COMSE6998: Modern Serverless Cloud Applications

Lecture 3: REST API and Lambda Patterns, Programmable Web

Dr. Donald F. Ferguson
Donald.F.Ferguson@gmail.com

Contents

- Introduction
  - Emails and Piazza
  - TAs
  - Q&A
- Evolution and Motivation: Web apps → SOA/Microservices → Serverless
- ~~Top-Level Design Specification~~
- REST API Patterns; Introduction to Swagger
  - Introduction to query
  - Links/references
  - Error codes
- Address Verification API and Integration into Project.

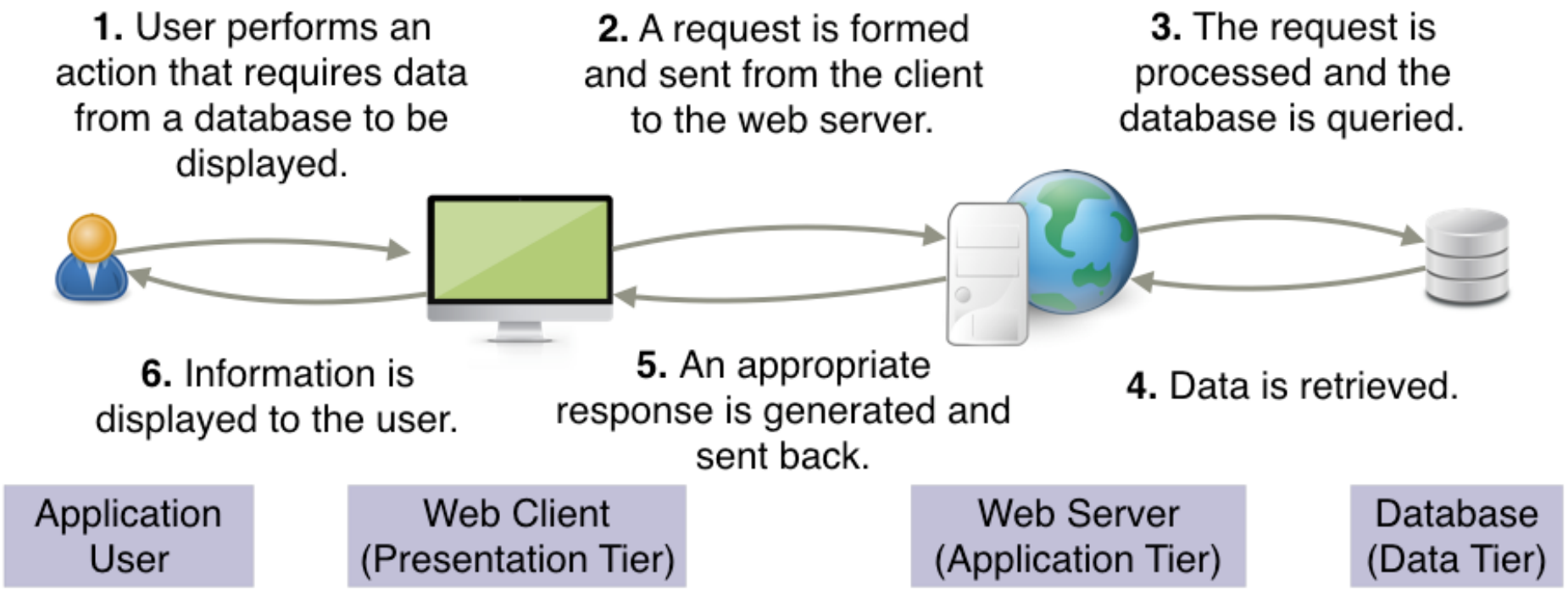
Introduction

Introduction

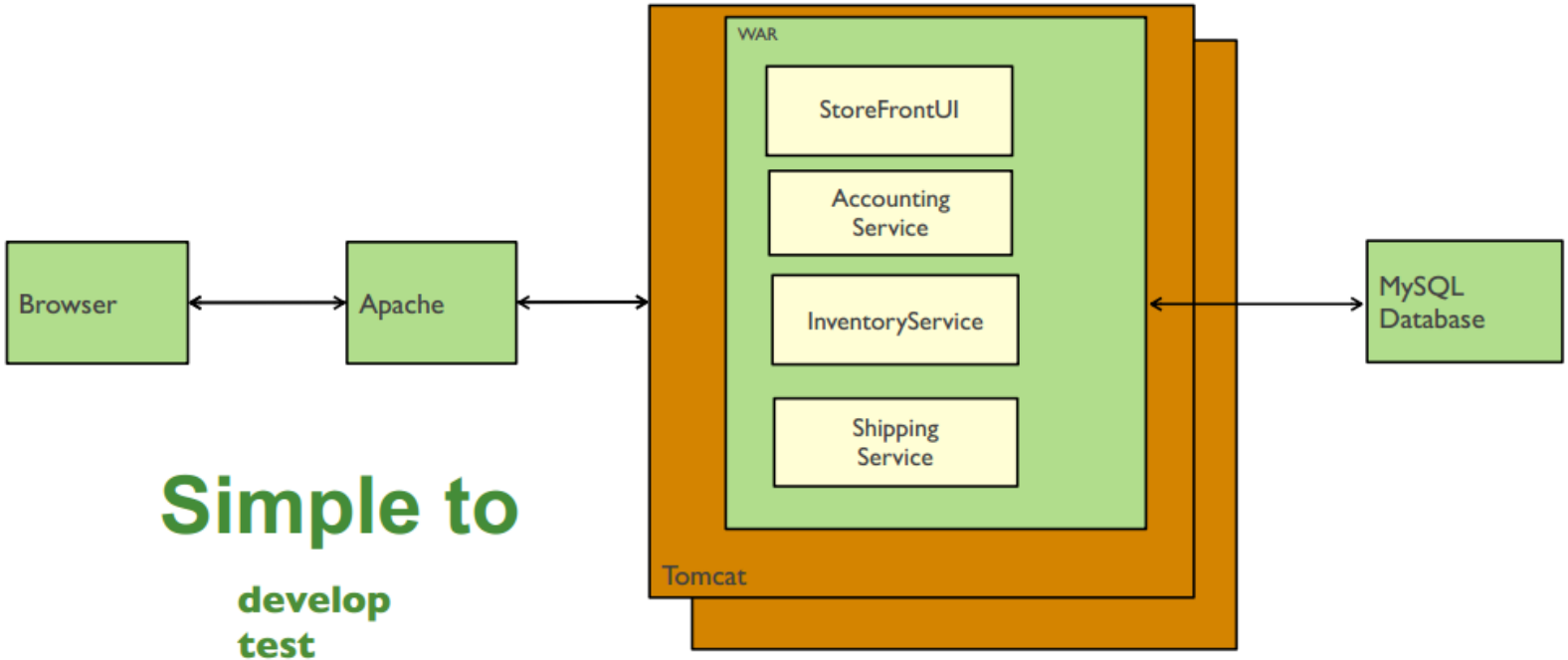
- Emails and Piazza
  - I have not been good at reading dff9@columbia.edu. “Out of practice.”
  - Piazza notifications → dff9@columbia.edu.
  - Until I get back into the habit, copy donald.f.ferguson@gmail.com if I do not respond quickly.
- Tas
  - Have one TA.
  - Will be finalizing one or two more.
- Q&A

Evolution and Motivation

Web Application Basic Concepts



Traditional web application architecture



Simple to

develop
test
deploy
scale

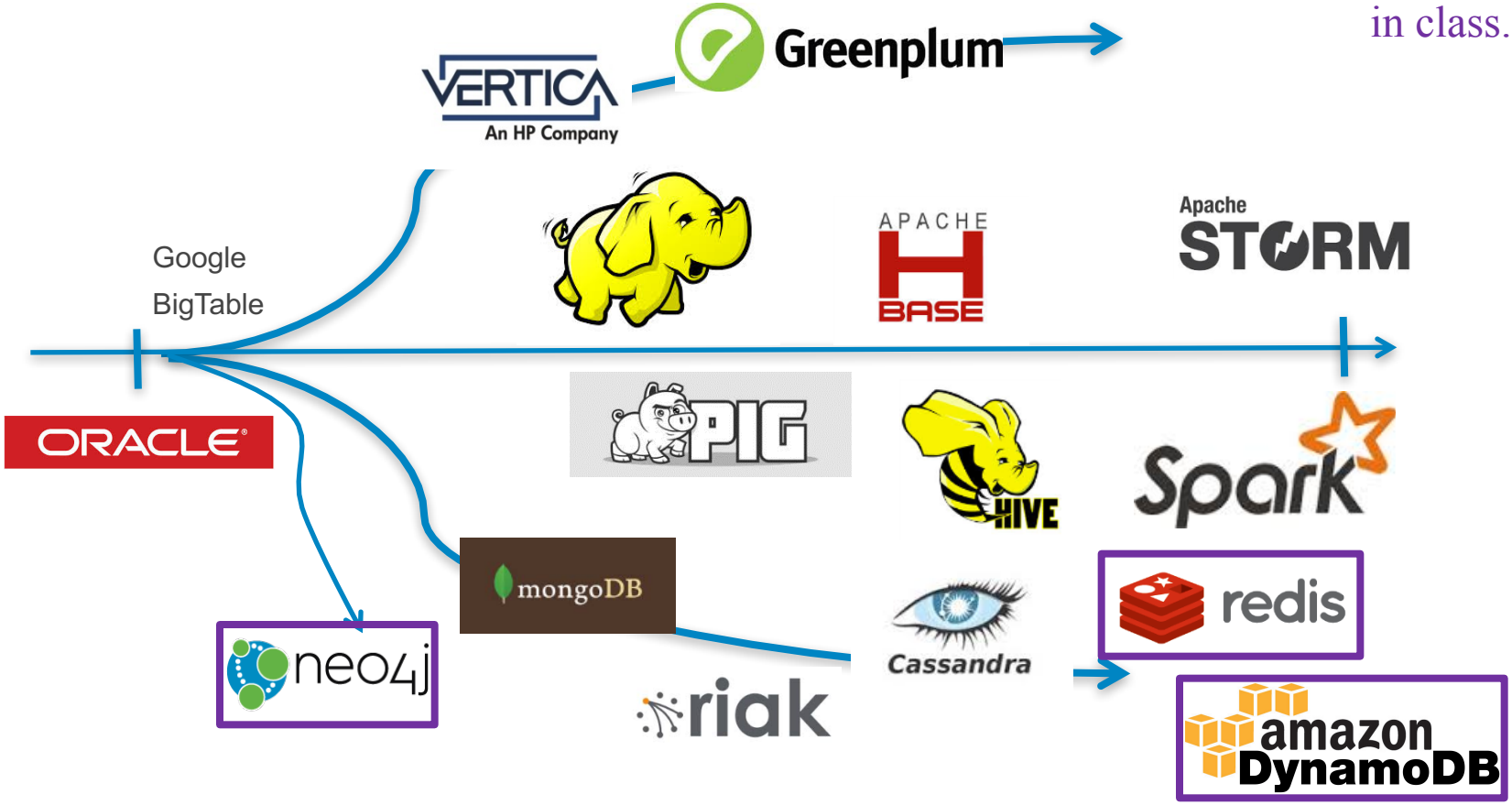
"Decomposing applications for deployability and scalability," Chris Richardson, http://plainoldobjects.com/

Evolution of Application Development

- (Web) application development went through a phase in which there were two dominant technologies:
  - J2EE: Java, JDBC, JMS, … …
  - .NET: C#, ADO.NET, SQL Server, … …
- Polyglot persistence emerged because
  - Use cases emerged
  - That were difficult to map to RDB semantics and optimizations.
  - Which drove the development of new, simple, problem focused DBs
- Polyglot programming emerged because
  - Solving some problems seems easier with specific, focused languages
  - Java and C# became powerful and complicated, and many scenarios needed much simpler and some different capabilities.
  - The browser document model is more dynamically typed than stricture languages →
    - Single language for {UI, business logic, data}.
    - More flexibly typed, dynamic languages.

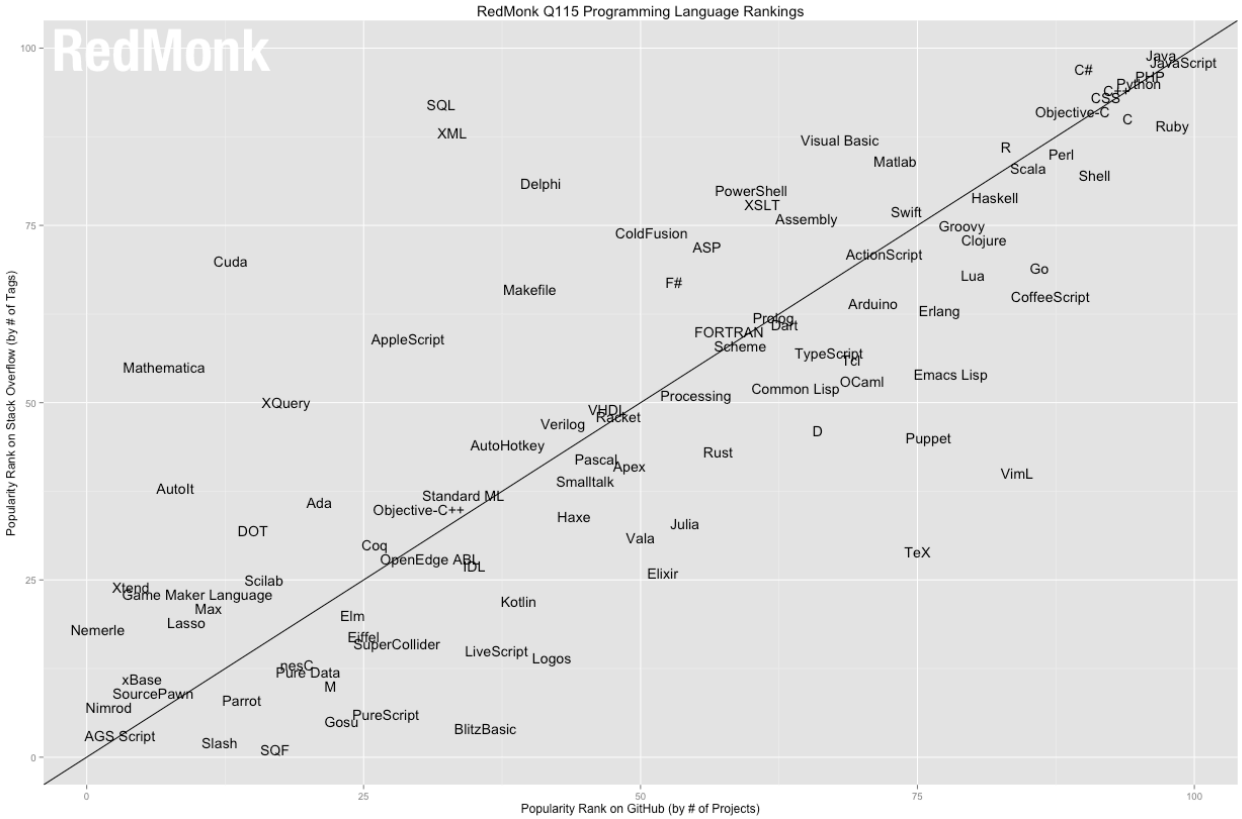
Polyglot Persistence (Very Incomplete List)

Will overview in class.

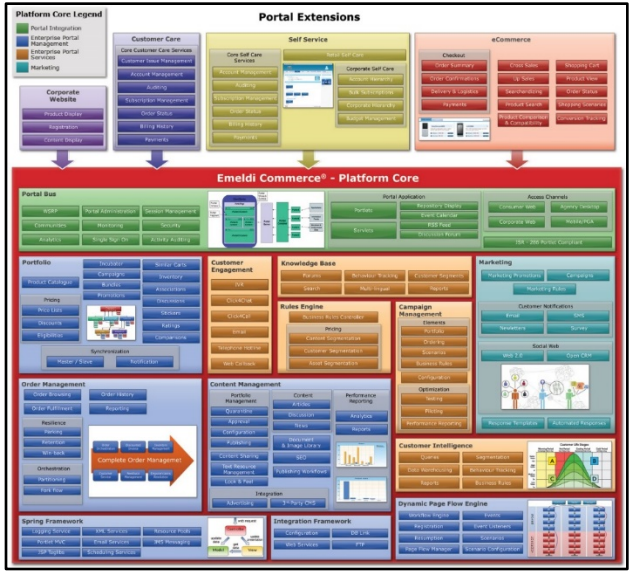


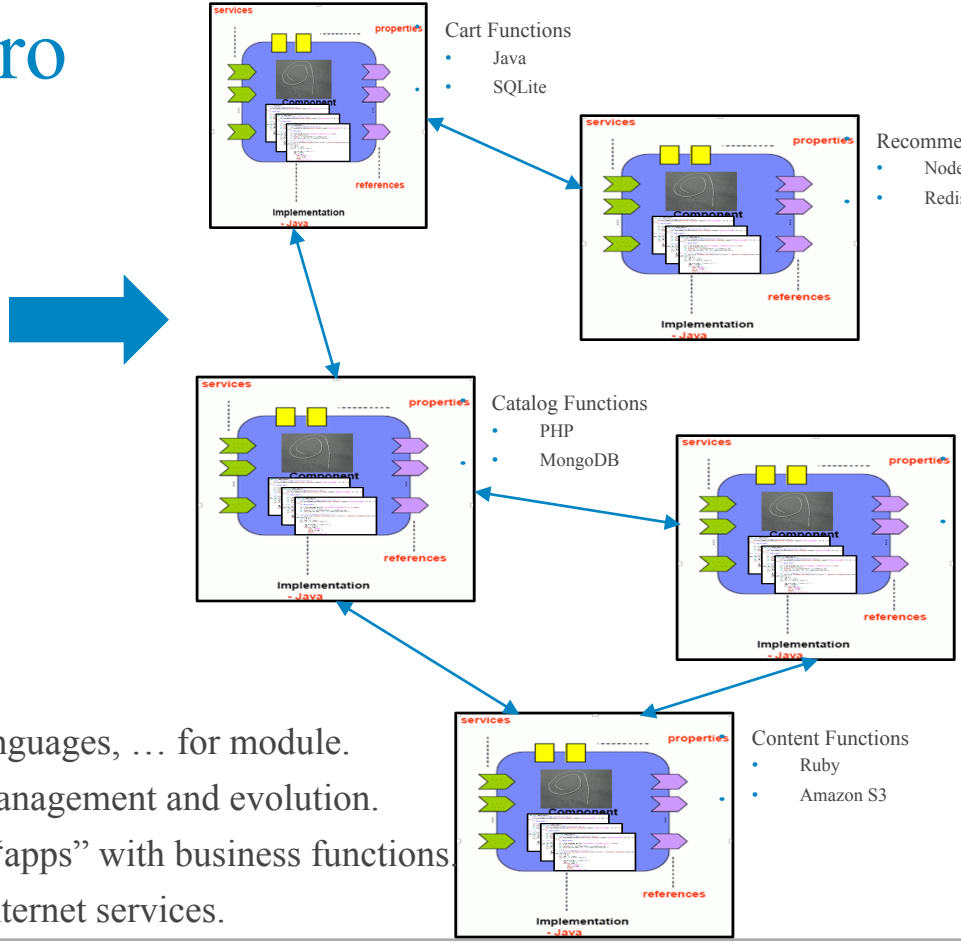
Polyglot Programming

Polyglot Programming



Monolithic to Micro





Motivations

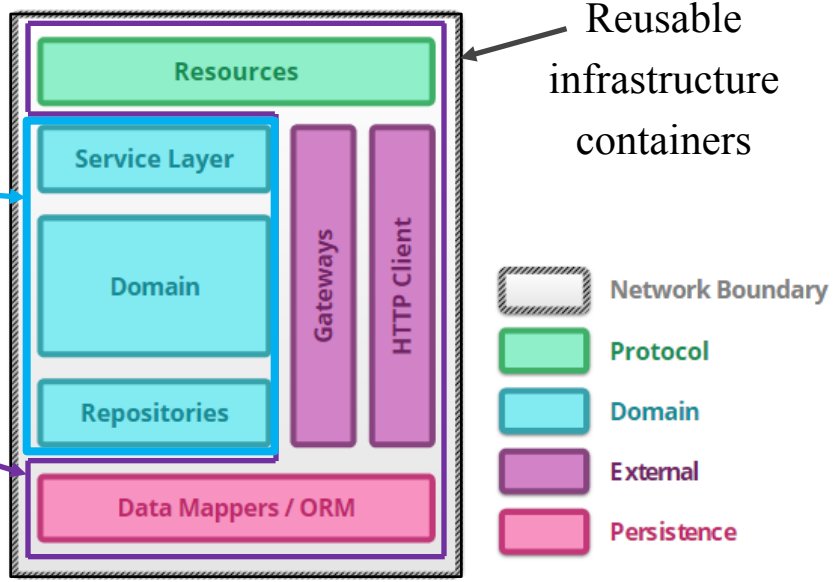
- Enable best tools, languages, … for module.
- Simplifies change management and evolution.
- Better alignment of “apps” with business functions.
- Reuse of code and internet services.

Microservices can usually be split into similar kinds of modules

Often, microservices display similar internal structure consisting of some or all of the displayed layers.

Inject application implementation into reusable SW containers

Reusable SW containers but with core technology and frameworks



Reusable infrastructure containers

Micro-services Characteristics

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

There are 5 principles of serverless architecture that describe how an ideal serverless system should be built. Use these principles to help guide your decisions when you create serverless architecture.

1. Use a compute service to execute code on demand (no servers)
2. Write single-purpose stateless functions
3. Design push-based, event-driven pipelines
4. Create thicker, more powerful front ends
5. Embrace third-party services

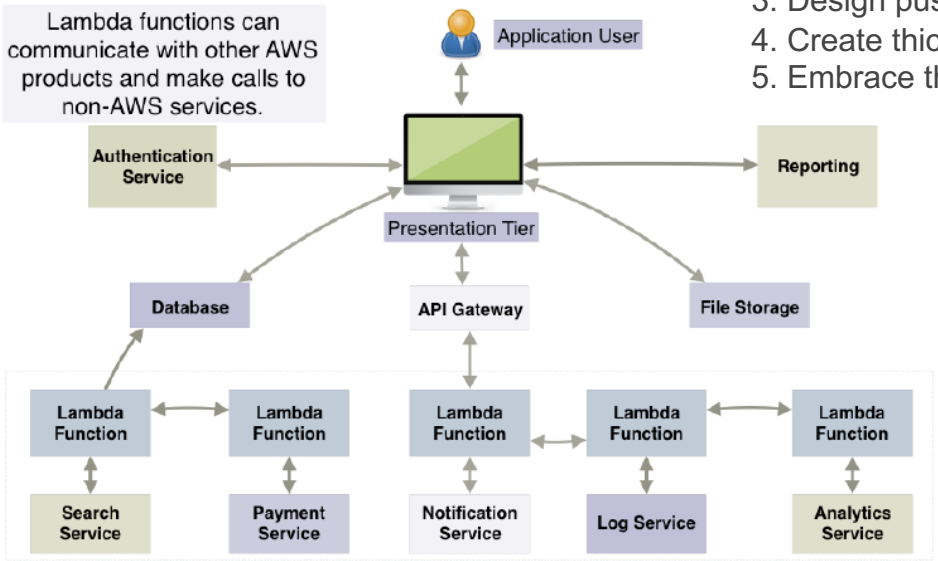
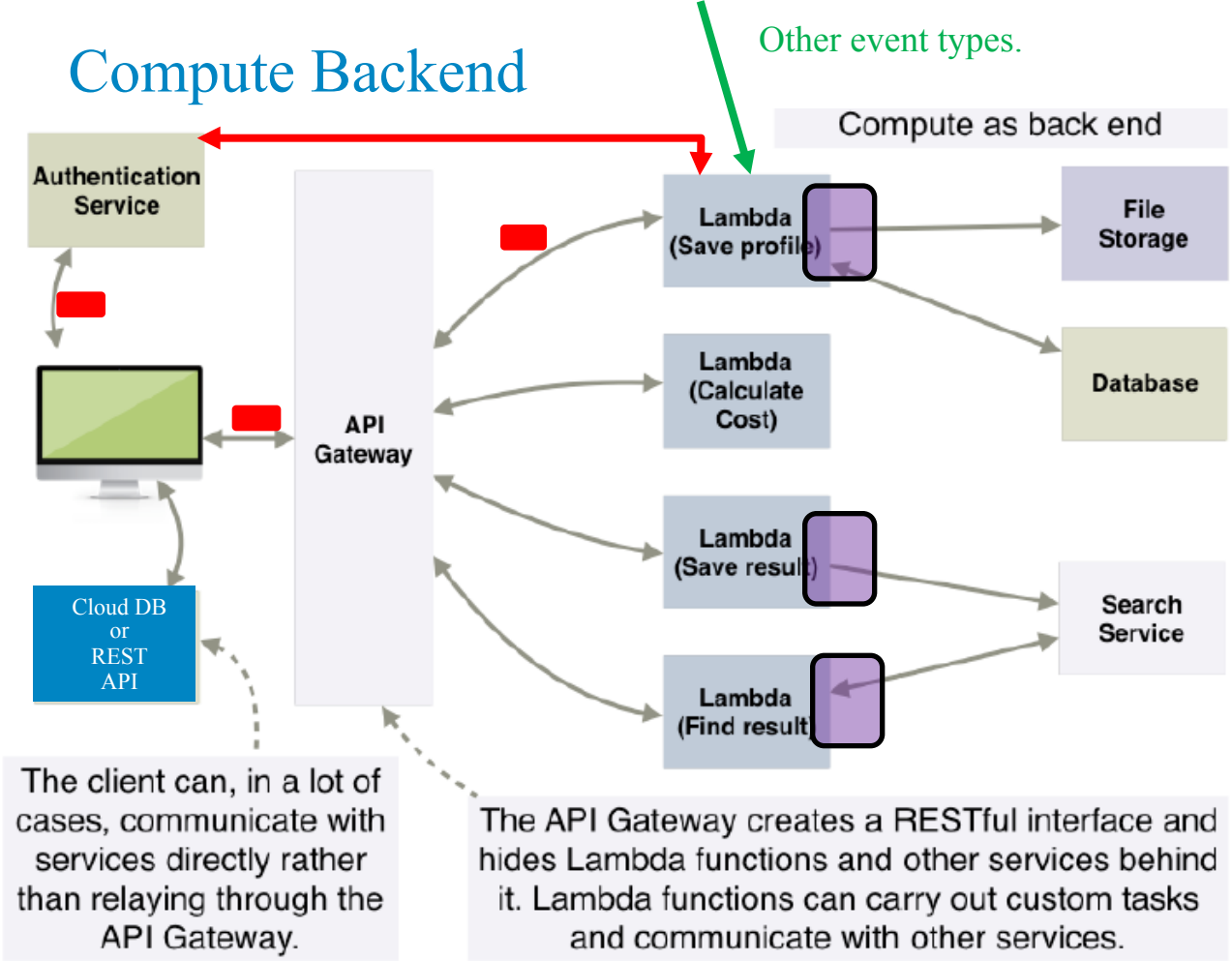


Figure 1.3: In a serverless architecture there is no single traditional back end. The front end of the application communicates directly with services, the database, or compute functions via an API gateway. Some services, however, must be hidden behind compute service functions where additional security measures and validation can take place.

Some observations on serverless

- There is running code → "some server somewhere."
- In IaaS,
  - You get the virtual sever from the cloud.
  - But know it is there, and manage and config it.
- In PaaS/microservices
  - You are aware of/build the "application sever."
  - And supporting frameworks.
  - And bundle/tarball it all together.
- In serverless,
  - You write a function based on a template.
  - Upload to an internet "event" endpoint.
  - Anything you call is a "cloud service."

Compute Backend



Observations

- Front end and backend both communicate with authentication service.
- Context flows on calls.
- Well-designed code, even a single function, uses a service abstraction for accessing data.
- Multiple event types drive "business function" → API cannot be coupled to specific event format.

Serverless from Microservices

- 1. Use a compute service to execute code on demand (no servers)
  - No need to define and maintain a runtime engine and app server.
  - Eliminates managing, monitoring, … app server runtime instances.
- 2. Write single-purpose stateless functions
  - More flexible and dynamic lifecycle → agility
  - Evolves to an HTML/wiki like model from a stop-deploy module-restart, especial where a lot of the module has not changed.
- 3. Design push-based, event-driven pipelines
  - Microservices implies invocation only by HTTP/REST.
  - Multiple event types trigger serverless: {event, condition, action, event} model.
- 4. Create thicker, more powerful front ends
  - No "module" → code that assembles multiple data sources and functions
  - Moves from microservice to front-end.
- 5. Embrace third-party services
  - No local libraries and server runtimes →
  - All calls are inherently "web" calls.

I do not totally agree with this observation.

~~Top-Level Design Specification~~

REST API Patterns

Implementing Simple Query

GET .../Resource?p1=v1&p2=v1&...

- Validate the query expression
  - Are p1, p2 and … *queryable* fields?
  - Is the combination correct, e.g. can I use p1 or p2 but not both?
  - Do v1, v2, ... seem reasonable, e.g. “zipcode=yellow” is not reasonable?
- Rewrite the query into the target DB engine’s query language?

```
var params = {
    TableName : "contosocustomers",
    IndexName : index_name,
    KeyConditionExpression: condition,
    ExpressionAttributeValues: {
        ":name": name
    }
};
```

SELECT * FROM customers
WHERE
lastname = ‘Ferguson’;

- Execute query and handle errors.
- Covert results to expected format, e.g. do not surface a DynamoDB format or SQL format.

Query

- Based on input, choose
  - Index name
  - Condition (“where clause’)
  - Values to insert into expression.
- Validate Query Parameters
  - Lastname only is valid.
  - Firstname only is valid.
  - Both are invalid.
  - Neither is invalid.
- Execute
  - Build parameter data.
  - Call “Query.”

```
function findCustomerByQuery(event, callback1, callback2) {

    var index_name      =   null;
    var condition       =   null;
    var name            =   null;

    if (event.lastname) {
        index_name = "lastname-index";
        condition = "lastname = :name";
        name = event.lastname;
    }
    else {
        if (event.firstname) {
            index_name = "firstname-index";
            condition = "firstname = :name";
            name = event.firstname;
        }
    }

    if ((event.lastname) && (event.firstname)) {
        callback2(new Error(`Crappy input "${event}"`));
    }
    if (index_name === null) {
         callback2(new Error(`Crappy input "${event}"`));
    }

    var params = {
        TableName : "contosocustomers",
        IndexName : index_name,
        KeyConditionExpression: condition,
        ExpressionAttributeValues: {
            ":name": name
        }
    };

    console.log("In getCustomerByQuery, params = " + JSON.stringify(params));

    dynamo.query(params, function(err, data) {
        if (err) {
            console.log ("Error = " + JSON.stringify(err));
            callback1(err, null, callback2);
        } else {
            console.log("Get customer success, data = " + JSON.stringify(data));
            callback1(null, data, callback2);
        }
    });
}
```

Mapping

GET .../Customers?lastname=Ferguson

Web sends.

```
#set($inputRoot = $input.path('$'))
{
    "operation" : "query",
    "lastname" : "$input.params('lastname')",
    "firstname" : "$input.params('firstname')"
}
```

```
Wed Sep 21 15:52:53 UTC 2016 : Endpoint request body after transformations: {
    "operation" : "query",
    "lastname" : "Ferguson",
    "firstname" : ""
}
```

Function expects.

```
#set($inputRoot = $input.path('$'))
$input.json('$.Items')
```

```
{
  "Items": [
    {
      "lastname": "Ferguson",
      "firstname": "Donald",
      "email": "dff9@columbia.edu",
      "address": "98bb32d0-32b2-44f0-8a4f-31176ac17340"
    },
    {
      "Firstname": "Donald",
      "lastname": "Ferguson",
      "email": "don@foo.edu"
    }
  ],
  "Count": 2,
  "ScannedCount": 2
}
```

Function returns.

Web expects.

```
  {
    "lastname": "Ferguson",
    "firstname": "Donald",
    "email": "dff9@columbia.edu",
    "address": "98bb32d0-32b2-44f0-8a4f-31176ac17340"
  },
  {
    "Firstname": "Donald",
    "lastname": "Ferguson",
    "email": "don@foo.edu"
  }
]
```

Representational State Transfer (REST)

- People confuse
  - Various forms of RPC/messaging over HTTP
  - With REST
- REST has six core tenets
  - Client/server
  - Stateless
  - Caching
  - Uniform Interface
  - Layered System
  - Code on Demand

HATEOAS: Hypertext as the Engine of Application State -- The principle is that a client interacts with a network application entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia.

Handling Links

- In a Customer
  - The address ID
  - Is actually a *link*
  - Into another set of *resources*

```
{
  "firstname": "Donald",
  "lastname": "Ferguson",
  "email": "dff9@columbia.edu",
  "address": "98bb32d0-32b2-44f0-8a4f-31176ac17340"
}
```

- Resolving the link would require
  - Non-hypertext/web
  - Side knowledge
  - And violate HATEOAS
  - And forces some out-of-band documentation/info.

```
{
  "firstname": "Donald",
  "lastname": "Ferguson",
  "email": "dff9@columbia.edu",
  "address": {
    "href": "../Addresses/98bb32d0-32b2-44f0-8a4f-31176ac17340"
  },
  "self": {
    "href": "../Customers/dff9@columbia.edu"
  }
}
```

- Instead, return links as links.

Several Patterns, for Example

Simple

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
  "href" : "https://api.stormpath.com/v1/accounts/cJoiwcorTTmkDDBsf02AbA",
  "username" : "jlpicard",
  "email" : "capt@enterprise.com",
  "givenName" : "Jean-Luc",
  "middleName" : "",
  "surname" : "Picard",
  "status" : "enabled",
  "directory" : {
    "href" : "https://api.stormpath.com/v1/directories/WpM9nyZ2TbaEzfbRvLk9K
  },
  …
}
```

Link with Resource Expansion

```
{
    "href": "https://api.stormpath.com/v1/accounts/ZugcG3JHQFOTKGEXAMPLE",
    "username": "lonestarr",
    "email": "lonestarr@druidia.com",
    "fullName": "Lonestarr Schwartz",
    "givenName": "Lonestarr",
    "middleName": "",
    "surname": "Schwartz",
    "status": "ENABLED",
    "emailVerificationToken": null,
    "directory": {
        "href": "https://api.stormpath.com/v1/directories/S2HZc7gXTumVYEXAMPLE",
        "name": "Spaceballs",
        "description": "",
        "status": "ENABLED",
        "accounts": {
            "href":"https://api.stormpath.com/v1/directories/S2HZc7gXTumVYEXAMPLE/accounts
        },
        "groups": {
            "href":"https://api.stormpath.com/v1/directories/S2HZc7gXTumVYEXAMPLE/groups"
        },
        "tenant":{
            "href":"https://api.stormpath.com/v1/tenants/wGbGaSNuTUix9EXAMPLE"
        }
    },
    "tenant": {
        "href": "https://api.stormpath.com/v1/tenants/wGbGaSNuTUix9EXAMPLE"
    },
}
```

Data fields followed by links section.

```
{
   …
   "links": [
       { "rel" : "directory", "link" : "https://api.stormpath.com/v1/directories/WpM9nyZ2TbaEzfbRvLk9
       { "rel" : "tenant", "link" : "https://api.stormpath.com/v1/tenants/wGbGaSNuTUix9"}
    ],
    …
}
```

My Crappy Code

- Some problems
  - Only works for single item return, e.g. does not work for Query.
  - Relative hrefs are fragile *and* expose that the same code is handling two resources.
  - Hardcoded strings.
  - Lambda function is now "protocol specific."
- For project
  - Do something simple.
  - Should work for query and individual resources.

```
function theCallback(err, data, callback) {
    console.log("getCustomer:Before callback");

    if (data) {
        //callback(null, JSON.stringify(data));
        if (data.Item.address) {
            data.Item.address = {
                "href" : "../Addresses/" + data.Item.address
            }
            data.Item.self = {
                "href" : "../Customers/" + data.Item.email
            }
        }
        callback(null, data);
        console.log("theCallback:  data = " + JSON.stringify(data));
    }
    if (err) {
        callback(err, null);
        console.log("theCallback: failure = " + JSON.stringify(err));
    }
}
```

Success Response Codes

202 means

- Your request went asynch.
- The HTTP header Link is where to poll for rsp.
- We will cover later.

Operation	HTTP Request	HTTP Response Codes Supported
READ	GET	200 - OK with message body 204 - OK no message body 206 - OK with partial message body
CREATE	POST	201 - Resource created (Operation Complete) 202 - Resource accepted (Operation Pending)
UPDATE	PUT	202 - Accepted (Operation Pending) 204 - Success (Operation Complete)
DELETE	DELETE	202 - Accepted (Operation Pending) 204 - Success (Operation Complete)

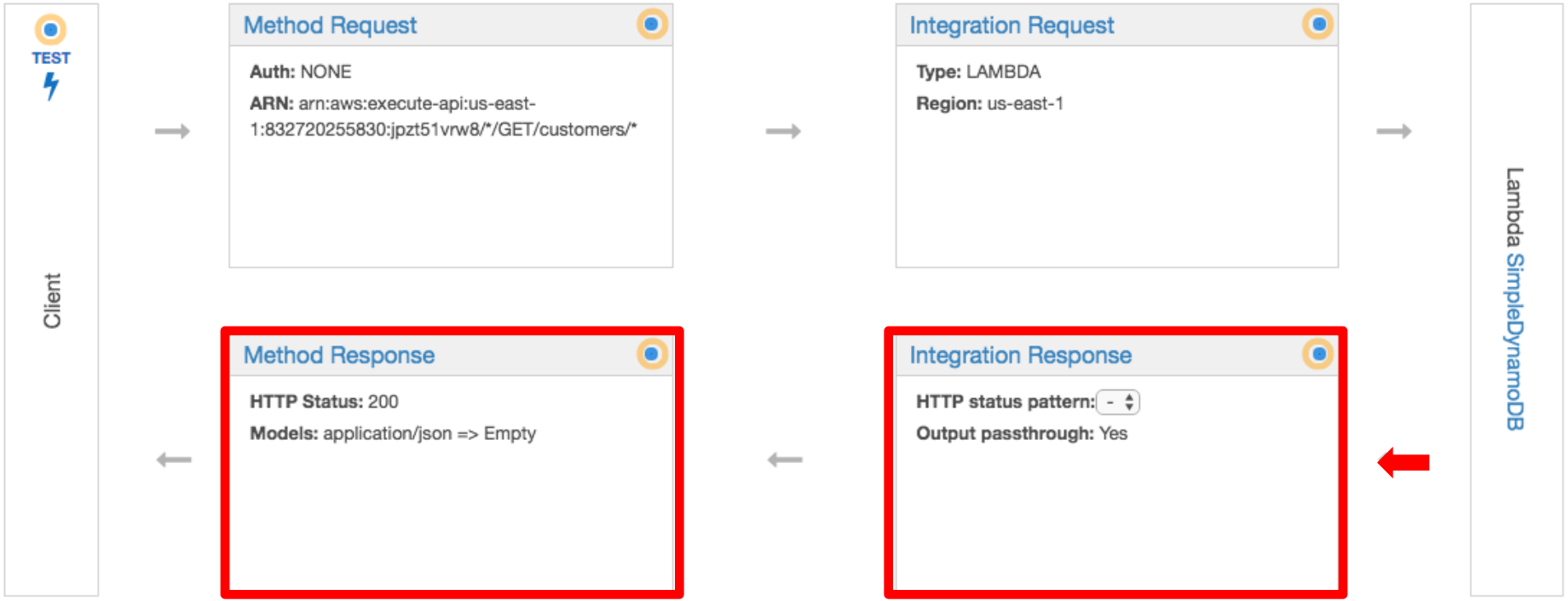
Examples of Link Headers in HTTP response:

```
Link: <http://api/jobs/j1>;rel=monitor;title="update profile"
Link: <http://api/reports/r1>;rel=summary;title="access report"
```

(Some) Failure Response Code

Error	Response Code
Invalid Parameter	400 - Invalid parameter
Authentication	401 - Authentication failure
Permission Denied	403 - Permission denied
Not Found	404 - Resource not found
Invalid Request Method	405 - Invalid request method
Internal Server Error	500 - Internal Server Error
Service Unavailable	503 - Service Unavailable

Responses



Integration Response

Resources | Actions

- /
  - OPTIONS
  - /dev
    - OPTIONS
    - /authentications
      - OPTIONS
      - POST
    - /commandprocessor
      - OPTIONS
      - POST
    - /comments
      - GET
      - OPTIONS
      - POST
      - /{commentid}
        - DELETE
        - GET
        - OPTIONS
        - PUT
    - /complexqueries
      - OPTIONS
      - POST
    - /contentinstances
      - GET
      - OPTIONS
      - POST
      - /{contentinstanceid}
        - DELETE
        - GET
        - OPTIONS
        - PUT
    - /customers
      - GET

← Method Execution /dev/comments - POST - Integration Response

First, declare response types using Method Response. Then, map the possible responses from the backend to this method's response types.

Lambda Error Regex	Method response status	Output model	Default mapping
-	200		Yes
401.*	401		No
403.*	403		No
404.*	404		No
422.*	422		No

Map the output from your Lambda function to the headers and output model of the 422 method response.

Lambda Error Regex 422.*

Method response status 422

Cancel Save

Header Mappings

Body Mapping Templates

Content-Type
application/json

Add mapping template

application/json

Generate template:

```
#set($inputRoot = $input.path('$'))
{
 "Error Message" : $input.json('$.errorMessage'),
 "Details" : $input.json('$.cause.errorMessage')
}
```

API Gateway Error Responses

Resources | Actions

- /
  - OPTIONS
  - /dev
    - OPTIONS
    - /authentications
      - OPTIONS
      - POST
    - /commandprocessor
      - OPTIONS
      - POST
    - /comments
      - GET
      - OPTIONS
      - POST
      - /{commentid}
        - DELETE
        - GET
        - OPTIONS
        - PUT
    - /complexqueries
      - OPTIONS
      - POST
    - /contentinstances
      - GET
      - OPTIONS
      - POST

← Method Execution /dev/comments - POST - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status
200
401
403
404
422
500

Response Headers for 500

Name
Access-Control-Allow-Origin

Add Header

Response Models for 500

Create a model

Content type	Models
application/json	Error

Add Response Model

Add Response

Some Java Code

- Internally,
  - Our code catches exceptions.
  - Detects errors.
  - Returns a “status” object to the top-level, “routing” function.
- If the status is not “success,”
  - Throw an exception back to Lambda.
  - “xxx This error occurred.”
  - Followed by details.
- Gateway applies regex to map application error into HTTP error.

```
protected RESTMessage postProcessRsp(RESTMessage response) {
    logger.info("Starts");
    RESTMessage result = null;
    RuntimeException e = null;
    result = response;
    Status status = response.getHeader().getMessageStatus();
    if (!status.successful()) {
        Integer code = status.getCode();
        String message = status.getMessage();
        String detailedMessage = status.getDetailedMessage();
        Throwable cause = new Throwable(detailedMessage);
        e = new RuntimeException(code + " " + message, cause);
        throw (e);
    }
    logger.info("Ends");
    return result;
}
```

Similar model applies for JavaScript, but you return the *err* on callback.

What's All This Model Stuff?

Models **Create**

- Customer
- Empty
- Error

Update Model

Make changes to your model in the form below. Models are declared using JSON schema.

Model name Customer

Content type application/json

Model description Customer

Model schema*

```
1  {
2    "$schema": "http://json-schema.org/draft-04/schema#",
3    "type": "object",
4    "properties": {
5      "firstname": {
6        "type": "string"
7      },
8      "lastname": {
9        "type": "string"
10     },
11     "email": {
12       "type": "string"
13     },
14     "address": {
15       "type": "object",
16       "properties": {
17         "href": {
18           "type": "string"
```

- How you "define" body formats.
- For application/xxx.
- Which is not defined in HTTP.
- Use JSON schema for JSON

JSON

URL

http://sparq.tv

JSON

```
{
  "firstname": "Donald",
  "lastname": "Ferguson",
  "email": "dff9@columbia.edu",
  "address": {
    "href": "../Addresses/98bb32d0-32b2-44f0-8a4f-31176ac17340"
  },
  "self": {
    "href": "../Customers/dff9@columbia.edu"
  }
}
```

Well done! You provided valid JSON.

Generate Schema | Reset

IDs

- ○ Absolute IDs
- ○ Relative IDs
- ◉ No IDs

Metadata

- ☐ Include metadata keywords

  Title and Description.

General

- ☐ Include default values

  Values are taken from JSON.

- ☐ Restrict values to enum

  Uses the default value and null.

- ☑ Force required

Objects

- ☑ Allow additional properties

  Controls whether it's valid to have additional properties in the object beyond what is defined in the schema.

Schema

Edit view has been disabled while bugs are fixed. Code and
Thank you for your patience.

Code View | Edit View | String View

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "firstname": {
      "type": "string"
    },
    "lastname": {
      "type": "string"
    },
    "email": {
      "type": "string"
    },
    "address": {
      "type": "object",
      "properties": {
        "href": {
          "type": "string"
        }
      },
      "required": [
        "href"
      ]
    },
    "self": {
      "type": "object",
      "properties": {
        "href": {
          "type": "string"
        }
      },
      "required": [
        "href"
      ]
    }
  },
  "required": [
    "firstname",
    "lastname",
    "email",
    "address",
    "self"
  ]
}
```

Simple Example

JSON

```
{
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York"
  },
  "phoneNumber": [
    {
      "location": "home",
      "code": 44
    }
  ]
}
```

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "address": {
      "type": "object",
      "properties": {
        "streetAddress": {
          "type": "string"
        },
        "city": {
          "type": "string"
        }
      },
      "required": [
        "streetAddress",
        "city"
      ]
    },
    "phoneNumber": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "location": {
            "type": "string"
          },
          "code": {
            "type": "integer"
          }
        },
        "required": [
          "location",
          "code"
        ]
      }
    }
  },
  "required": [
    "address",
    "phoneNumber"
  ]
}
```

Swagger

Stages **Create**

- prod

prod Stage Editor

Delete Stage

Invoke URL: https://jpzt51vrw8.execute-api.us-east-1.amazonaws.com/prod

Settings | **Stage Variables** | **SDK Generation** | **Export** | **Deployment History**

Export as Swagger

Export as Swagger + API Gateway Extensions

Export as Swagger + Postman Extensions


```
1  {
2    "swagger": "2.0",
3    "info": {
4      "version": "2016-09-21T19:07:41Z",
5      "title": "LambdaMicroservice"
6    },
7    "host": "jpzt51vrw8.execute-api.us-east-1.amazonaws.com",
8    "basePath": "/prod",
9    "schemes": [
10     "https"
11   ],
12   "paths": {
13     "/customers": {
14       "get": {
15         "consumes": [
16           "application/json"
17         ],
18         "produces": [
```

Swagger Editor: http://editor.swagger.io/#/

File ▾ Preferences ▾ Generate Server ▾ Generate Client ▾ Help ▾ ✔ Processed with no error

```yaml
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
  '/customers/{email}':
definitions:
  Empty:
    type: object
    title: Empty Schema
  Customer:
  CustomerPUT:
  CustomerQueryRSP:
    type: array
    items:
  CustomerPOST:
```

LambdaMicroservice

Version 2016-09-21T19:07:41Z

Paths

/customers

/customers/{email}

Models

Empty

```
▼Empty Schema {
}
```

Customer

```
▶Customer { }
```

CustomerPUT

```
▶CustomerPUT { }
```

CustomerQueryRSP

```
▶CustomerQueryRSP[]
```

CustomerPOST

```
▶CustomerPOST { }
```

Models

File ▾ Preferences ▾ Generate Server ▾ Generate Client ▾ Help ▾

```
1   swagger: '2.0'
2   info:
3     version: '2016-09-21T19:07:41Z'
4     title: LambdaMicroservice
5   host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
6   basePath: /prod
7   schemes:
8     - https
9   paths:
10    /customers:
126   '/customers/{email}':
229 definitions:
230   Empty:
231     type: object
232
233   Customer:
234     type: object
235     required:
236       - address
237       - email
238       - firstname
239       - lastname
240       - self
241     properties:
242       firstname:
243         type: string
244       lastname:
245         type: string
246       email:
247         type: string
248       address:
249         type: object
250         properties:
251           href:
252             type: string
253         required:
254           - href
255       self:
256         type: object
257         properties:
258           href:
259             type: string
260         required:
261           - href
262
278   CustomerQueryRSP:
279     type: array
280     items:
309   CustomerPOST:
330
```

LambdaMicroservice

Version 2016-09-21T19:07:41Z

Paths

/customers

/customers/{email}

Models

Empty

```
▼Empty Schema {
}
```

Customer

```
▼Customer {
  firstname: string *
  lastname:  string *
  email:     string *
  address:   ▼ {
               href: string *
             }
  self:      ▼ {
               href: string *
             }
}
```

CustomerPUT

```
▶CustomerPUT { }
```

CustomerQueryRSP

```
▶CustomerQueryRSP[]
```

CustomerPOST

```
▶CustomerPOST { }
```

Operations

```yaml
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
    get:
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: lastname
          in: query
          required: false
          type: string
        - name: firstname
          in: query
          required: false
          type: string
      responses:
        '200':
          description: 200 response
          schema:
            $ref: '#/definitions/CustomerQueryRSP'
          headers:
            Access-Control-Allow-Origin:
              type: string
      x-amazon-apigateway-integration:
        requestTemplates:
          application/json: |-
            #set($inputRoot = $input.path('$'))
            {
                "operation" : "query",
                "lastname" : "$input.params('lastname')",
                "firstname" : "$input.params('firstname')"
            }
        uri: >-
          arn:aws:apigateway:us-east-1:lambda:path/2015-03-31
            /functions/arn:aws:lambda:us-east-1:832720255830
            :function:SimpleDynamoDB/invocations
        passthroughBehavior: when_no_templates
        httpMethod: POST
        responses:
          default:
            statusCode: '200'
            responseParameters:
              method.response.header.Access-Control-Allow
                -Origin: '''*'''
            responseTemplates:
              application/json: |-
```

Paths

/customers

GET /customers

Parameters

Name	Located in	Required	Schema
lastname	query	No	string
firstname	query	No	string

Responses

Code	Description	Headers				Schema
		Name	Description	Type	Details	
200	200 response	Access-Control-Allow-Origin		"string"	Object	CustomerQueryRSP[{ }]

Try this operation

POST /customers

Parameters

Name	Located in	Required	Schema
CustomerPOST	body	Yes	CustomerPOST { firstname: string * lastname: string * email: string * address: { } }

Responses

Code	Description	Headers				Schema
		Name	Description	Type	Details	
200	200 response	Access-Control-Allow-Origin		"string"	Object	Customer { firstname: string * lastname: string * email: string * address: { } self: { } }

Try this operation

File ▾ Preferences ▾ Generate Server ▾ Generate Client ▾ Help ▾ ✔ Processe

```yaml
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
    get:
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: lastname
          in: query
          required: false
          type: string
        - name: firstname
          in: query
          required: false
          type: string
      responses:
        '200':
          description: 200 response
          schema:
            $ref: '#/definitions/CustomerQueryRSP'
          headers:
            Access-Control-Allow-Origin:
              type: string
      x-amazon-apigateway-integration:
        requestTemplates:
          application/json: |-
            #set($inputRoot = $input.path('$'))
            {
                "operation" : "query",
                "lastname" : "$input.params('lastname')",
                "firstname" : "$input.params('firstname')"
            }
        uri: >-
          arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us
            -east-1:832720255830:function:SimpleDynamoDB/invocations
        passthroughBehavior: when_no_templates
        httpMethod: POST
        responses:
          default:
            statusCode: '200'
            responseParameters:
              method.response.header.Access-Control-Allow-Origin: '''*'''
            responseTemplates:
              application/json: |-
                #set($inputRoot = $input.path('$'))
                $input.json('$.Items')
```

Scheme

https

Accept

application/json

Parameters

Method Execution

email

dff9@columbia.edu

```
GET https://jpzt51vrw8.execute-api.us-east-1.amazonaws.com/prod/customers/dff9@columbia.edu HTTP/1.1

Host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
Accept: application/json
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,fa;q=0.6,sv;q=0.4
Cache-Control: no-cache
Connection: keep-alive
Origin: http://editor.swagger.io
Referer: http://editor.swagger.io/
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36
```

⚠ This is a cross-origin call. Make sure the server at jpzt51vrw8.execute-api.us-east-1.amazonaws.com accepts GET requests from editor.swagger.io . Learn more

Send Request

Response

SUCCESS

Rendered | Pretty | Raw

```
{
  "firstname": "Donald",
  "lastname": "Ferguson",
  "email": "dff9@columbia.edu",
  "address": {
    "href": "../Addresses/98bb32d0-32b2-44f0-8a4f-31176ac17340"
  },
```

http://www.slideshare.net/VictorTrakhtenberg/swagger2

Swagger is…

Technology

A framework

for

- producing
- consuming
- visualizing

RESTful APIs

Methodology

A specification

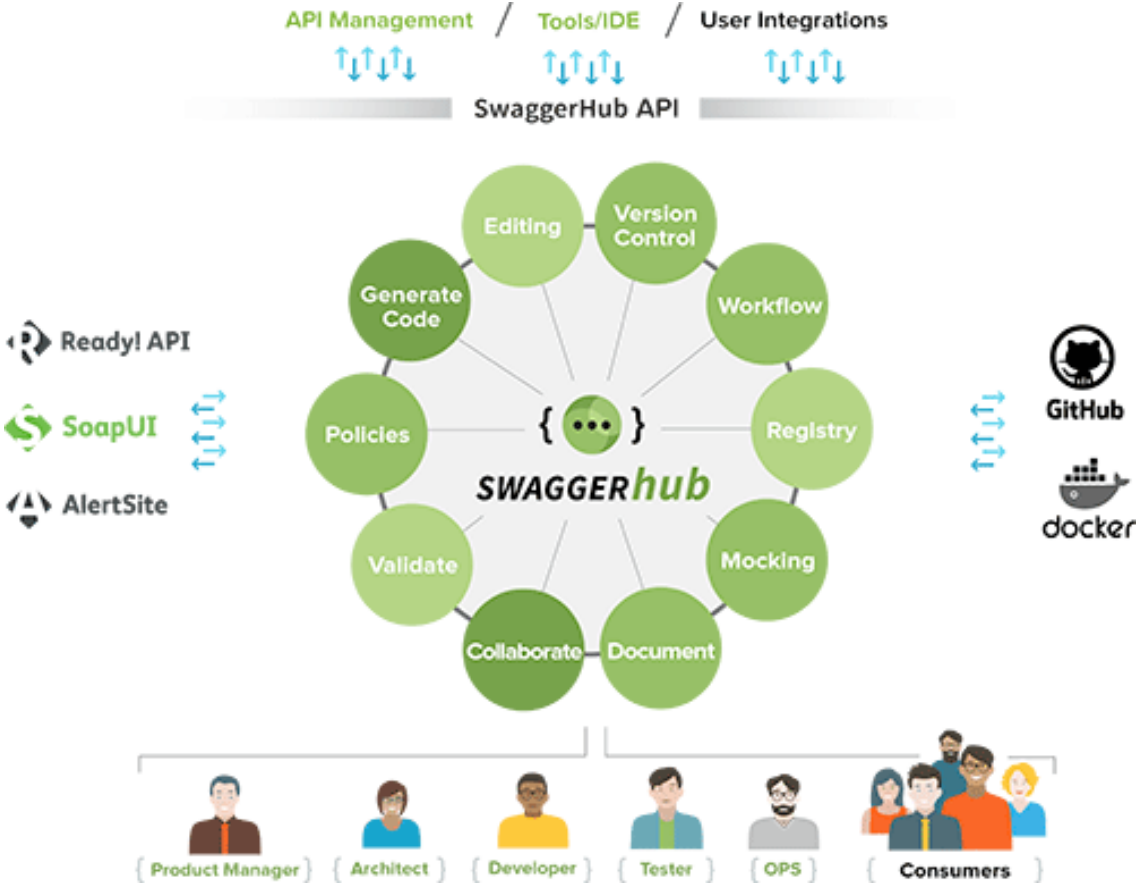
for

- describing
- documenting

RESTful APIs

Swagger Hub

https://smartbear.com/product/swaggerhub/features/



Swagger Hub
https://smartbear.com/product/swaggerhub/features/

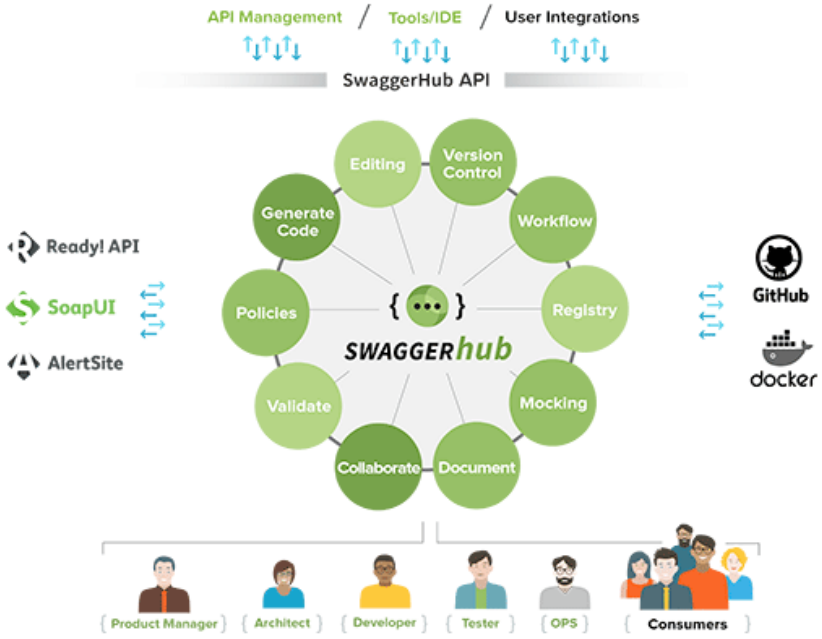
Design • Document • Discover

With SwaggerHub, you have the whole API Lifecycle at your fingertips:

- Collaborative design.
- Interactive documentation.
- And an easily searchable registry of Swagger-based APIs.

At its core, SwaggerHub is based on the Swagger principles of open, integrated technologies.

- Integrate with GitHub to protect your API versions.
- Link to DockerHub so API consumers can easily download and use your API.
- Use the SwaggerHub Registry API to integrate with our API directory.



Swagger Hub

- Methodology for
  - Designing
  - Building
  - Testing
  - Finding/Using

  "Services"
- Serverless/AWS is
  - Programming Model
  - For Implementing *methods* with
    - Agility
    - Flexibility
    - Efficiency
    - … …

Definition Editor

Use our intuitive editors to create your API definition and collaborate with others on their API definitions. On-the-fly validation keeps you honest.

LEARN MORE

Code Gen

Get your development project off to a quick start by using our client and server code templates.

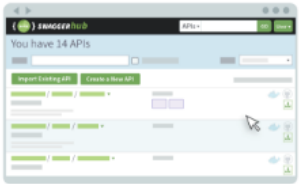
LEARN MORE

Versioning

Manage different versions of your API definition and decide for yourself when to publish a version or push it to GitHub.

LEARN MORE

API Registry



Browse our list of Swagger-based APIs and explore them using our interactive documentation.

LEARN MORE

Domains

Store all your re-usable components that can later be used across multiple API definitions, saving you time and maximizing productivity

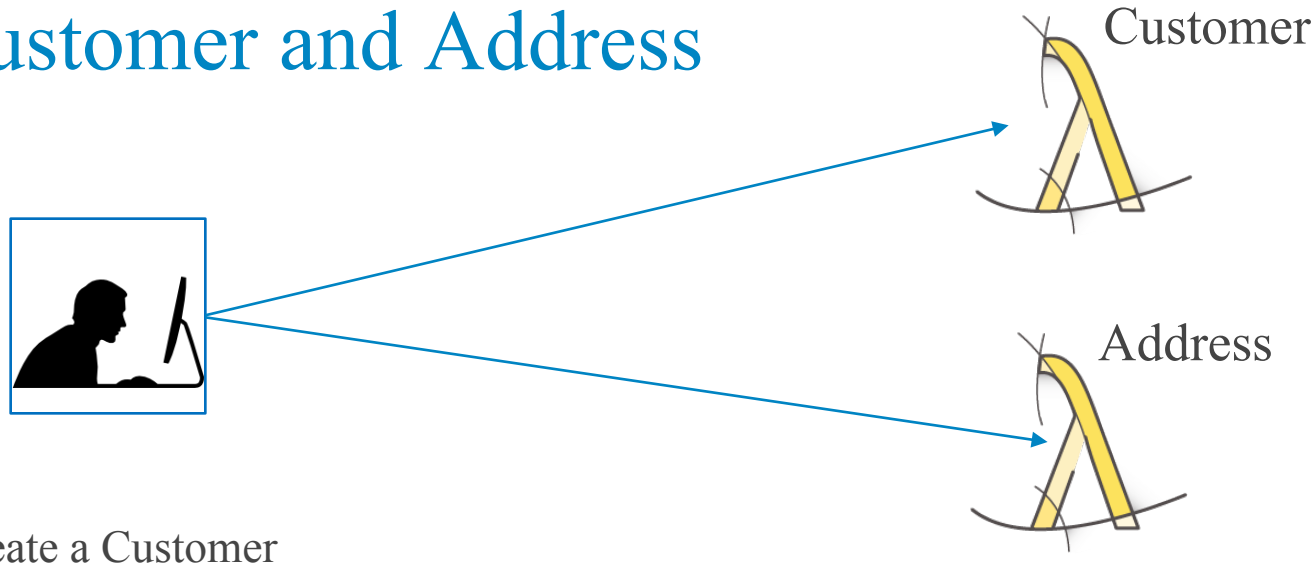
LEARN MORE

Plugins

LEARN MORE

Address Verification

Customer and Address



Create a Customer

- I can enter the address information and customer info on one “form” and make two calls.
- How do I know if
  - The address does/does not already exist?
  - Do I completely match the text? What if one used “S. Salem” and one used “South Salem?”
  - If the address already exists, how do I give the user the option of choosing without showing all of the address in a zipcode?
  - … ...

Reuse an API

Why write a service
And incrementally
Build data if
Someone has
Already done it?

For example
https://smartystreets.com/demo/api

Input Fields

Try a sample

auth-id	21102174564513388	-
Need help?		
candidates	10	-
Need help?		
street	1600 Pensilvania Ave	-
Need help?		
city	Washington	-
Need help?		
state	DC	-
Need help?		
zipcode	value	-
Need help?		
Select a component name	value	-
Need help?		

+ add component

Submit Request

Example REST Call and Response

Request URL

```
https://api.smartystreets.com/street-address?auth-id=21102174564513388&candidates=10&street=1600%20Pensilvania%20Ave&city=Washington&state=DC&zipcode=&
```

HTTP Status Code

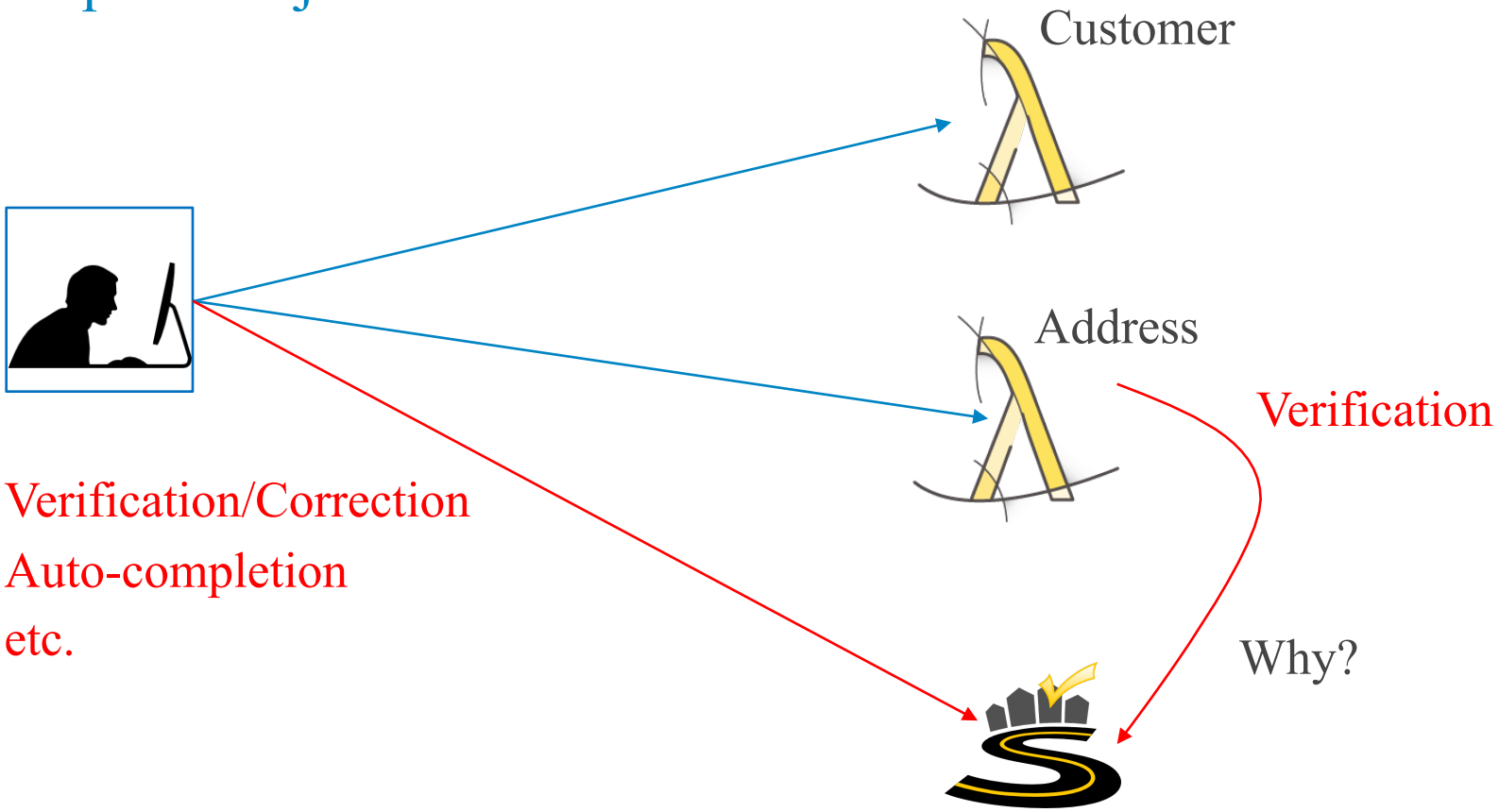
```
200
```


```
[
  {
    "input_index": 0,
    "candidate_index": 0,
    "delivery_line_1": "1600 Pennsylvania Ave SE",
    "last_line": "Washington DC 20003-3228",
    "delivery_point_barcode": "200033228992",
    "components": {
      "primary_number": "1600",
      "street_name": "Pennsylvania",
      "street_postdirection": "SE",
      "street_suffix": "Ave",
      "city_name": "Washington",
      "state_abbreviation": "DC",
      "zipcode": "20003",
      "plus4_code": "3228",
      "delivery_point": "99",
      "delivery_point_check_digit": "2"
    },
```


Some observations:

1. We will start to understand API security models.
2. Guesses corrections to incorrect data, e.g. misspelling “Pensilvania.” and missing zipcode.
3. Delivery_point_barcode provides unique ID, and duplicate detection.

Next Steps in Project



- Verification/Correction
- Auto-completion
- etc.