The Serverless Way

Rob Allen

October 2018

Deployment options

1. Physical servers
2. Virtual machines
3. Containers

Container deployments

1. Platform (e.g. Kubernetes)
2. Application (e.g. Cloud Foundry)
3. Serverless (e.g. OpenWhisk)

Serverless?

The first thing to know about serverless computing is that "serverless" is a pretty bad name to call it.

– Brandon Butler, Network World

AKA: *Functions as a Service*

- Code
- Deployed to the cloud
- Executed in response to an event
- On-demand scaling
- Pay for execution, not when idle

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)

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)

Streaming

- Continuous data flow to be processed

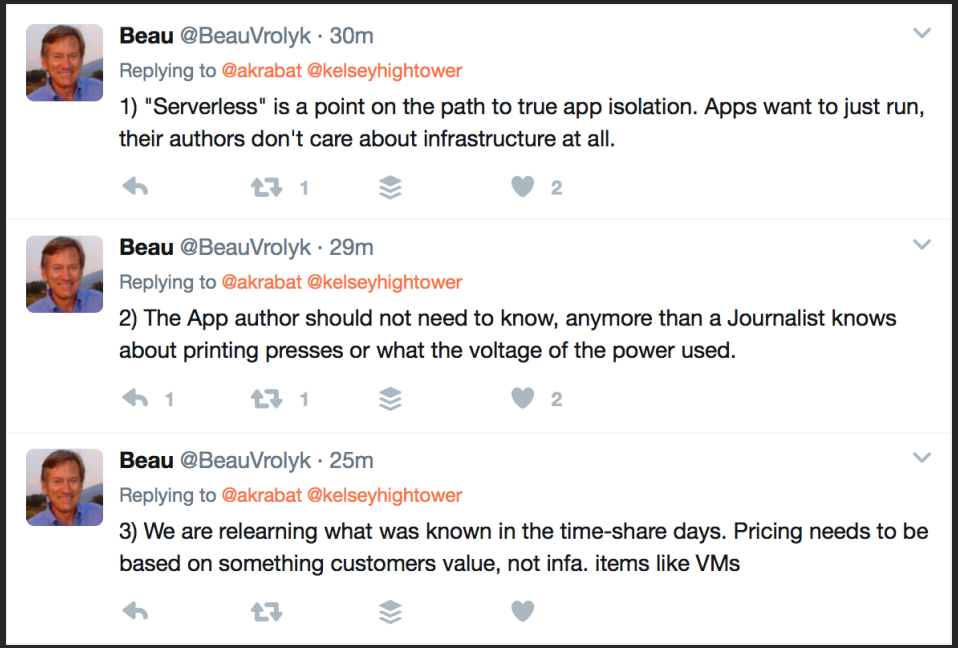
Benefits

- No need to think about servers
- Concentrate on application code
- Pay only for what you use, when you use it
- Language agnostic:
  - NodeJS, Python, Swift, Go, Java, C#, PHP, Ruby, etc

Challenges

- Start up latency
- Time limit
- State is external
- DevOps is still a thing

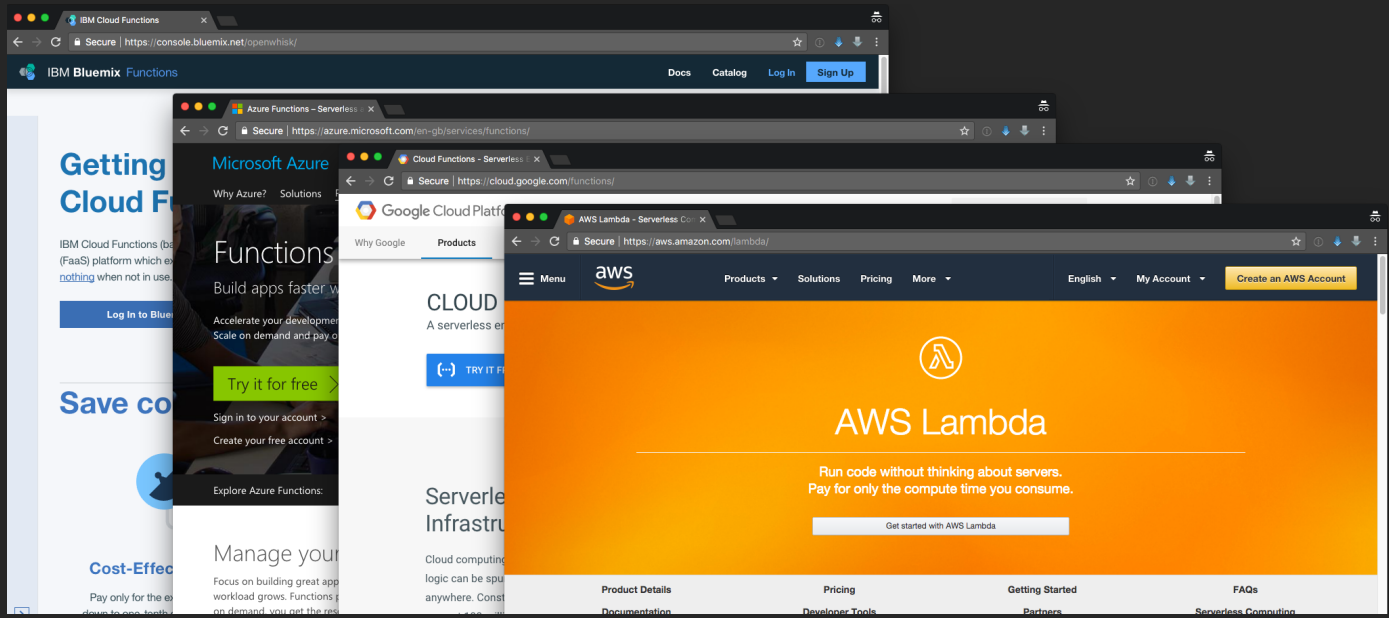
It's about value



When should you use serverless?

- Occasional server needs on a static site
- Variable traffic levels
- Additional compute without extending current platform
- Responding to web hooks
- Any web app that you want to be cheaper to run!

Serverless providers



Hello World

AWS Lambda:

```
def my_handler(event, context):
    name = event.get("name", "World")
    message = 'Hello {}!'.format(name)

    return {'message': message}
```

Hello World

Apache OpenWhisk:

```
def main(args):
    name = args.get("name", "World")
    message = 'Hello {}!'.format(name)

    return {'message': message}
```

Hello World

Google Cloud Functions

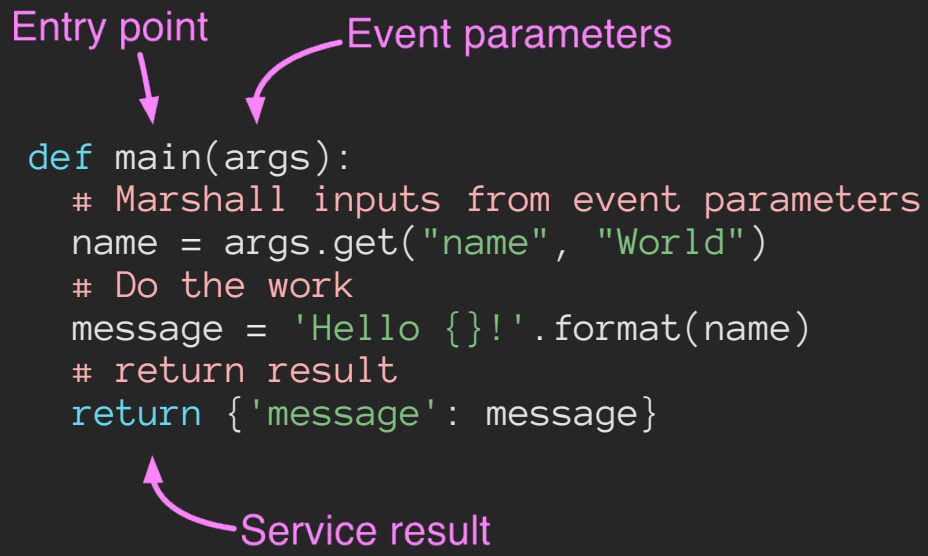
```
def hello_http(request):
    name = request.json.get("name", "World")
    message = 'Hello {}!'.format(name)

    return message
```

Hello World

Azure Cloud Functions

```
import azure.functions as func

def main(req: func.HttpRequest,
         msg: func.Out[func.QueueMessage]) -> str:

    name = req.params.json.get("name", "World")
    message = 'Hello {}!'.format(name)

    msg.set(message)
    return message
```

The anatomy of an action



Creating your action

```
$ wsk action create hello hello.py
ok: updated action hello
```

Running your action

```
$ wsk action create hello hello.py
ok: updated action hello
```

```
$ wsk action invoke hello --result --param name Rob
```

Running your action

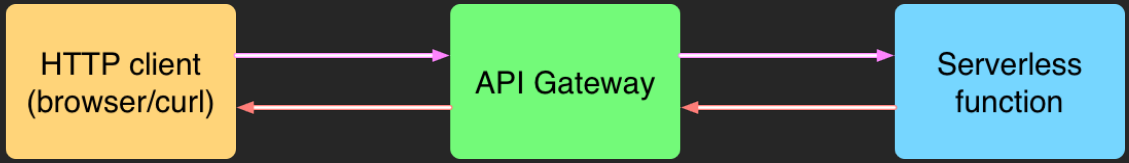
```
$ wsk action create hello hello.py
ok: updated action hello


$ wsk action invoke hello --result --param name Rob
{
    "message": "Hello Rob!"
}
```

Access from the web

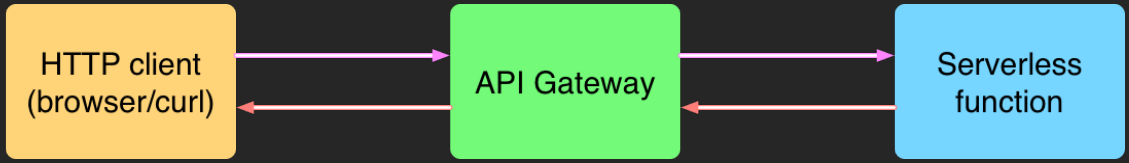
API Gateway provides:

- API routing
- Rate limiting
- Authentication
- Custom domains

API Gateway

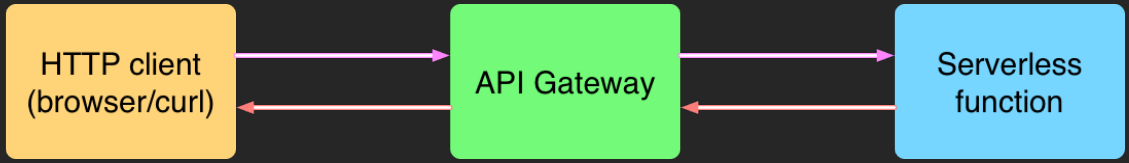


API Gateway



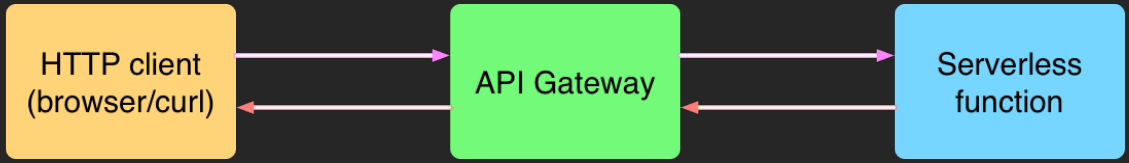

```
$ wsk api create /myapp /hello GET hello
```

API Gateway




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

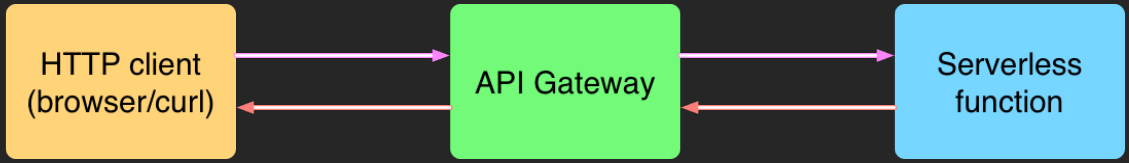
API Gateway




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

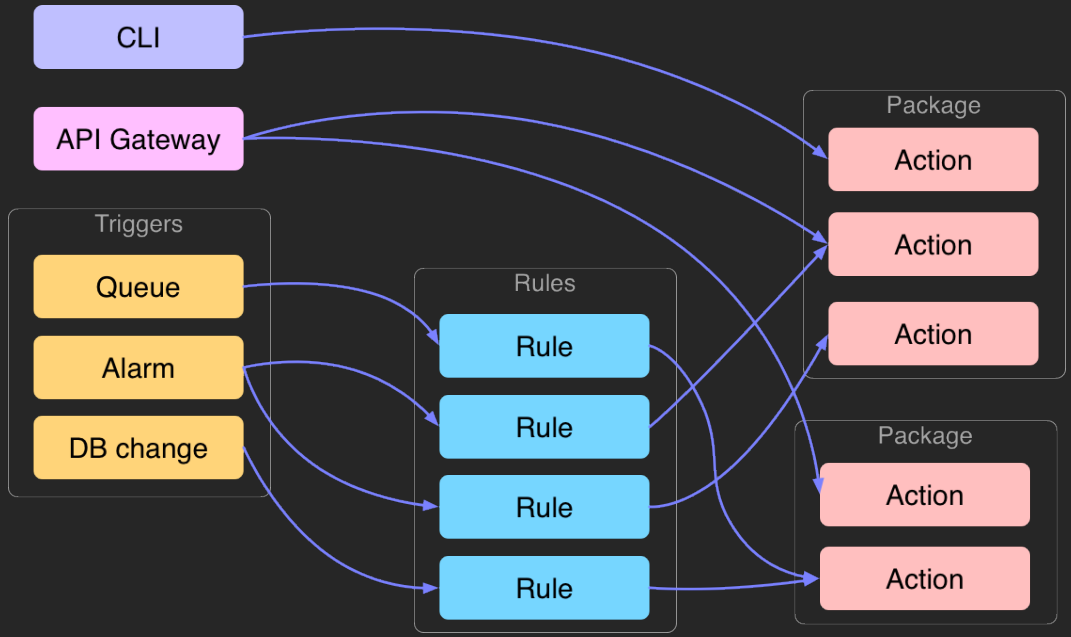
```
$ curl https://example.com/myapp/hello?name=Rob
```

API Gateway



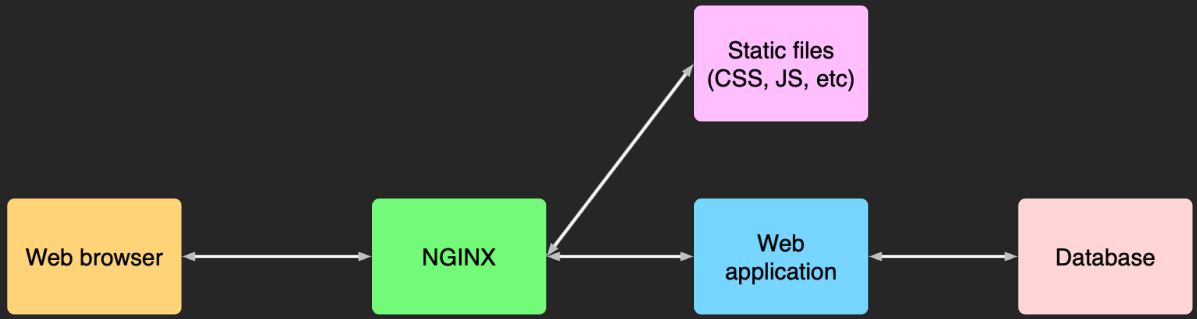

```
$ wsk api create /myapp /hello GET hello
ok: created API /myapp/hello GET for action /_/hello

$ curl https://example.com/myapp/hello?name=Rob
{
  "message": "Hello Rob!"
}
```

Invoking a function

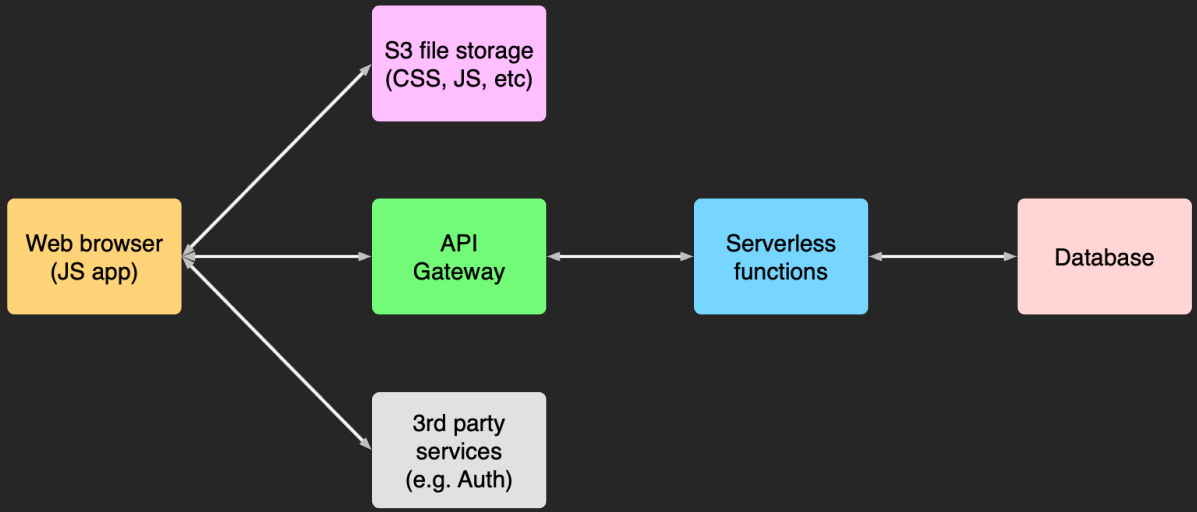


Architecture

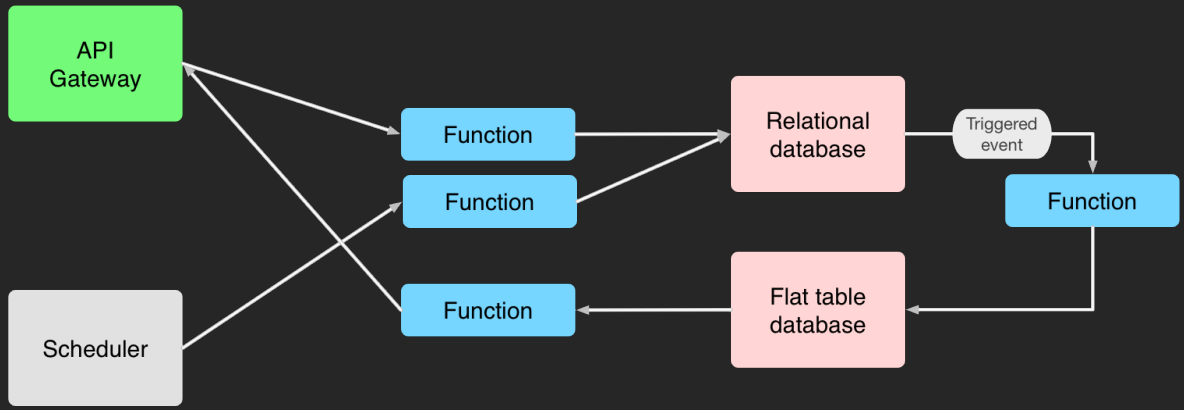
Monolith architecture



Serverless architecture



Serverless architecture pattern



Functions are key

Functions are the Unit of Deployment

Functions are the Unit of Scale

Functions are Stateless

Functions have Structure

Structure

If it's non-trivial, software engineering principles apply!

- Use multiple methods

Structure

If it's non-trivial, software engineering principles apply!

- Use multiple methods
- Use multiple files

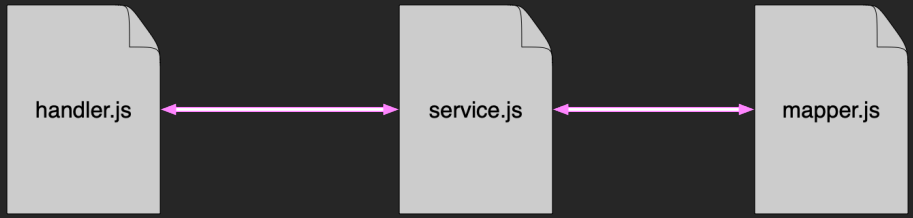
Structure

If it's non-trivial, software engineering principles apply!

- Use multiple methods
- Use multiple files
- Integrate reusable dependencies

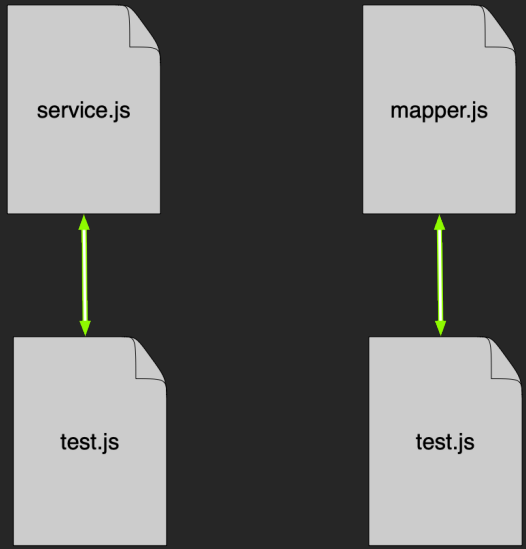
Separation

Separate the action handler from your business logic



Separation

Separate the action handler from your business logic



Example from Adobe CIF-Magento

```
function postCoupon(args) {
  const validator = new InputValidator(args, ERROR_TYPE)
      .checkArguments().mandatoryParameter('id')
  if (validator.error) {
      return validator.buildErrorResponse();
  }

  const cart = new MagentoCart(args, cartMapper.mapCart, 'guest-carts');
  return cart.byId(args.id).addCoupon(args.code).then(function () {
      return cart.byId(args.id).get();
  }).catch(error => {
      return cart.handleError(error);
  });
}
```

Example from Adobe CIF-Magento

```
function postCoupon(args) {
  const validator = new InputValidator(args, ERROR_TYPE)
      .checkArguments().mandatoryParameter('id')
  if (validator.error) {
      return validator.buildErrorResponse();
  }

  const cart = new MagentoCart(args, cartMapper.mapCart, 'guest-carts');
  return cart.byId(args.id).addCoupon(args.code).then(function () {
      return cart.byId(args.id).get();
  }).catch(error => {
      return cart.handleError(error);
  });
}
```

Example from Adobe CIF-Magento

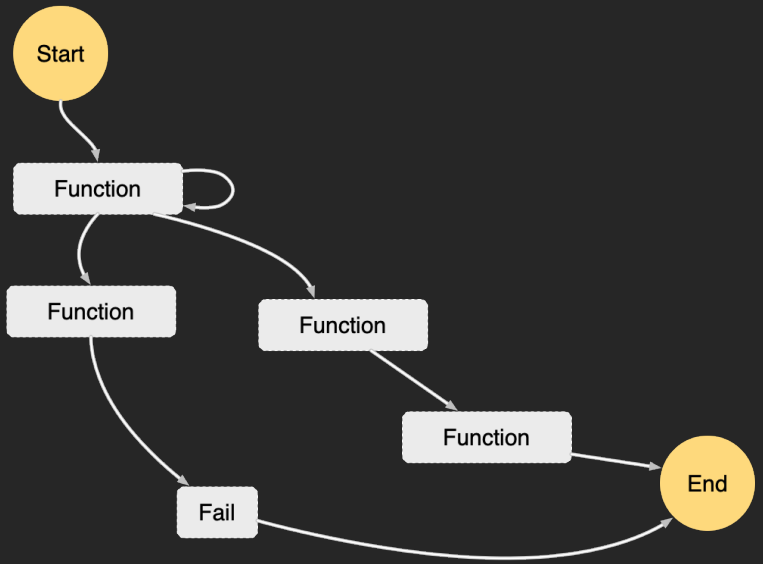
```
function postCoupon(args) {
  const validator = new InputValidator(args, ERROR_TYPE)
      .checkArguments().mandatoryParameter('id')
  if (validator.error) {
      return validator.buildErrorResponse();
  }

  const cart = new MagentoCart(args, cartMapper.mapCart, 'guest-carts');
  return cart.byId(args.id).addCoupon(args.code).then(function () {
      return cart.byId(args.id).get();
  }).catch(error => {
      return cart.handleError(error);
  });
}
```

Example from Adobe CIF-Magento

```
function postCoupon(args) {
  const validator = new InputValidator(args, ERROR_TYPE)
      .checkArguments().mandatoryParameter('id')
  if (validator.error) {
      return validator.buildErrorResponse();
  }

  const cart = new MagentoCart(args, cartMapper.mapCart, 'guest-carts');
  return cart.byId(args.id).addCoupon(args.code).then(function () {
      return cart.byId(args.id).get();
  }).catch(error => {
      return cart.handleError(error);
  });
}
```

Serverless state machines

Serverless state machines

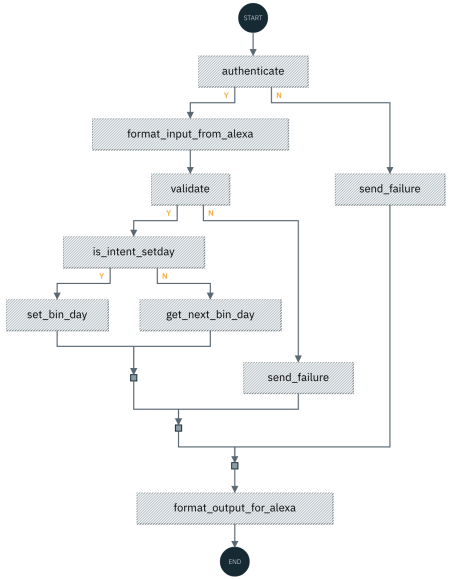


IBM Cloud Shell

HELP

binday.js

Composer



Visualization JSON Code

1:1

OpenWhisk Composition

```
composer.sequence(
    composer.if(
        'binday/authenticate',
        composer.if(
            'binday/validate',
            composer.if(
                'binday/is_intent_setday',
                'binday/set_bin_day',
                'binday/get_next_bin_day'
            ),
            'binday/send_failure'
        ),
        'binday/send_failure'
    ),
    'binday/format_output_for_alexa'
)
```

Tips & tricks

- Make troubleshooting easier:
  - Ability to disable event triggers
  - Identifiers that run throughout functions for a single "operation"
- Don't forget HTTP circuit breakers - retry with a back-off algorithm
- Tune memory settings for each function - affects price (& performance)

To sum up

Thank you!