
]);
```

Upload to S3

```
$s3 = new S3Client([  
    'version' => 'latest',  
    'region'  => getenv('AWS_DEFAULT_REGION')  
]);
```

```
$s3->putObject([  
    'Bucket' => $bucketName,  
    'ACL'    => 'public-read',  
    'Key'    => "$year.html",  
    'Body'   => $data,  
    'ContentType' => 'text/html',  
]);
```

Invalidate CloudFront

```
$cft = new CloudFrontClient([ .. ]);

$result = $cft->createInvalidation([
    'DistributionId' => $cloudFrontId,
    'InvalidationBatch' => [
        'CallerReference' => date('YmdHis'),
        'Paths' => [
            'Items' => ["/$year.html"],
            'Quantity' => 1,
        ],
    ],
]);
```

Invalidate CloudFront

```
$cft = new CloudFrontClient([ .. ]);

$result = $cft->createInvalidation([
    'DistributionId' => $cloudFrontId,
    'InvalidationBatch' => [
        'CallerReference' => date('YmdHis'),
        'Paths' => [
            'Items' => ["/$year.html"],
            'Quantity' => 1,
        ],
    ],
]);
```

Invalidate CloudFront

```
$cft = new CloudFrontClient([ .. ]);

$result = $cft->createInvalidation([
    'DistributionId' => $cloudFrontId,
    'InvalidationBatch' => [
        'CallerReference' => date('YmdHis'),
        'Paths' => [
            'Items' => ["/$year.html"],
            'Quantity' => 1,
        ],
    ],
]);
```

Invalidate CloudFront

```
$cft = new CloudFrontClient([ .. ]);

$result = $cft->createInvalidation([
    'DistributionId' => $cloudFrontId,
    'InvalidationBatch' => [
        'CallerReference' => date('YmdHis'),
        'Paths' => [
            'Items' => ["/$year.html"],
            'Quantity' => 1,
        ],
    ],
]);
```

The finished website



To sum up

Resources

- <https://akrabat.com>
- <https://www.martinfowler.com/articles/serverless.html>
- <https://github.com/akrabat/ow-php-todo-backend>
- <https://github.com/akrabat/project365-photos-website>
- <http://www.openwhisk.org>
- <https://aws.amazon.com/lambda/>
- <https://bref.sh>

Thank you!

<https://joind.in/talk/e6b2a>