Serverless Microservices Are The New Black

Lorna Mitchell, IBM

Serverless

FaaS: Functions as a Service

- write a function (many languages supported)
- deploy it to the cloud (Lambda, Cloud Functions, etc)
- only pay while the function is running (charged per GBsec)
- your platform scales on demand

When To Go Serverless

- To create a small, scalable application (focussed API, microservices)
- For occasional server needs (contact form on static site)
- To provide compute power (processing quantities of data)
- To move heavy lifting off web platform (classic example: PDF generation)

FaaS + HTTP = Microservices!

Microservices

Microservices are:

- small and modular
- loosely coupled
- independently developed and deployed
- great for building components
- decentralised

... they are basically small HTTP APIs

Microservice Design Points

I prefer RESTful-ish APIs

- Status codes are important
- Headers are for metadata
- URLs and verbs together define what happens
- All endpoints are stateless
- SSL is required

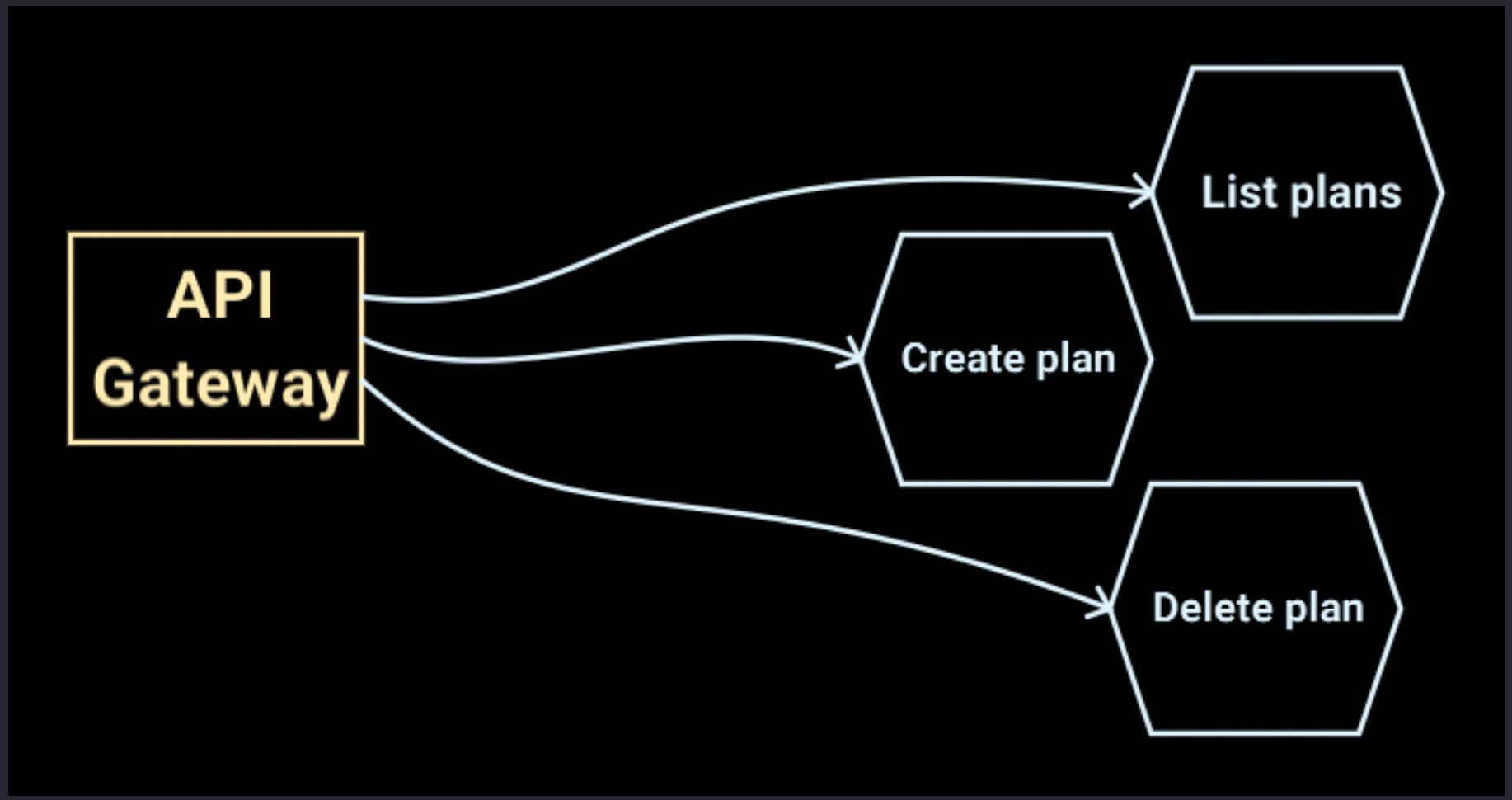
Lorna's Plans Service

Keep a list of my travel plans, with locations and dates.

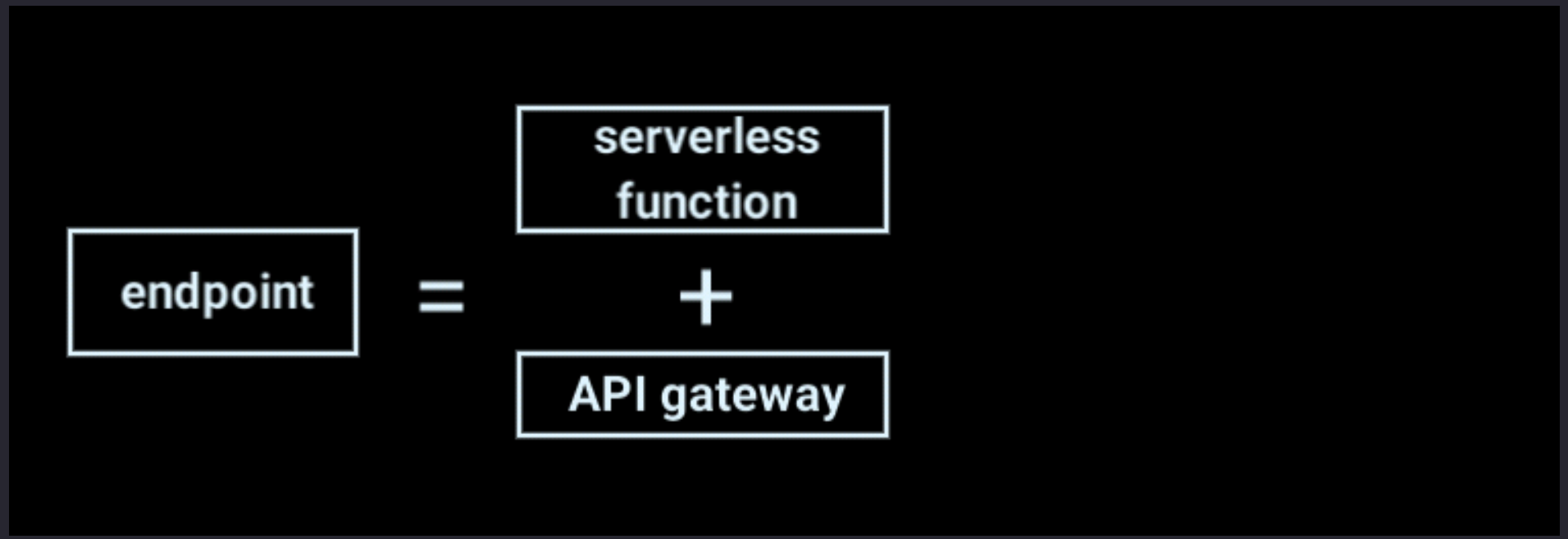
Use a serverless platform (IBM Cloud Functions) and PostgreSQL

`GET /plans`	list all plans
`GET /plans/42`	show one plan
`POST /plans`	create a new plan
`DELETE /plans/42`	delete a plan

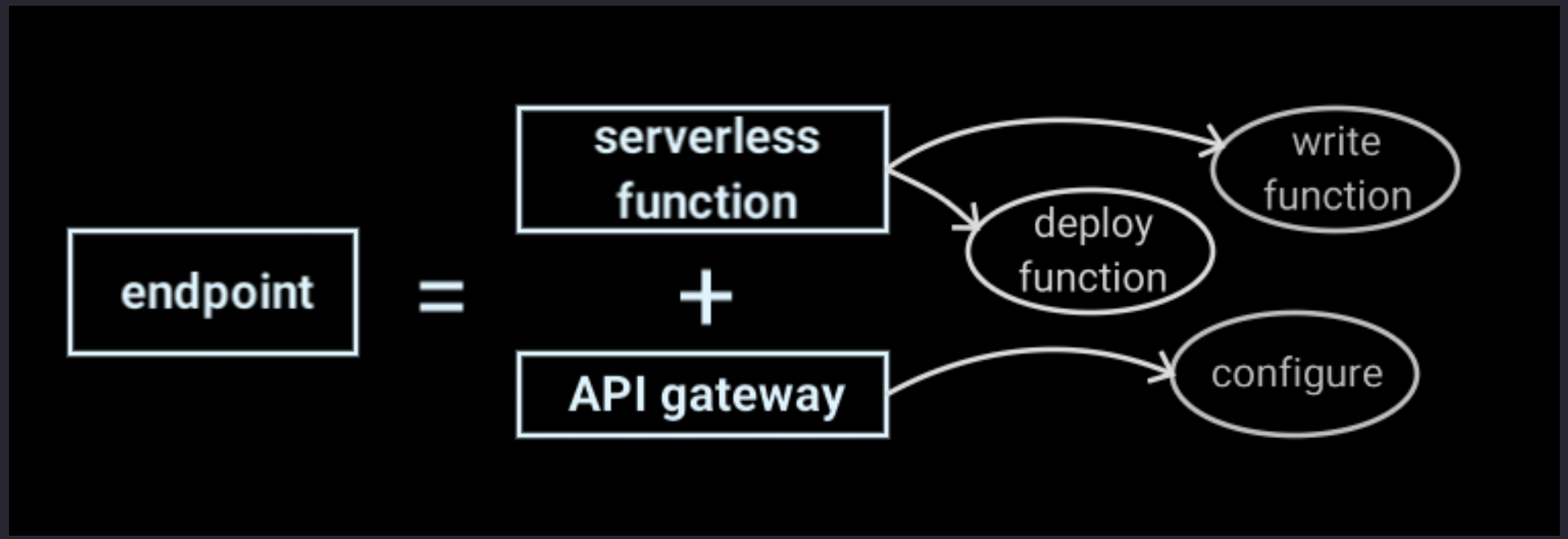
Lorna's Plans Service



Creating the Microservices



Creating the Microservices



Writing Serverless Functions

- Small, self-contained units of functionality
- Run on-demand in response to an event
- Incoming parameters can include
  - event information
  - parameters set at deploy time
  - connection info for other services

Make Plans: the Code

```
const pgp = require('pg-promise')();
function main(params) {
  var postgres_url = params['__bx_creds']['compose-for-postgresql']['uri'];
  var base_url = params['__ow_headers']['x-forwarded-url'];
  return new Promise(function(resolve, reject) {
    db = pgp(postgres_url, []);

    db.one("INSERT INTO plans (location, travel_date) VALUES
      ($1, $2) RETURNING plan_id",
      [location, travel_date])
      .then(function(data) {
        var redirect_to = base_url + "/" + data.plan_id;
        resolve({headers: {"Location": redirect_to},
          statusCode: 303})
      })
```

Prepare to Deploy: package

In OpenWhisk, there are "packages". These let us:

- group actions together
- add parameters to a package that will be available to all actions

From the deployment script, the line to create **`plans-api`**:

```
ibmcloud wsk package update plans-api
```

Prepare to Deploy: services

The function needs to connect to the database. We can *bind* the database to the package to achieve this:

```
ibmcloud wsk service bind compose-for-postgresql plans-api
```

Prepare to Deploy: libraries

To include extra libraries, we can:

- create **`package.json`** and run **`npm install`**
- zip up **`index.js`** and **`node_modules`** into a zip file
- deploy the zip file, and include runtime instructions

```
cd write-plan
zip -rq write-plan.zip index.js node_modules
```

Make Plans: Deploy

We're ready! Push the action to the cloud:

```
ibmcloud wsk action update --kind nodejs:8 --web raw \
plans-api/write-plan write-plan.zip
```

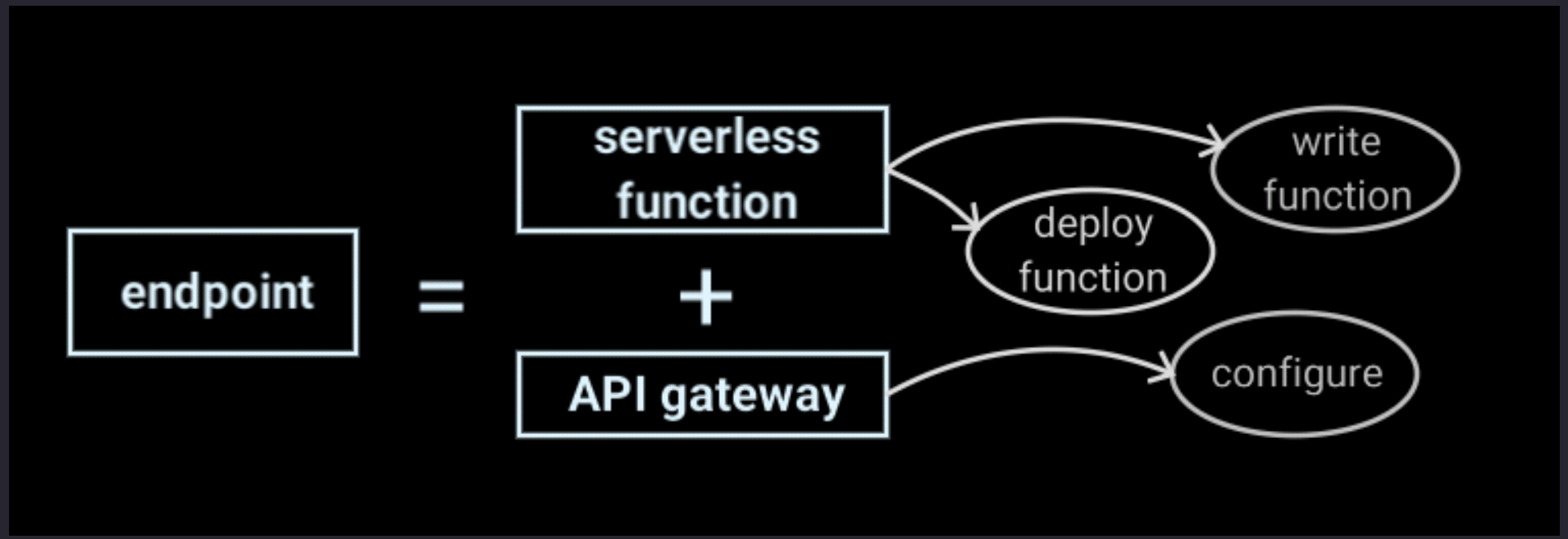
Make Plans: API Gateway

To have this action repsond to our HTTP request, set up the API Gateway:

- create the API
- set up the path, verb and action to link together

```
ibmcloud wsk api create /plans GET plans-api/get-plans \
--response-type http
```

Creating the Microservices



Calling The Endpoint

Quick example with cURL (other clients available):

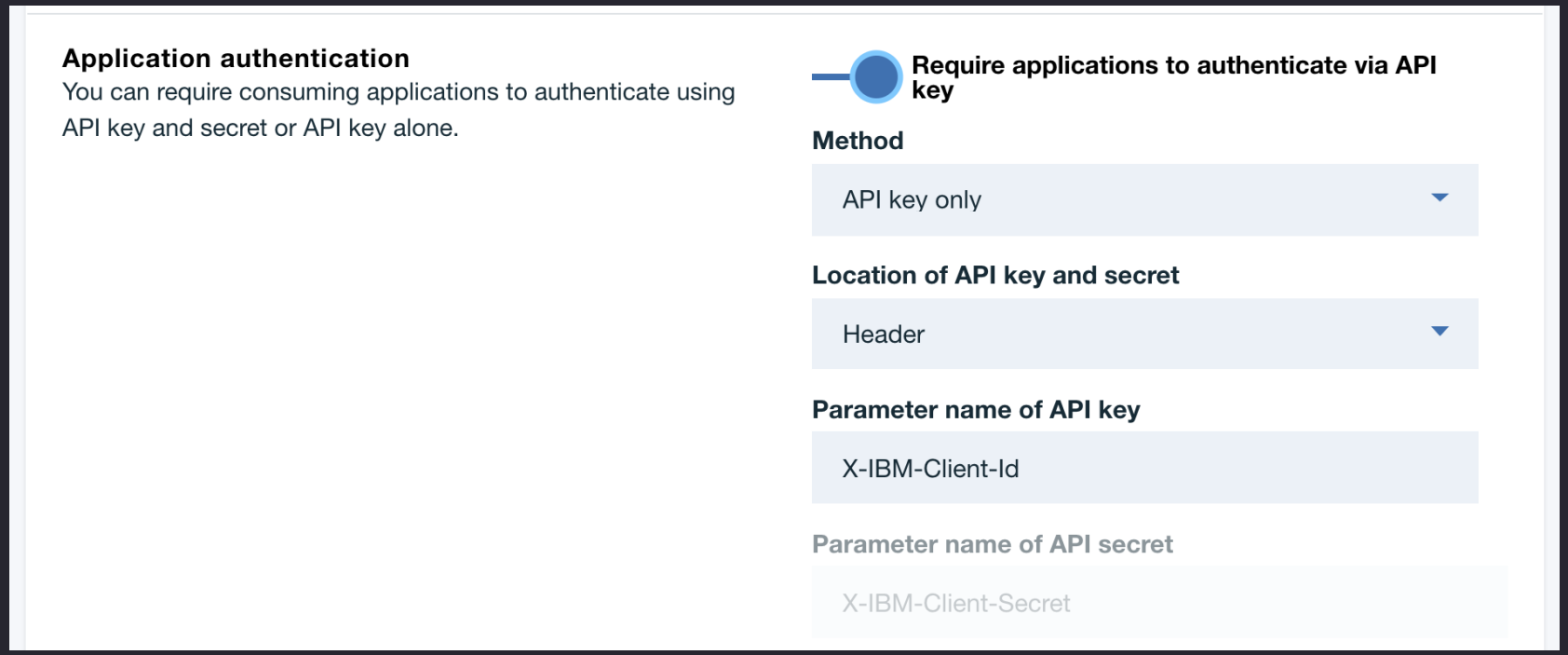
```
$ curl -L -H "Content-Type: application/json" \
      https://service.eu.apiconnect.ibmcloud.com/.../plans \
      -d '{"location": "Turin", "travel_date": "2018-04-11"}'

{
      "plans": [{
        "plan_id": 3,
        "travel_date": "2018-04-11T00:00:00.000Z",
        "location": "Turin"
      }]
}
```

Microservices: Security

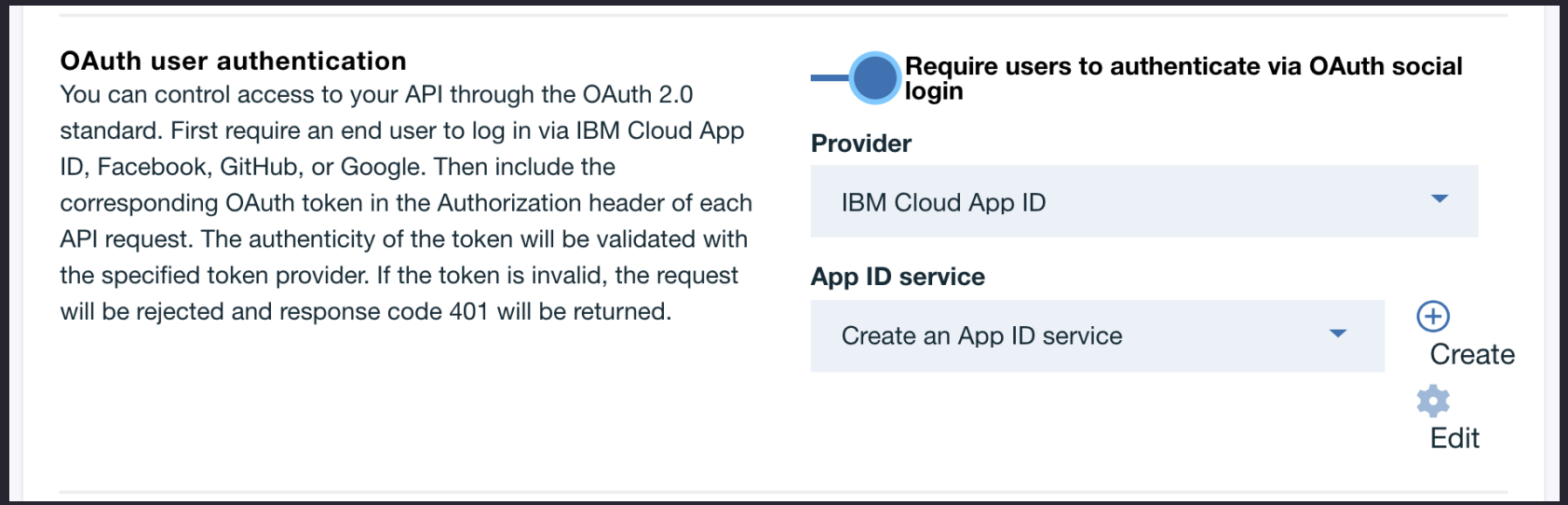
Security

In web console:
https://console.bluemix.net/openwhisk/apimanagement



Security

In web console:
https://console.bluemix.net/openwhisk/apimanagement



Project Structure

Many possible approaches, this is mine:

```
.
    deploy.sh
    get-plans
        index.js
        node_modules
        package-lock.json
        package.json
    write-plan
        index.js
        node_modules
        package-lock.json
        package.json
```

Deployment

Using https://travis-ci.com/

- deploy with a script
- script downloads **`ibmcloud`** tool and **`cloud-functions`** plugin
- set an API key as an environment variable
- then run commands (see **`deploy.sh`** in GitHub project)

Serverless Microservices

Serverless

Ideal for working with many small parts

Apache OpenWhisk paired with API Gateway: perfect candidate for microservices

Microservices

Service Oriented Architecture is alive and well

- microservices expose endpoints
- they share reusable components
- specific components guard access to services/datastores
- each component can be separately developed, tested and deployed

Resources

- Code: https://github.com/lornajane/plans-microservice
- Apache OpenWhisk: http://openwhisk.apache.org
- IBM Cloud Functions: https://www.ibm.com/cloud/functions
- My blog: https://lornajane.net