

Serverless PHP Applications

Rob Allen, April 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



Beau @BeauVrolyk · 30m

Replies 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.



1



2



Beau @BeauVrolyk · 29m

Replies 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



2



Beau @BeauVrolyk · 25m

Replies 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 infra. 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

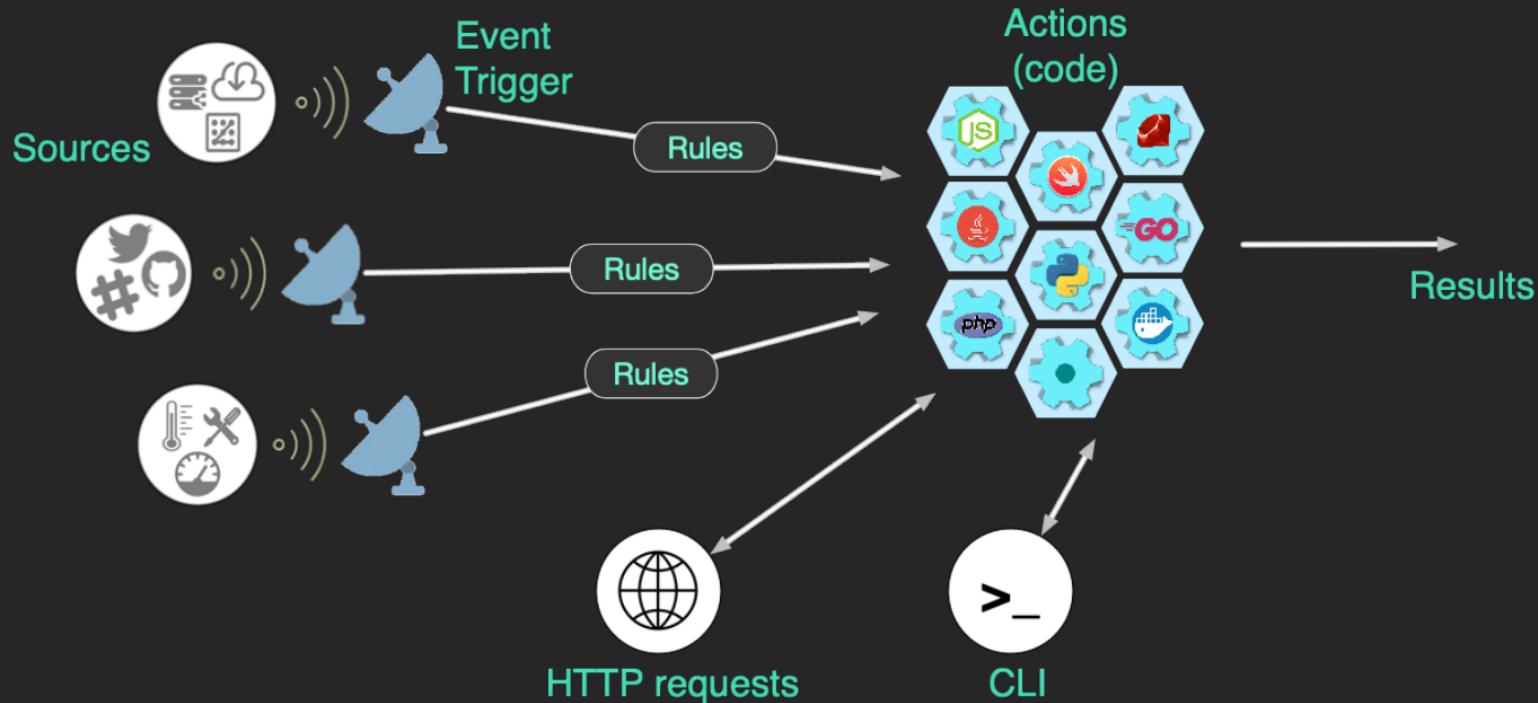


IBM Cloud





Concepts







php

The PHP logo, consisting of the lowercase letters "php" in a bold, black, italicized font. The letters are partially overlaid by a large, semi-transparent blue oval.

java

The word "java" in a red, sans-serif font, positioned at the bottom of the image.

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

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

Entry point

Event parameters

Service result

The image shows annotations for a PHP code snippet. A pink arrow points from the text 'Entry point' to the opening tag '<?php'. Another pink arrow points from the text 'Event parameters' to the parameter '\$args' in the function signature. A third pink arrow points from the text 'Service result' to the return value of the function, which is an array containing a single element.

Upload your action

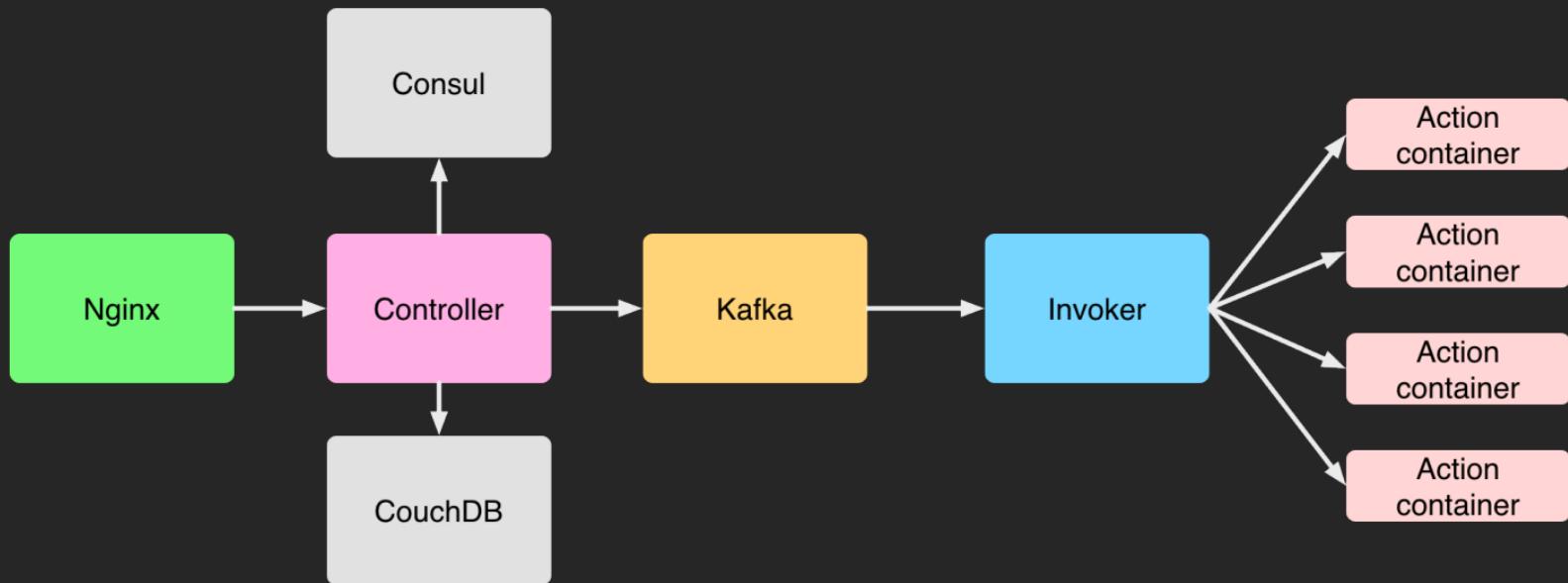
```
$ wsk action update hello hello.php  
ok: updated action hello
```

Run your action

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

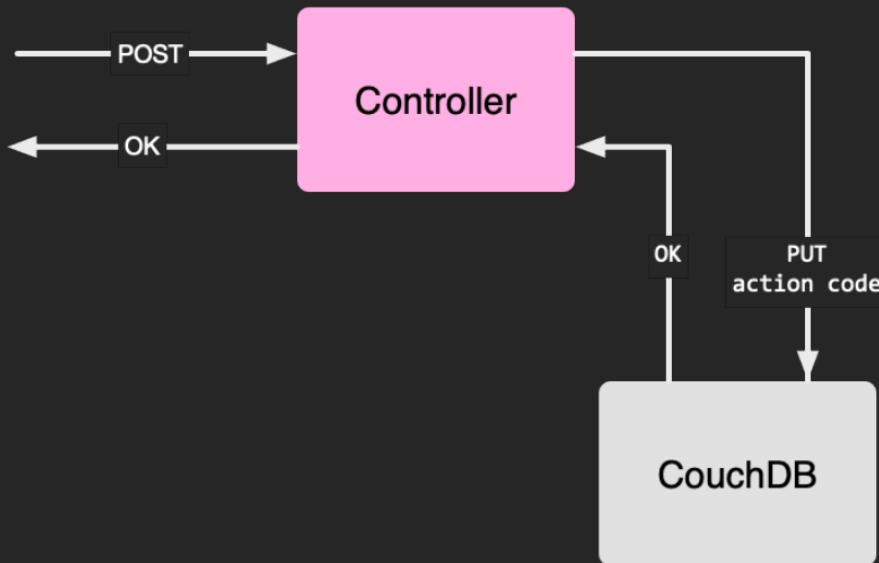
Segue: How did it do this?

OpenWhisk's architecture



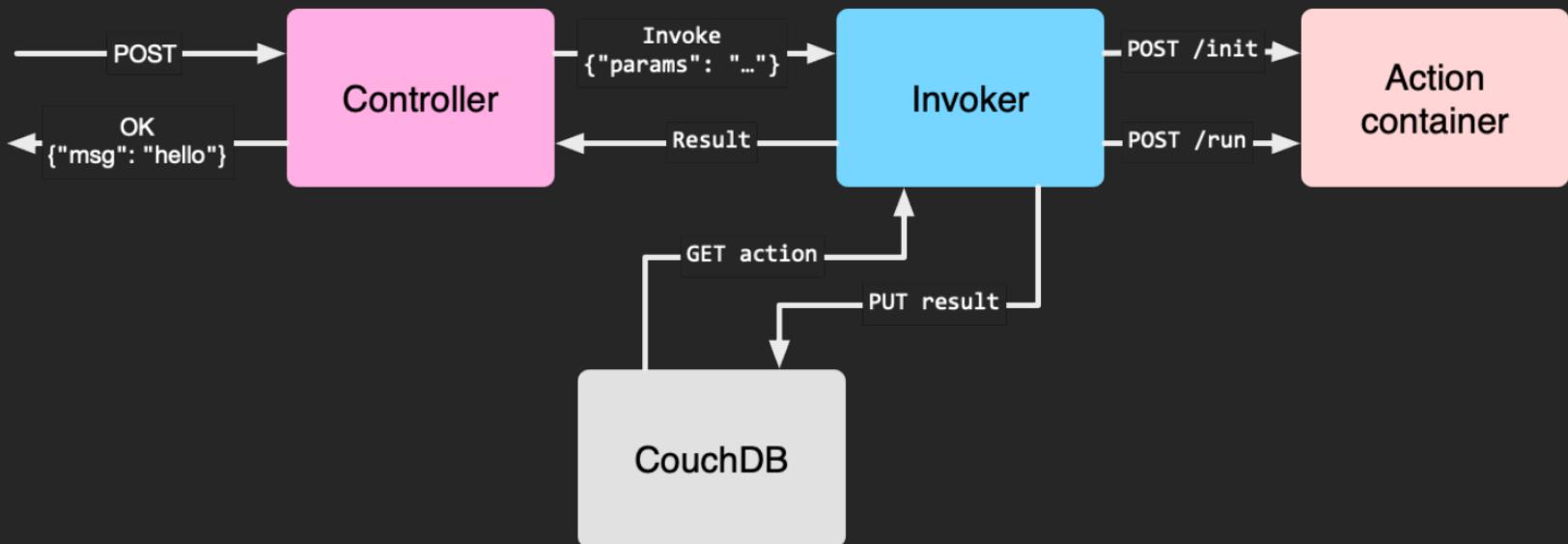
Create an action

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



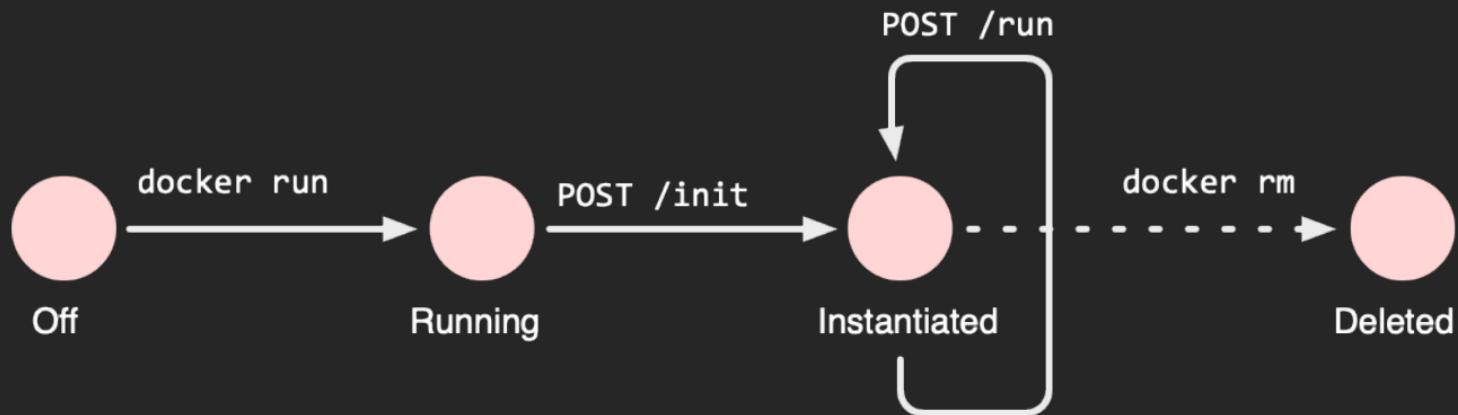
Invoke an action

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



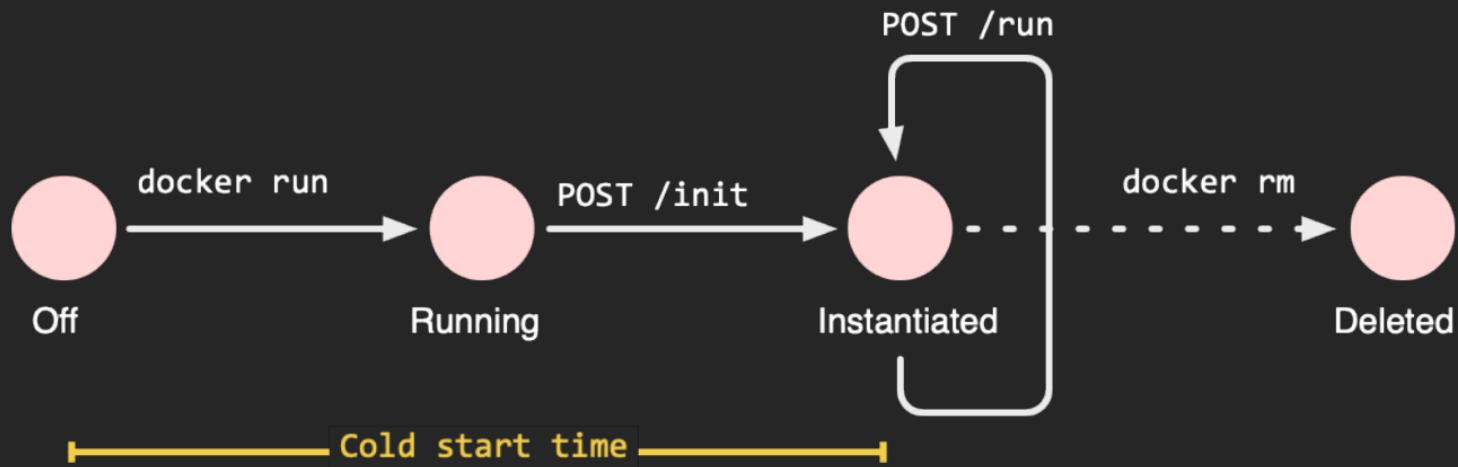
Action container lifecycle

- 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



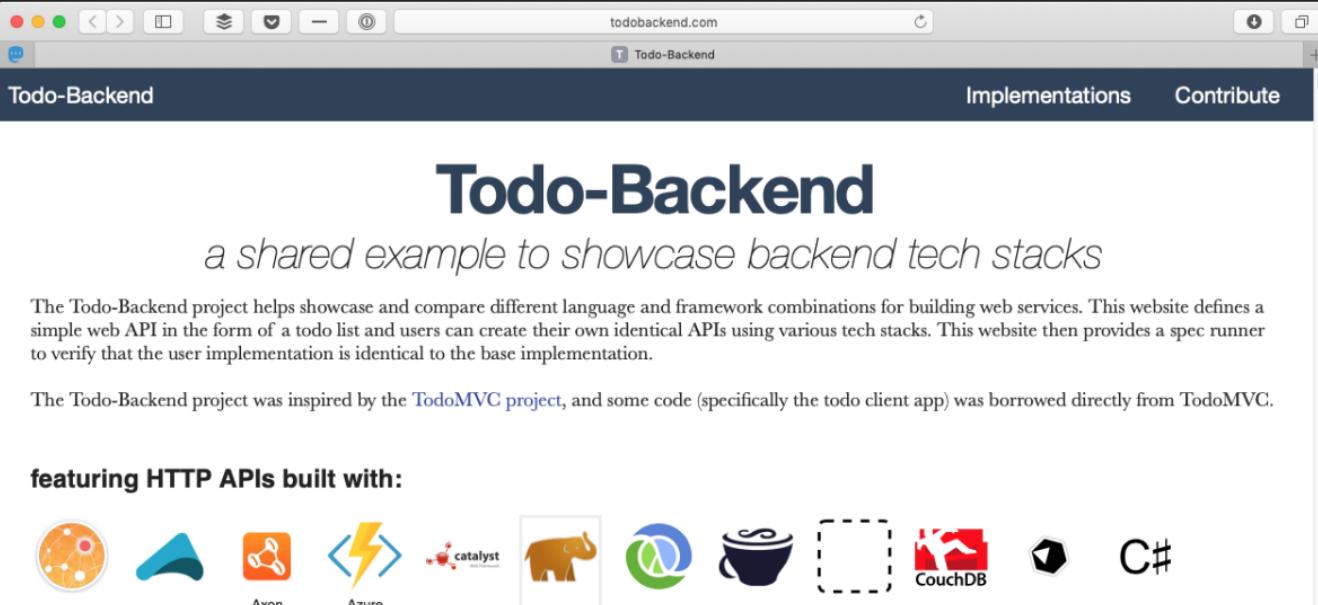
```
$ wsk api create /demo /hello GET hello  
ok: created API /demo/hello GET for action /_/hello
```

API Gateway



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

A Serverless API



The screenshot shows a web browser window displaying the Todo-Backend website at todobackend.com. The page title is "Todo-Backend". The main heading is "Todo-Backend" with the subtitle "a shared example to showcase backend tech stacks". Below this, there is a paragraph of text explaining the project's purpose: "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." Another paragraph states: "The Todo-Backend project was inspired by the [TodoMVC project](#), and some code (specifically the todo client app) was borrowed directly from TodoMVC." At the bottom, there is a section titled "featuring HTTP APIs built with:" followed by a row of icons representing various technology stacks: Axon, Azure, catalyst, CouchDB, C#, and others.

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.

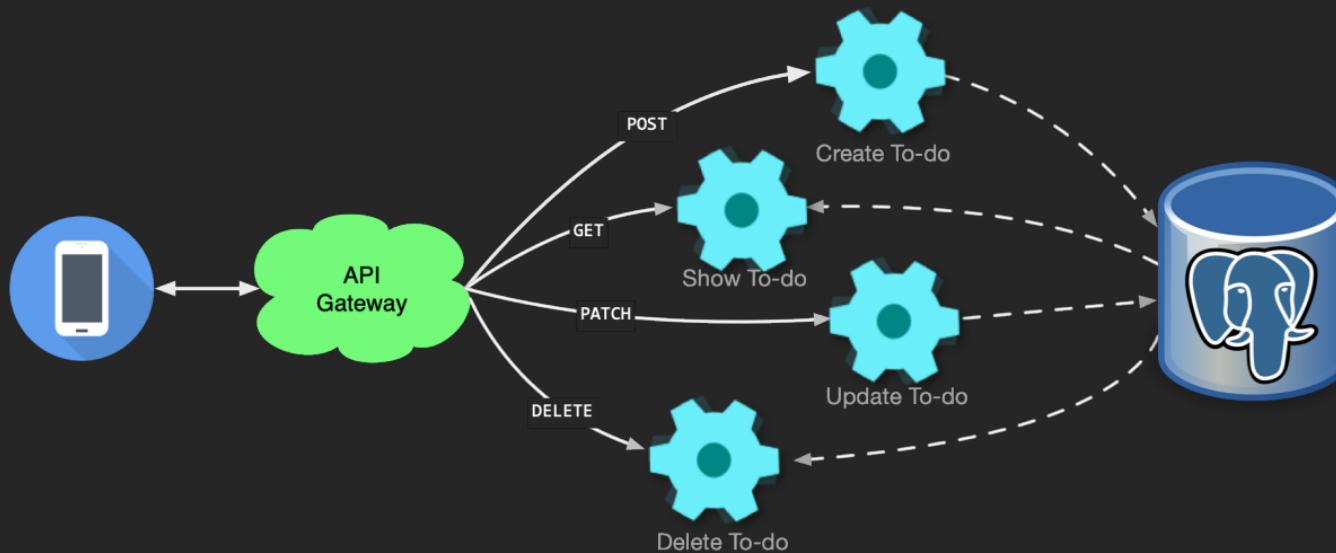
The Todo-Backend project was inspired by the [TodoMVC project](#), and some code (specifically the todo client app) was borrowed directly from TodoMVC.

featuring HTTP APIs built with:

- Axon
- Azure
- catalyst
- CouchDB
- C#

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

```
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(),
];
```

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!

The screenshot shows a web browser window with the URL todobackend.com/specs/index.html?https://service.eu.apiconnect.ibmcloud.com/gws/apigateway/api/2b6669604cd665fbf8fa459d90a6. The page title is "these tests are targeting:". Below the title, the target URL is listed as <https://service.eu.apiconnect.ibmcloud.com/gws/apigateway/api/2b6669604cd665fbf8fa459d90a6> and the endpoint `php-todo-backend/todos`. A link to "choose a different server to target" is also present. On the right side of the page, there is a red diagonal banner with the text "Fork me on GitHub". The main content area displays the test results: "passes: 16 failures: 0 duration: 15.37s" and a progress bar indicating 100%. Below this, the "Todo-Backend API" is described as residing at the same URL and endpoint. The "pre-requisites" section lists several successful assertions, such as "the api root responds to a GET (i.e. the server is up and accessible, CORS headers are set up)" and "the api root responds to a POST with the todo which was posted to it". The "storing new todos by posting to the root url" section also lists several successful assertions. The "working with an existing todo" section lists one successful assertion: "can navigate from a list of todos to an individual todo via urls".

these tests are targeting:

<https://service.eu.apiconnect.ibmcloud.com/gws/apigateway/api/2b6669604cd665fbf8fa459d90a6>
`php-todo-backend/todos`
[choose a different server to target](#)

passes: 16 failures: 0 duration: 15.37s

Todo-Backend API residing at
<https://service.eu.apiconnect.ibmcloud.com/gws/apigateway/api/2b6669604cd665fbf8fa459d90a6>
`php-todo-backend/todos`

the pre-requisites

- ✓ the api root responds to a GET (i.e. the server is up and accessible, CORS headers are set up)
- ✓ the api root responds to a POST with the todo which was posted to it
- ✓ the api root responds successfully to a DELETE
- ✓ after a DELETE the api root responds to a GET with a JSON representation of an empty array

storing new todos by posting to the root url

- ✓ adds a new todo to the list of todos at the root url
- ✓ sets up a new todo as initially not completed
- ✓ each new todo has a url
- ✓ each new todo has a url, which returns a todo

working with an existing todo

- ✓ can navigate from a list of todos to an individual todo via urls
- ✓ can change the todo's title by PATCHing to the todo's url



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 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-73}
```

serverless.yml

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

functions:
  hello:
    handler: index.php
    layers:
      - ${bref:layer.php-73}
```

serverless.yml

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

functions:
  hello:
    handler: index.php
    layers:
      - ${bref:layer.php-73}
```

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

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

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

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

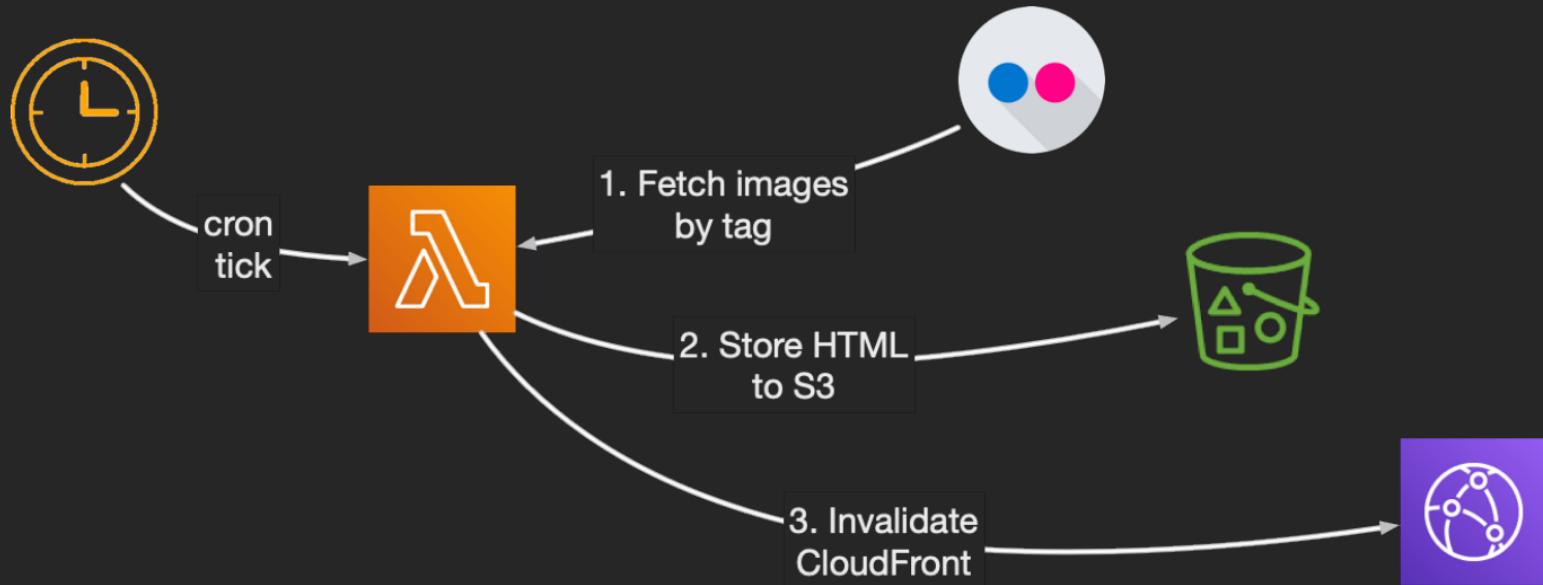
My photo-a-day website

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,  
        ],  
    ],  
]);
```

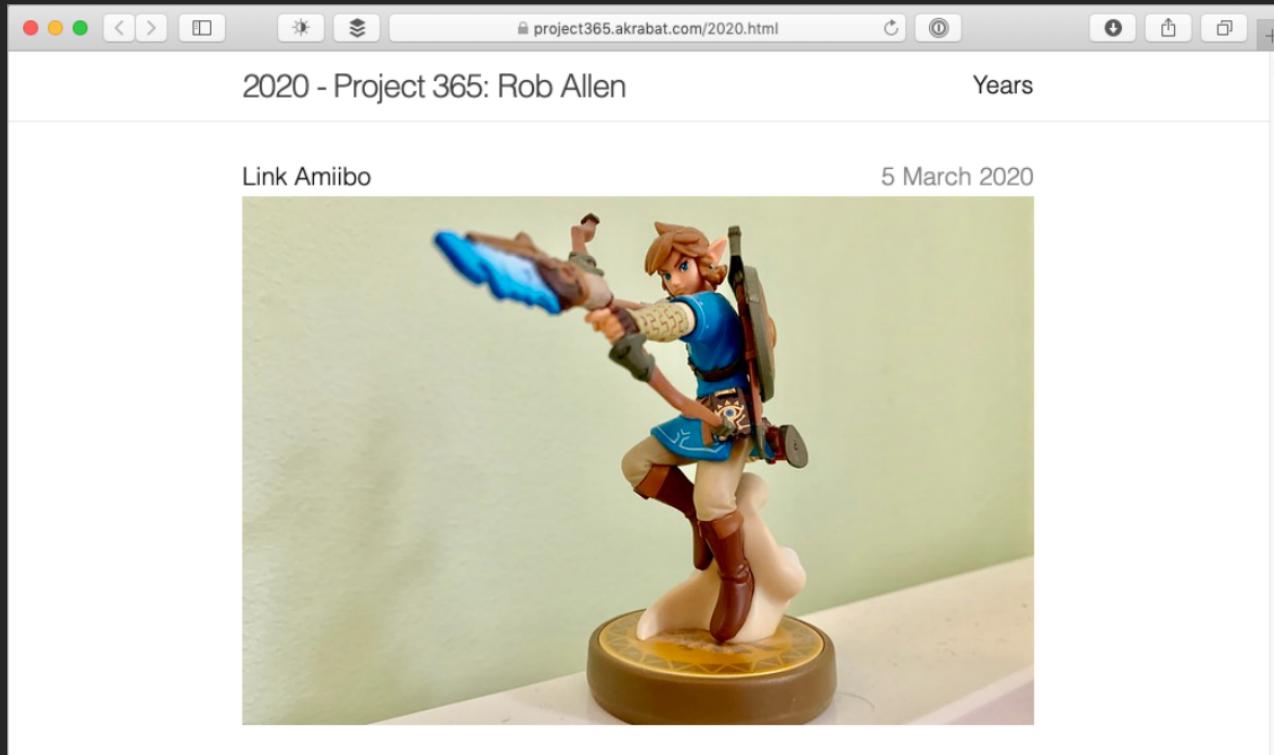
Invalidate CloudFront

```
$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