The Serverless PHP Application

Rob Allen

PHPFest, October 2020

Serverless

A Serverless solution is one that costs you nothing to run if nobody is using it

... excluding data storage costs.

Paul D. Johnston

... as a Service

Storage *as a Service*
Database *as a Service*
Cache *as a Service*
Auth *as a Service*
Search *as a Service*
Function *as a Service*

Function as a Service

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

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)

Challenges

Challenges

- Start up latency

Challenges

- Start up latency
- Time limit

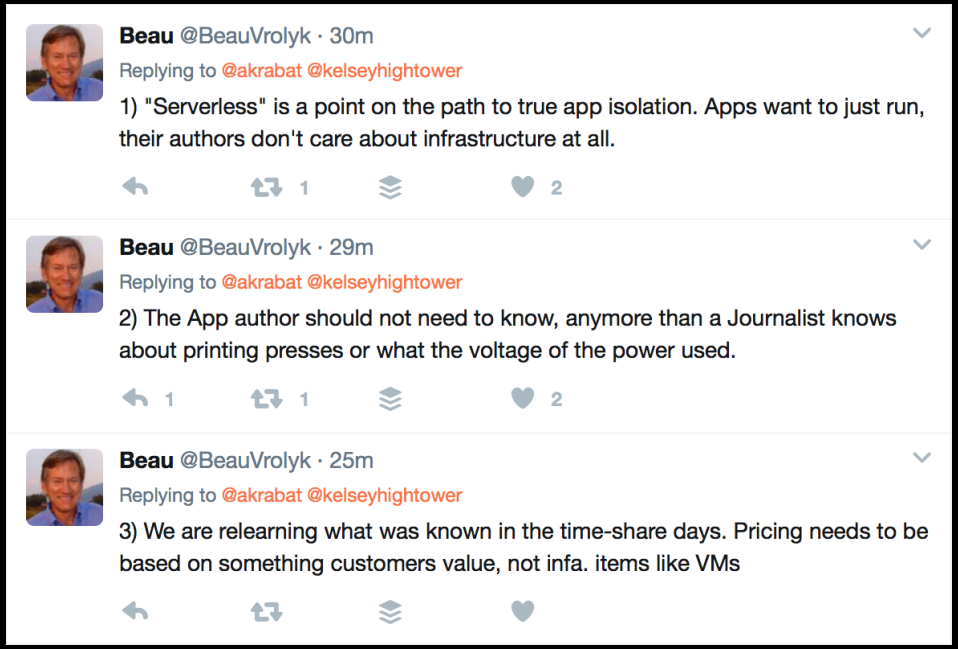
Challenges

- 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



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?

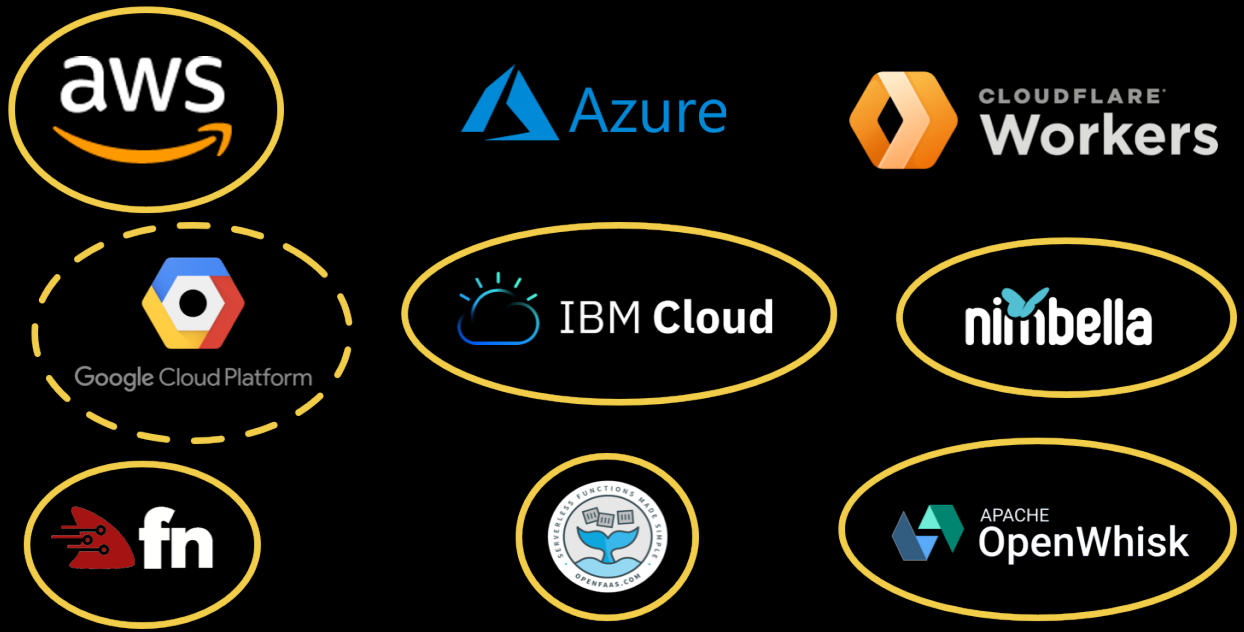
- Responding to web hooks
- PWA/Static site contact form, et al.
- Additional features without extending current platform
- Variable traffic levels

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

Serverless platforms with PHP support



APACHE
OpenWhisk


JS

.NET
Core


php



GO

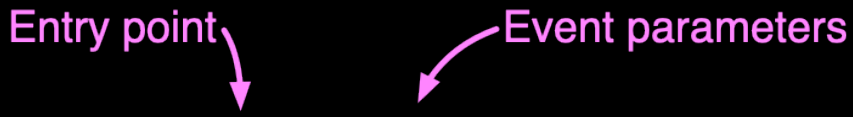
Java


JS
php
GO
Java

The anatomy of an action

```php
function main(array $args) : array
{
    // Marshall inputs from event parameters
    $name = $args['name'] ?? 'world';
    // Do the work
    $message = 'Hello ' . $name
    // Return result
    return ["message" => $message];
}
```

Hello World




```php
function main(array $args) : array
{
    // Marshall inputs from event parameters
    $name = $args['name'] ?? 'world';
    // Do the work
    $message = 'Hello ' . $name
    // Return result
    return ["message" => $message];
}
```

Hello World

```
function main(array $args) : array
{
    // Marshall inputs from event parameters
    $name = $args['name'] ?? 'world';
    // Do the work
    $message = 'Hello ' . $name
    // Return result
    return ["message" => $message];
}
```

Hello World

```
function main(array $args) : array
{
    // Marshall inputs from event parameters
    $name = $args['name'] ?? 'world';
    // Do the work
    $message = 'Hello ' . $name
    // Return result
    return ["message" => $message];
}
```

Hello World

```
function main(array $args) : array
{
    // Marshall inputs from event parameters
    $name = $args['name'] ?? 'world';
    // Do the work
    $message = 'Hello ' . $name
    // Return result
    return ["message" => $message];
}
```

Upload your action

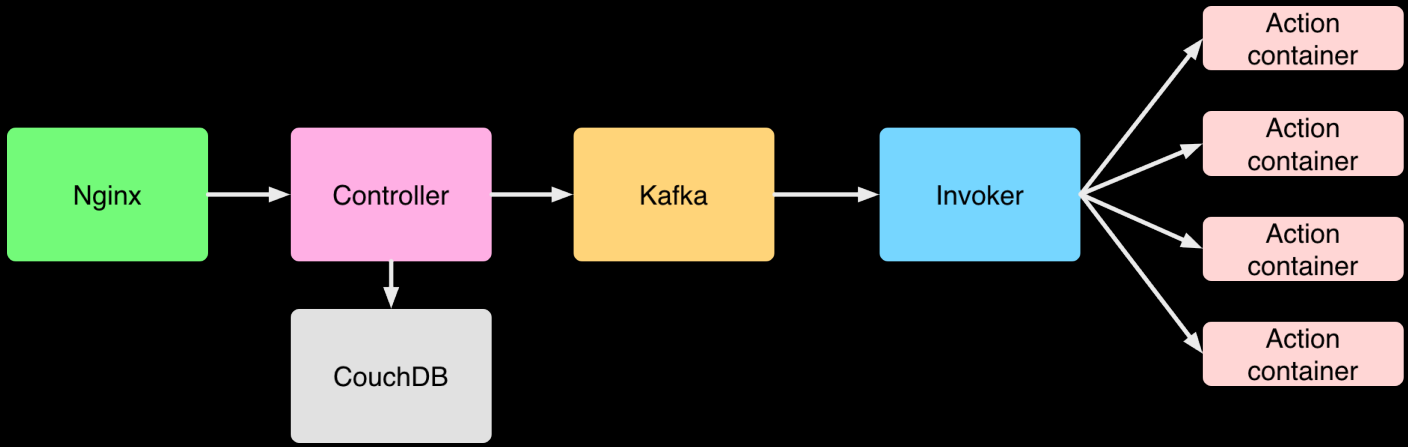
```
$ wsk action update hello hello.php
ok: updated action hello
```

Run your action

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

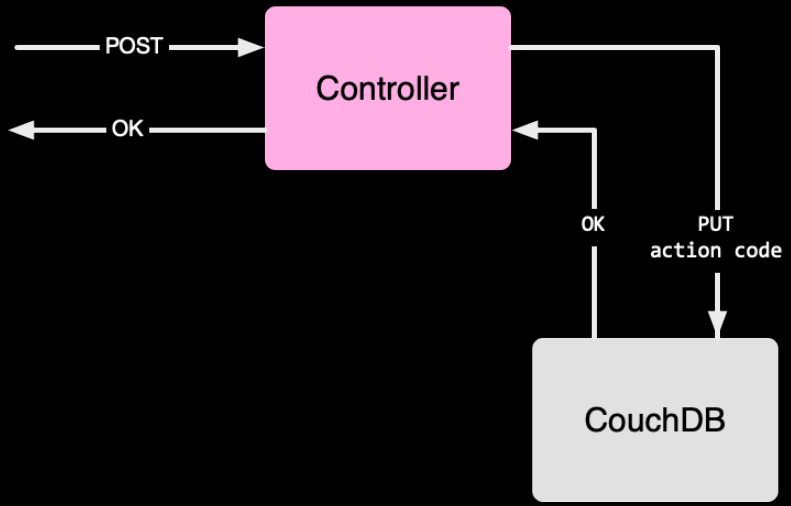
Segue: How did it do this?

OpenWhisk's architecture



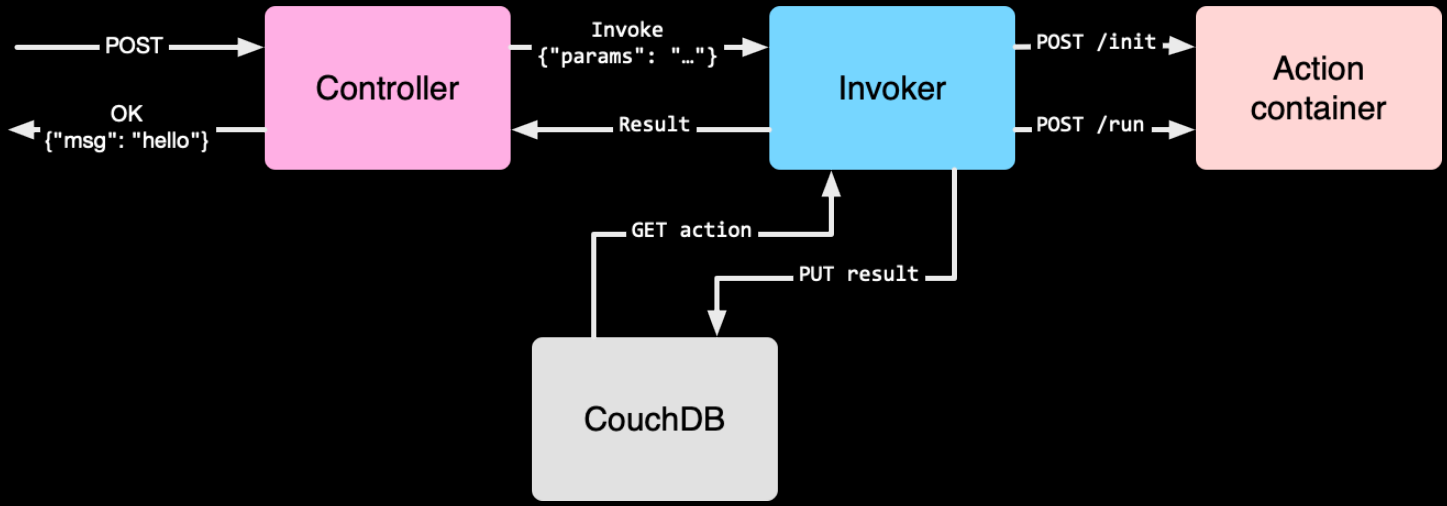
Create an action

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



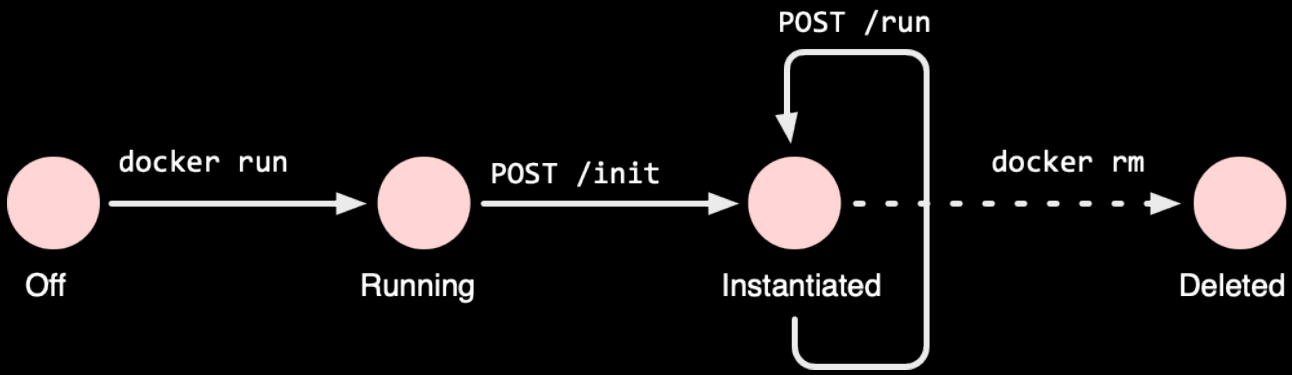
Invoke an action

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



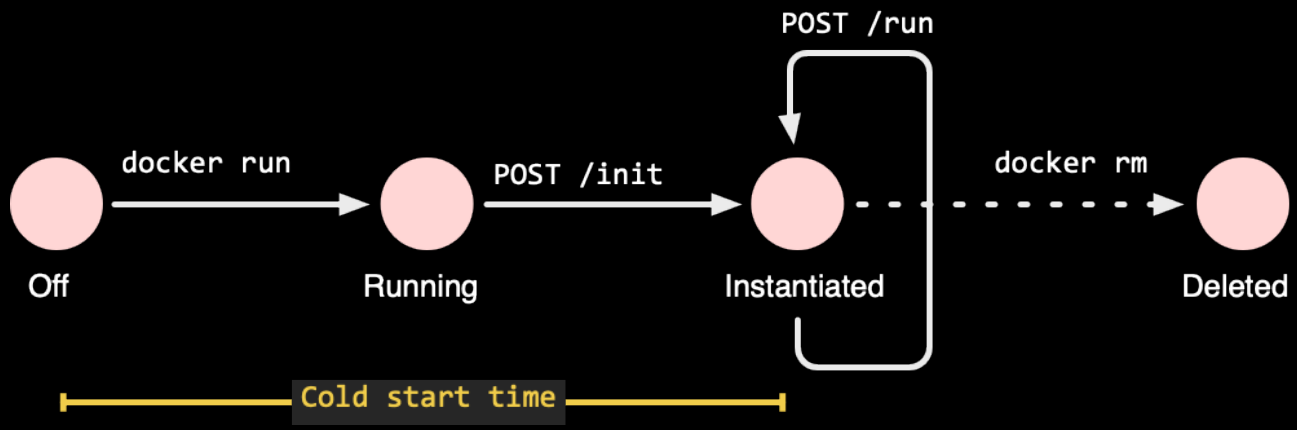
Action container lifecycle

- 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

Turn it into an API

Add the `--web` flag:

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

Turn it into an API

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
- Custom domains
- Rate limiting
- Security (API keys, OAuth, CORS)
- Analytics

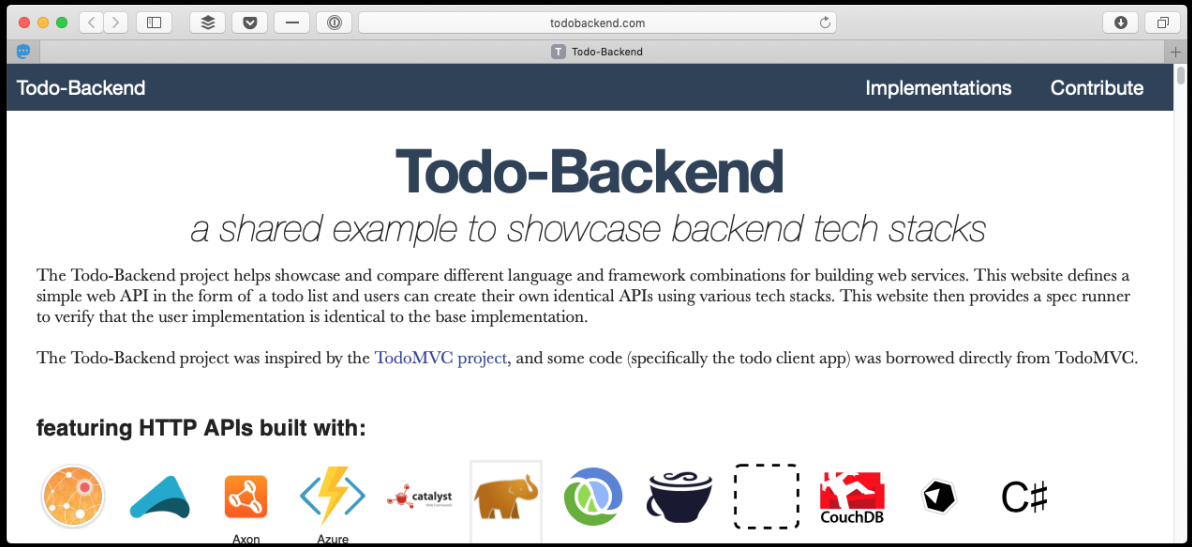
API Gateway


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

API Gateway

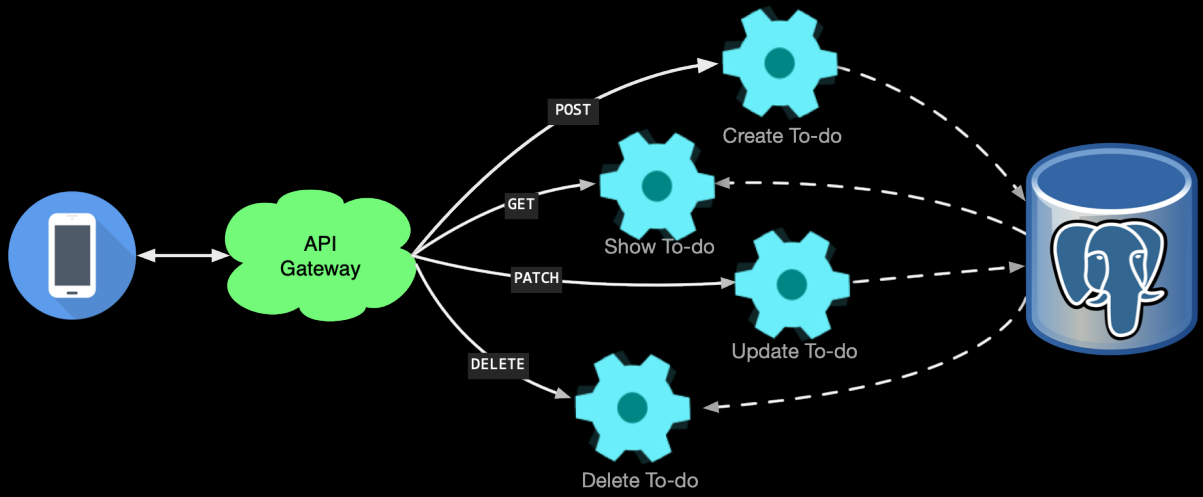

```
$ curl https://ow.akrabat.com/demo/hello?name=Rob
{
  "message": "Hello Rob!"
}
```

A Serverless API



Todo-Backend

An OpenWhisk PHP implementation of a to-do list API



Serverless Framework

Deployment tooling for serverless applications

`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

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

```
src/
├── Todo/
│   ├── Todo.php
│   ├── TodoMapper.php
│   └── TodoTransformer.php
├── actions/
│   ├── addTodo.php
│   ├── deleteTodo.php
│   ├── editTodo.php
│   ├── listTodos.php
│   └── showTodo.php
└── AppContainer.php
```

editTodo.php

```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

```php
$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

```php
$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(),
];
```

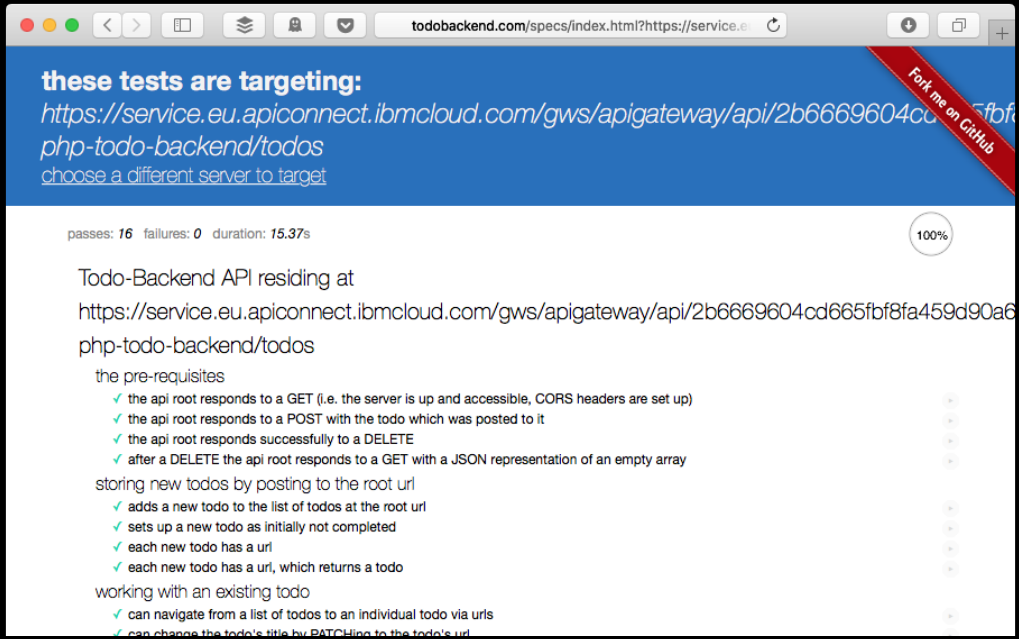
editTodo.php: Present results

```php
$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...
[...]
```

A working API!




aws

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!

bref

running PHP made simple

Everything you need to easily
deploy and run serverless PHP applications.

Bref

- Maintained PHP runtimes for AWS Lambda
- Deployment via Serverless Framework
- Great documentation!

Bref PHP function

```php
<?php declare(strict_types=1);

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

return function($event) {
    return 'Hello ' . ($event['name'] ?? 'world');
}
```

serverless.yml

```
service: helloapp
provider:
    name: aws
    runtime: provided

functions:
    hello:
        handler: index.php
        layers:
            - ${bref:layer.php-74}
```

serverless.yml

```
service: helloapp
provider:
    name: aws
    runtime: provided

functions:
    hello:
        handler: index.php
        layers:
            - ${bref:layer.php-74}
```

serverless.yml

```
service: helloapp
provider:
    name: aws
    runtime: provided

functions:
    hello:
        handler: index.php
        layers:
            - ${bref:layer.php-74}
```

Deploy

```
$ serverless deploy
Serverless: Packaging service...
...
Serverless: Stack update finished...

Service Information
service: helloapp
stage: dev
stack: helloapp-dev
functions:
  hello: helloapp-dev-hello
```

Run

```
$ serverless invoke -f hello
"Hello world"
```

Run

```
$ serverless invoke -f hello
"Hello world"


$ serverless invoke -f hello -d '{"name": "Rob"}'
"Hello Rob"
```

Run locally in Docker

```
$ serverless invoke local --docker -f hello
Serverless: Building Docker image...
...
REPORT RequestId: 6a653a94-ee51-1f3e-65c7-1f1954842f29
  Init Duration: 265.93 ms Duration: 145.37 ms
  Billed Duration: 200 ms Memory Size: 1024 MB
  Max Memory Used: 27 MB

"Hello world"
```

Xdebug also works!

Add AWS API Gateway

serverless.yml:

```
functions:
    hello:
        handler: index.php
        ...
        events:
            - http: "GET /hello"
            - http: "GET /hi/{name}"
```

Return a PSR-15 RequestHandler

```php
class HelloHandler implements RequestHandlerInterface
{
  public function handle(ServerRequest $request): Response
  {
    $name = ($request->getQueryParams()['name'] ?? 'world');
    $body = 'Hello ' . $name;

    return new Response(200,
      ['Content-Type' => 'text/plain'],
      $body);
  }
}
```

Deploy

```
$ serverless deploy
Serverless: Packaging service...
...

Service Information
service: helloapp
stage: dev
stack: helloapp-dev
endpoints:
  GET - https://l1v6cz13zb.execute-api.eu-west-2
    .amazonaws.com/dev/hello
```

Test

```
$ curl -i https://l1v6cz...naws.com/dev/hello?name=Rob
HTTP/2 200
content-type: text/plain
content-length: 9

Hello Rob
```

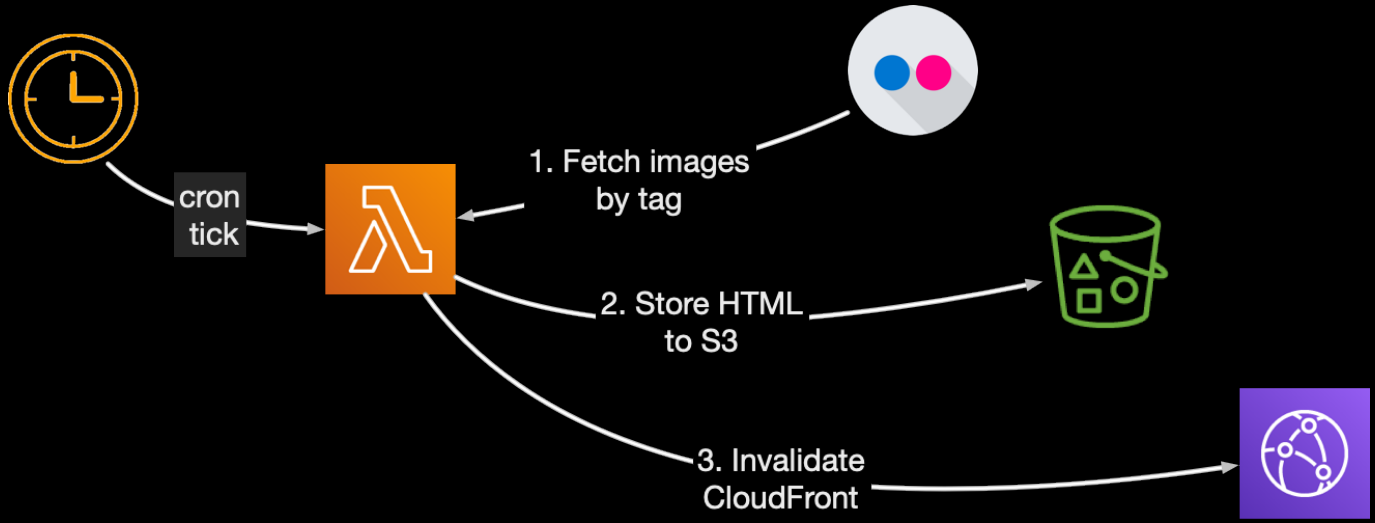
Project 365
My photo-a-day website

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: index.php
    events:
      - schedule:
          name: project365-build
          rate: cron(0 */2 * * ? *)
```

Serverless configuration

```
functions:
  update:
    handler: index.php
    events:
      - schedule:
          name: project365-build
          rate: cron(0 */2 * * ? *)
```

Serverless configuration

```
functions:
  update:
    handler: index.php
    events:
      - schedule:
          name: project365-build
          rate: cron(0 */2 * * ? *)
```

main()

```
function main(array $eventData) : array
{
  $year = $eventData['year'] ?? date('Y');

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

main()

```
function main(array $eventData) : array
{
  $year = $eventData['year'] ?? date('Y');

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

main()

```
function main(array $eventData) : array
{
  $year = $eventData['year'] ?? date('Y');

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

main()

```
function main(array $eventData) : array
{
  $year = $eventData['year'] ?? date('Y');

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

main()

```
function main(array $eventData) : array
{
  $year = $eventData['year'] ?? date('Y');

  $pageCreator = new PhotoPageCreator();
  $html = $pageCreator->update($year);
  $uploader = new Uploader($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'];
```

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,
        ],
    ],
]);
```

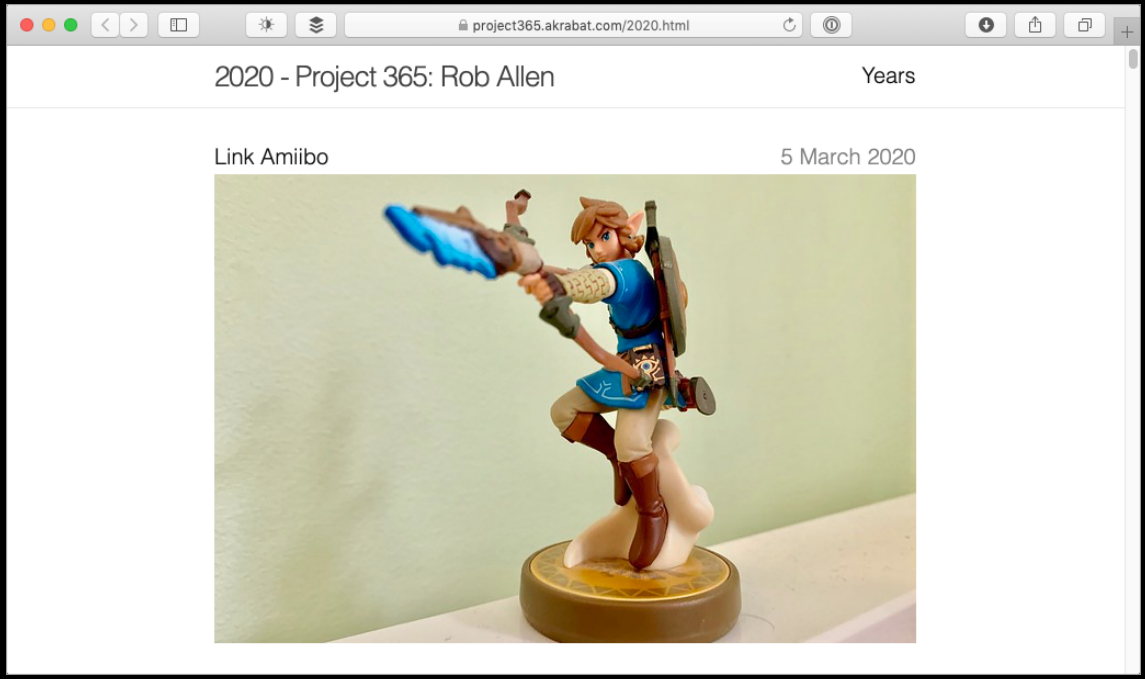
Invalidate CloudFront

```
$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!

Rob Allen - http://akrabat.com - @akrabat