Desinging and implementing Microservices Tools

2016 Winner ORACLE Oracle Excellence Award
SPECIALIZED PARTNER OF THE YEAR: MIDDLEWARE — LATIN AMERICA

AGENDA

2

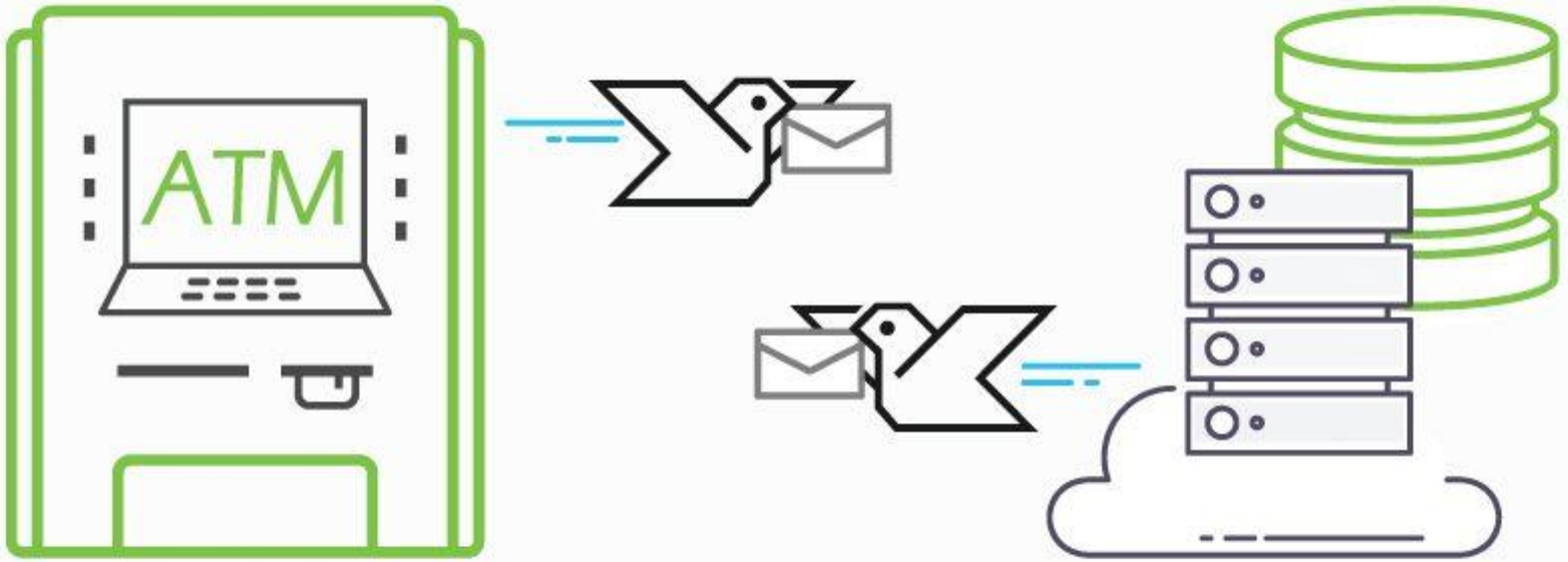
7. Introduction to APIs Lifecycle
8. API Gateway implementation
9. APIs design
10. Introduction to API Blueprint and Swagger
11. Deployment of APIs on microservices
12. Scalability for your microservices.

Certifica TIC Beyond Certification

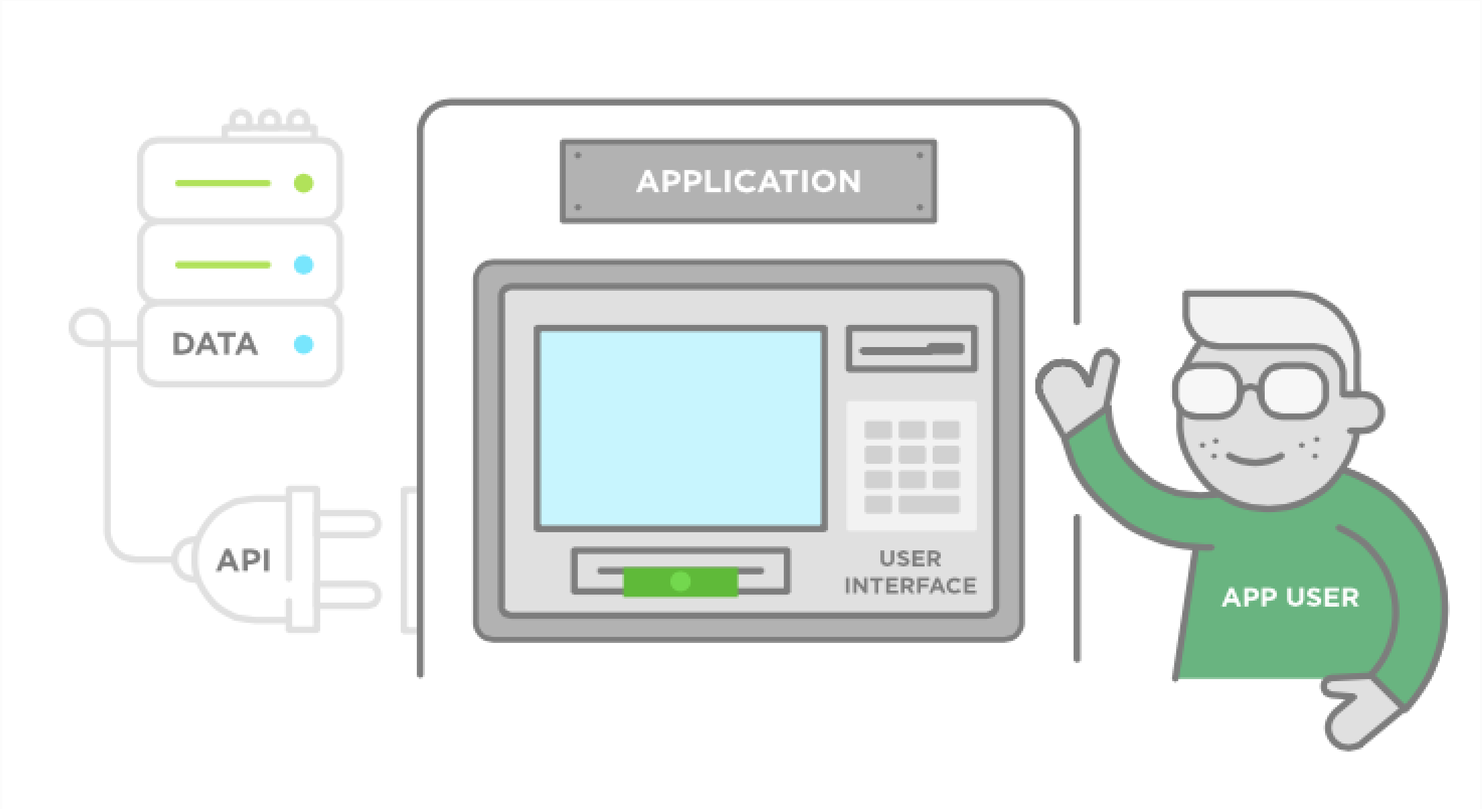
S&P Solutions

7. Introduction to APIs Lifecycle

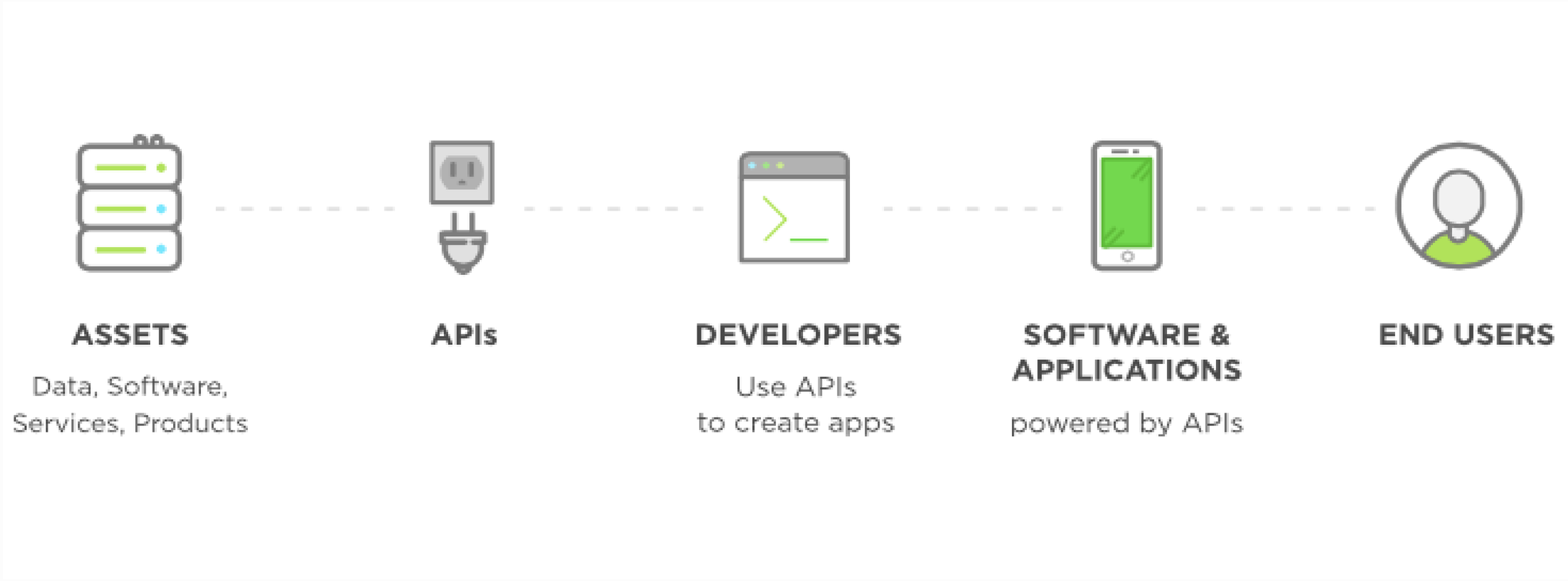
Messaging system



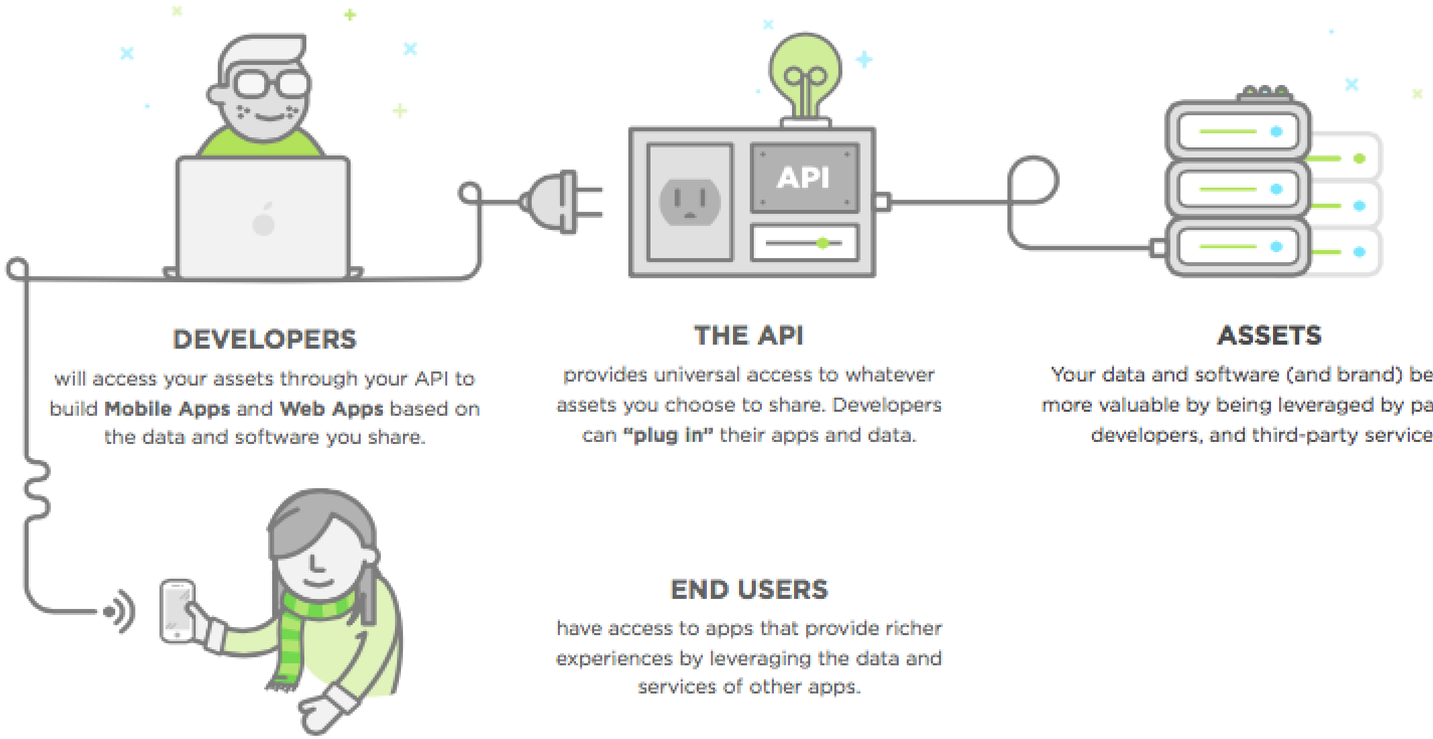
API flow



API Actors


ASSETS
Data, Software,
Services, Products
APIs
DEVELOPERS
Use APIs
to create apps
SOFTWARE &
APPLICATIONS
powered by APIs
END USERS

API Lifecycle



DEVELOPERS

will access your assets through your API to build **Mobile Apps** and **Web Apps** based on the data and software you share.

THE API

provides universal access to whatever assets you choose to share. Developers can **"plug in"** their apps and data.

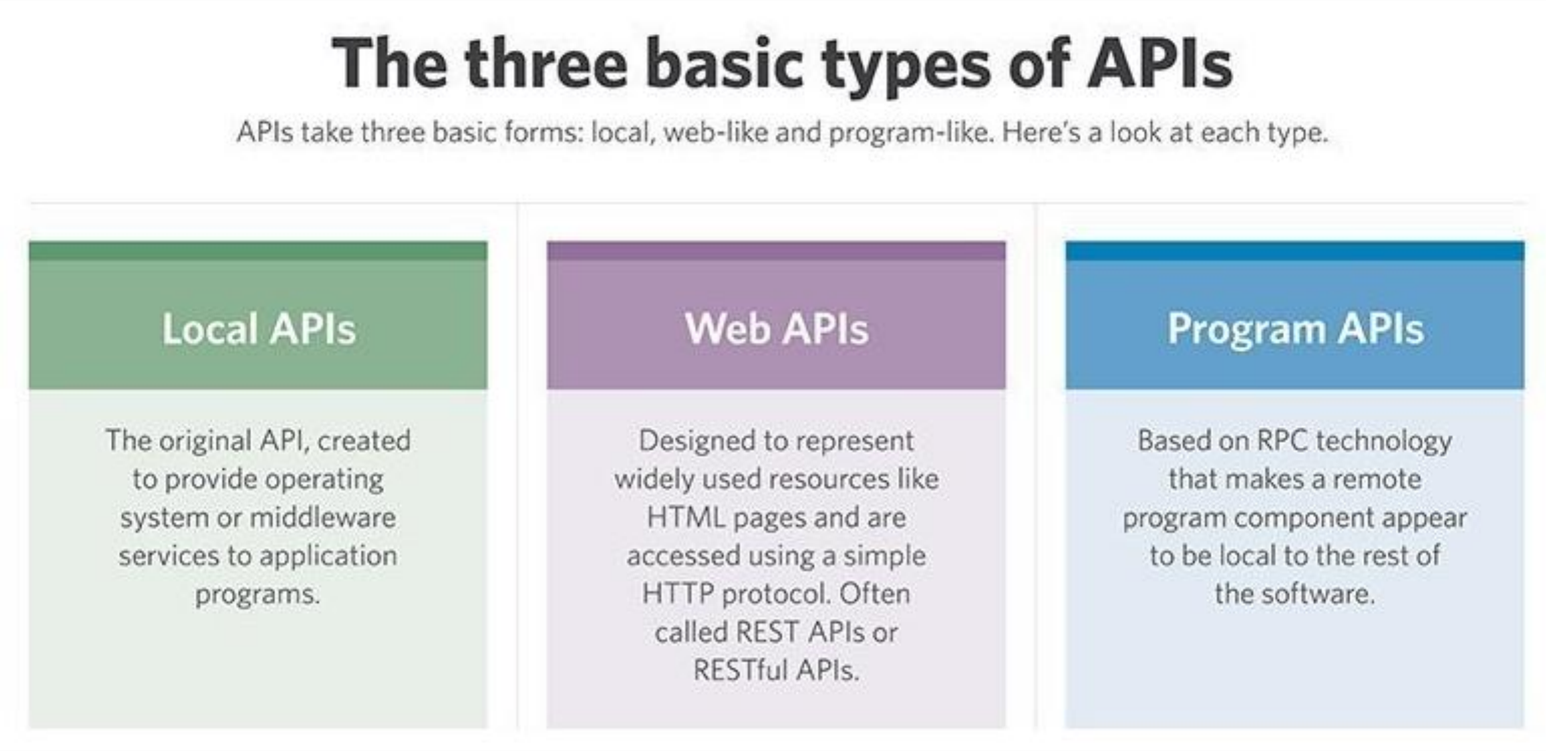
ASSETS

Your data and software (and brand) become more valuable by being leveraged by partners, developers, and third-party services.

END USERS

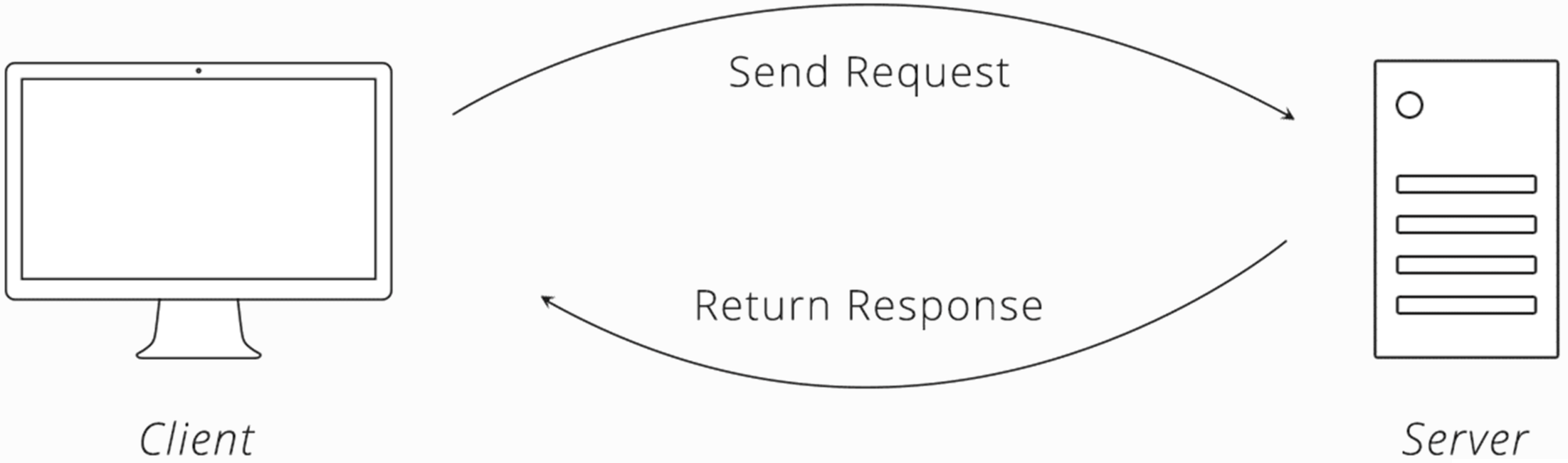
have access to apps that provide richer experiences by leveraging the data and services of other apps.

Types of API

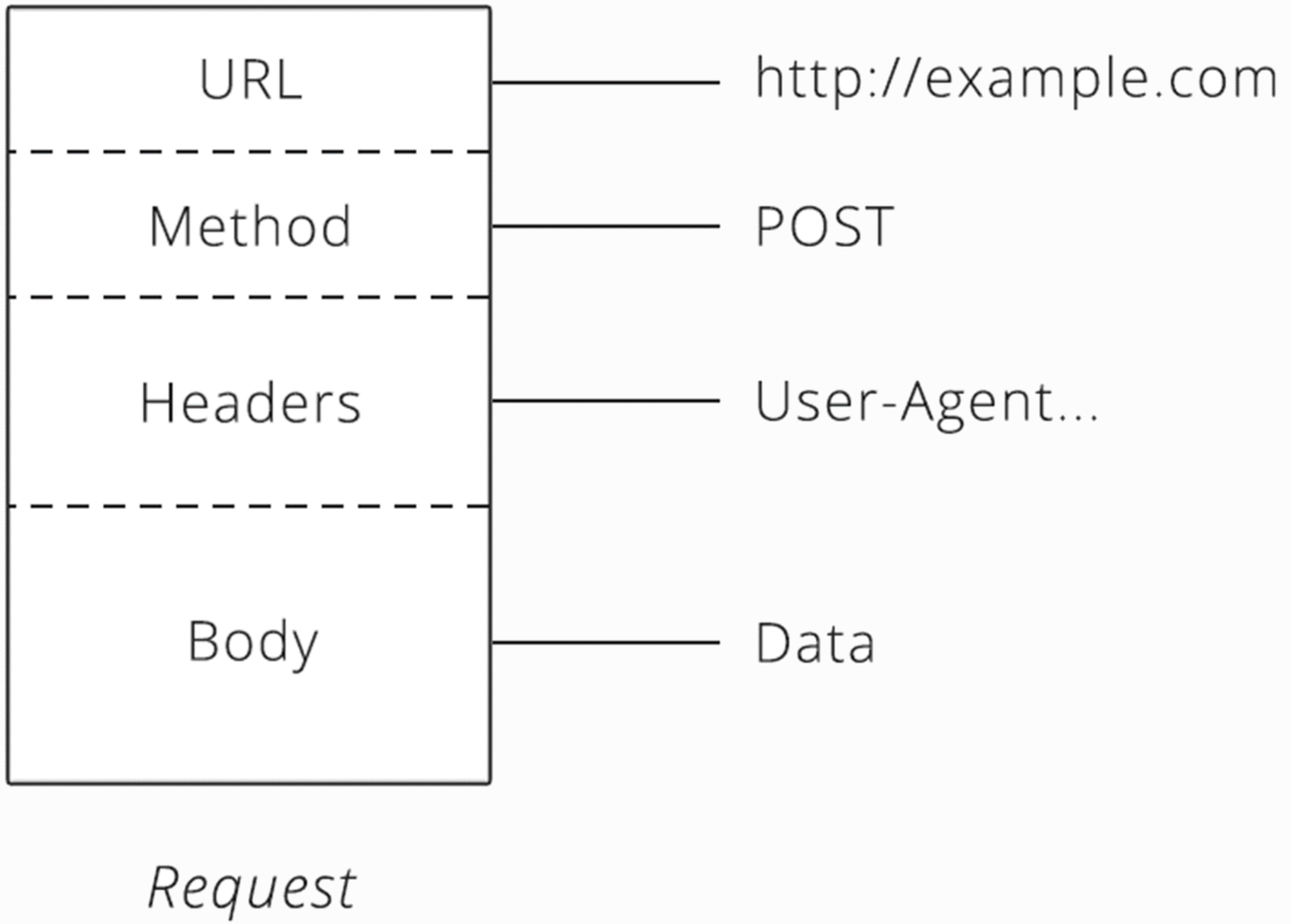


9. APIs design

Message Exchange Protocol



API Request



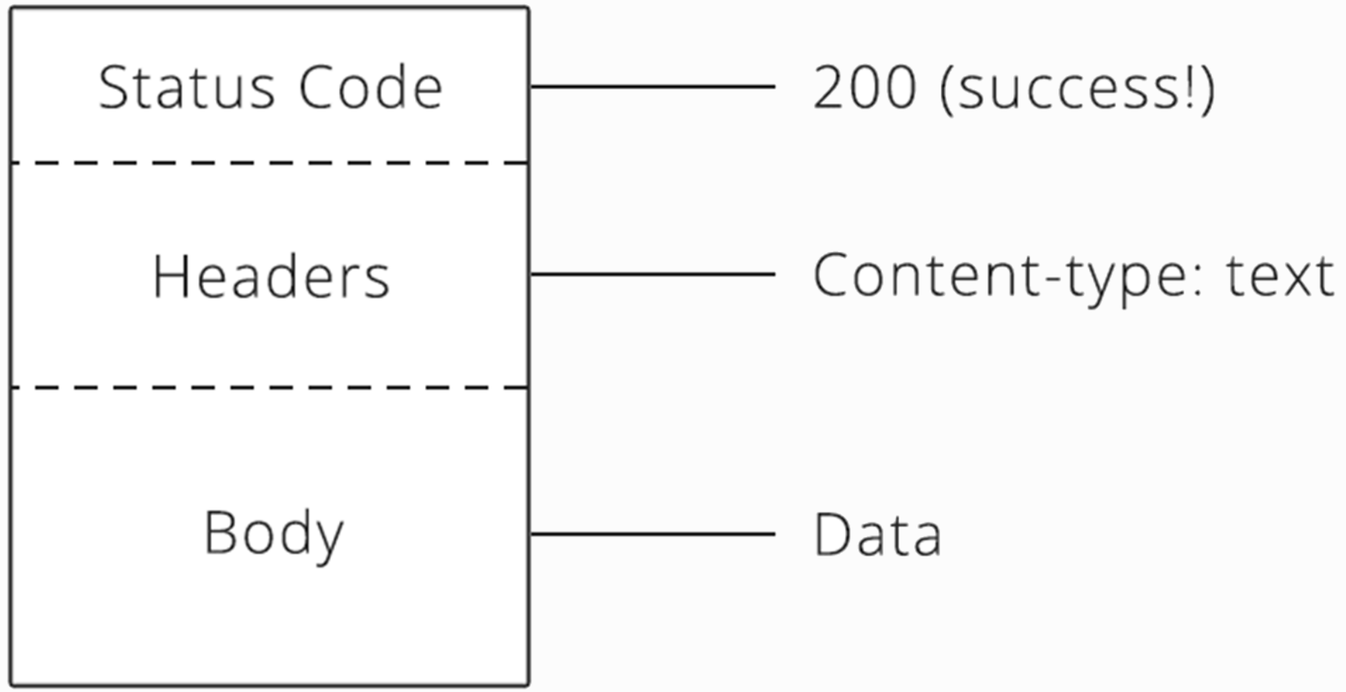
Request

API Response

Not Found

The requested URL / was not found on this server.

Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4 with Suhosin-Patch Se



Response

JSON

```
{ "crust": "original", "toppings": ["cheese", "pepperoni",
                "garlic"], "status": "cooking" }
```

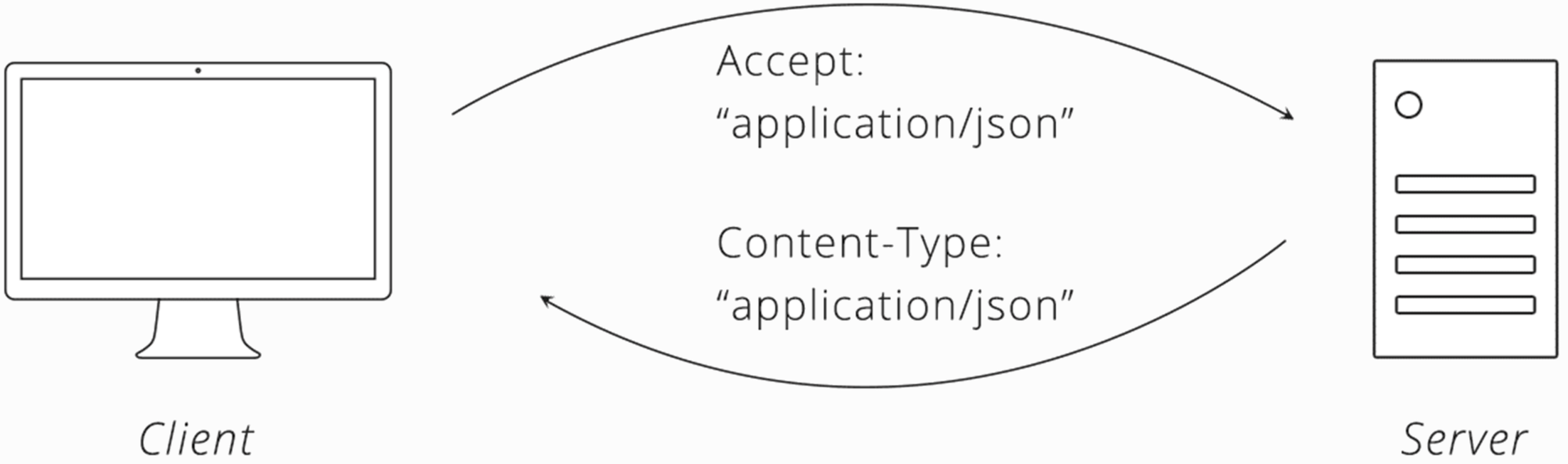
XML

```
<order>
    <crust>original</crust>
  - <toppings>
        <topping>cheese</topping>
        <topping>pepperoni</topping>
        <topping>garlic</topping>
    </toppings>
    <status>cooking</status>
</order>
```

node opening tag *value* *node closing tag*

<crust>original</crust>

Content type



10. Introduction to API Blueprint and Swagger

API Blueprint

- Description and specification language for web APIs (http://apiblueprint.org)
- From Apiary.io - but can be used independently from that
- Language spec: https://github.com/apiaryio/api-blueprint
- Offers platform-neutral documentation and additional tooling
  - Aglio
  - API-Mock

API Blueprint

Markdown-based

```
FORMAT: 1A

# Message of the Day API
A simple [MOTD](http://en.wikipedia.org/wiki/motd) API.

# Message [/messages/{id}]
This resource represents one particular message identified by its
*id*.

## Retrieve Message [GET]
Retrieve a message by its *id*.

+ Response 200 (text/plain)

        Hello World!

## Delete Message [DELETE]
Delete a message.
**Warning:** This action **permanently** removes the message from
the database.
```

API Blueprint

MSON

```
Shop Entry
- id: 1
- name: A green SUV
- price: 12.50
- tags: home, green
```

JSON

```
{
    "id": "1",
    "name": "A green SUV",
    "price": "12.50",
    "tags": [ "home", "green" ]
}
```

JSON Schema

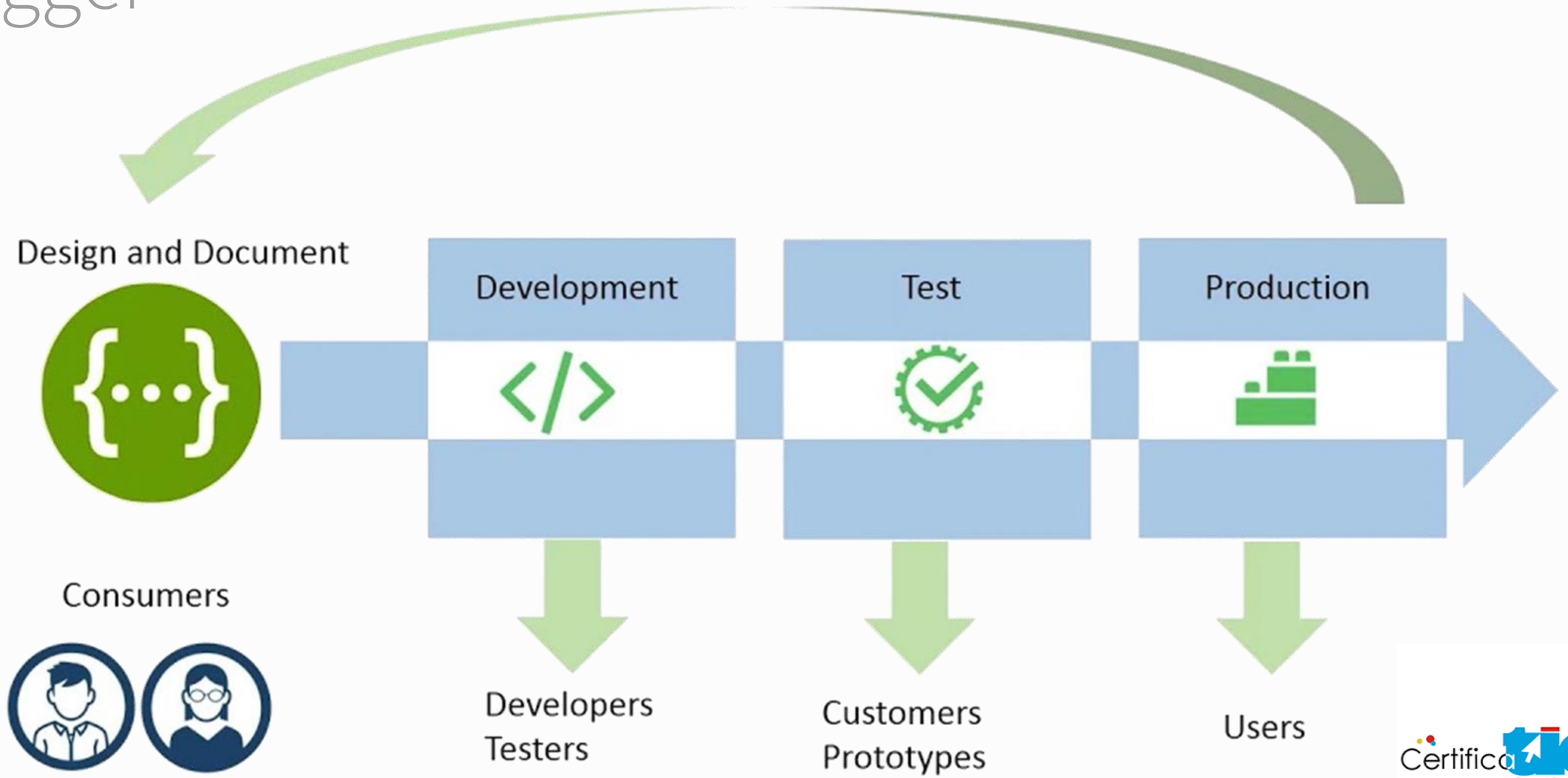
```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    “name”: “Shop Entry”,
    "type": "object",
    "properties": {
        "id": {
            "type": "number"
        },
        "name": {
            "type": "string"
        },
        "price": {
            "type": "number"
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    }
}
```

API Blueprint & MSON

Open Source

- Rich set of tools
- Open Specification
  - https://apiblueprint.org/
- Source on github
  - https://github.com/apiaryio/api-blueprint/
  - https://github.com/apiaryio/mson

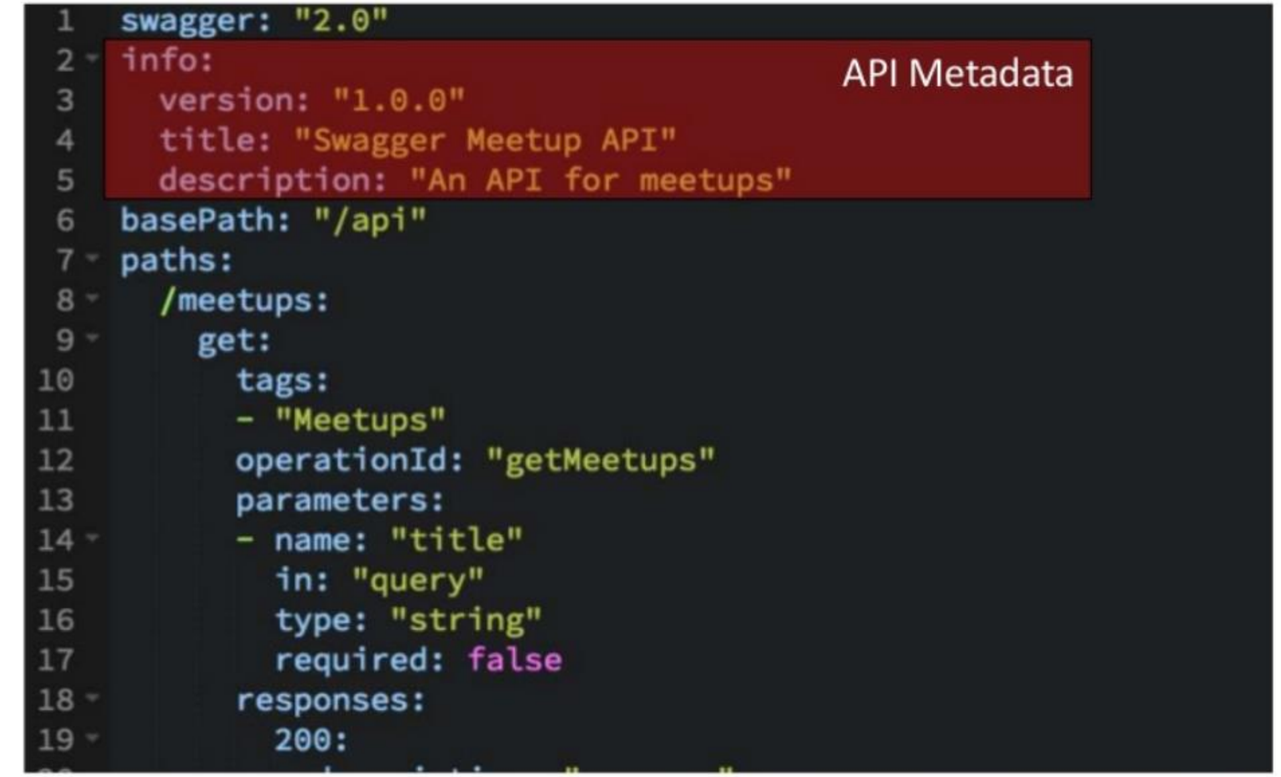
Swagger


Design and Document
Development
Test
Production
Consumers
Developers
Testers
Customers
Prototypes
Users

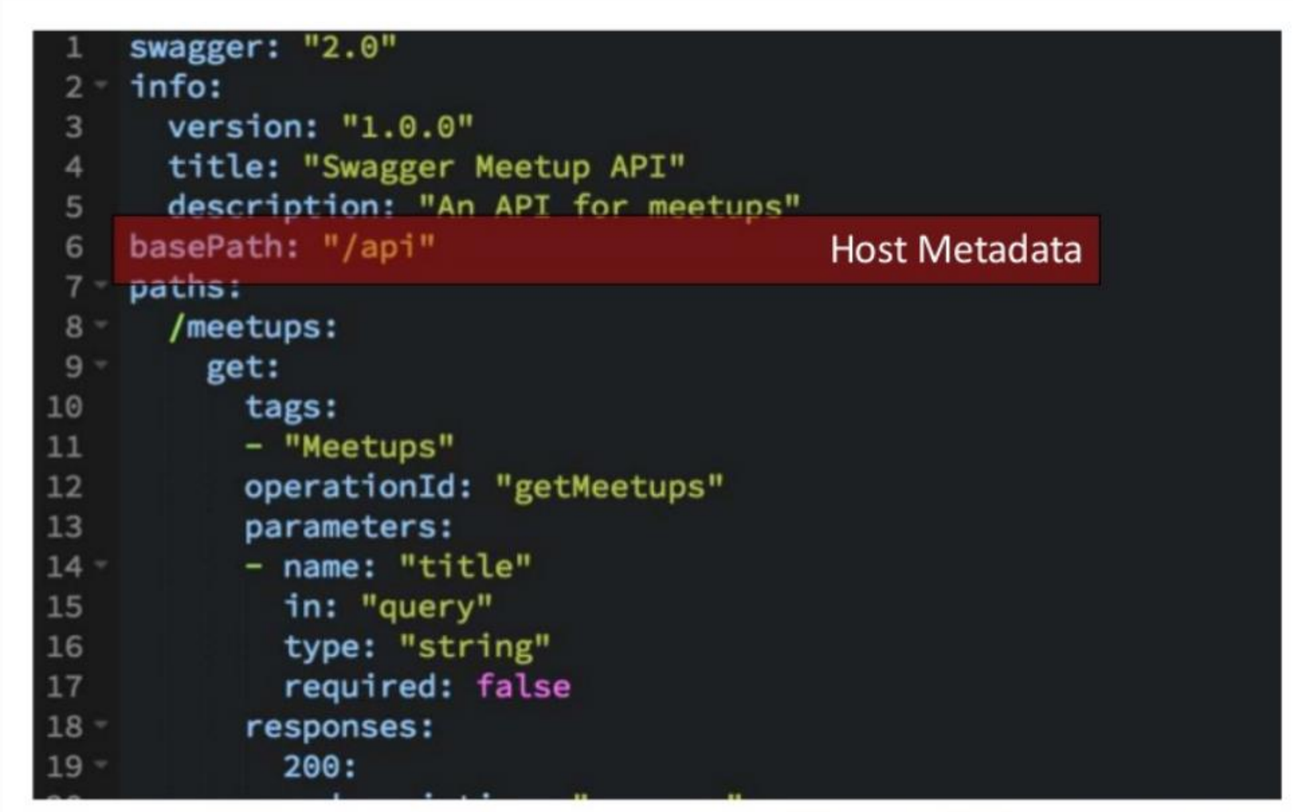
Swagger

- The basis of the Open API Specification
  - A Linux Foundation Project
  - https://openapis.org
- A simple, structured way to describe your API
- Methods, Resources, Parameters, media types
- *Everything* your consumers need to know to consume your API

API Metadata



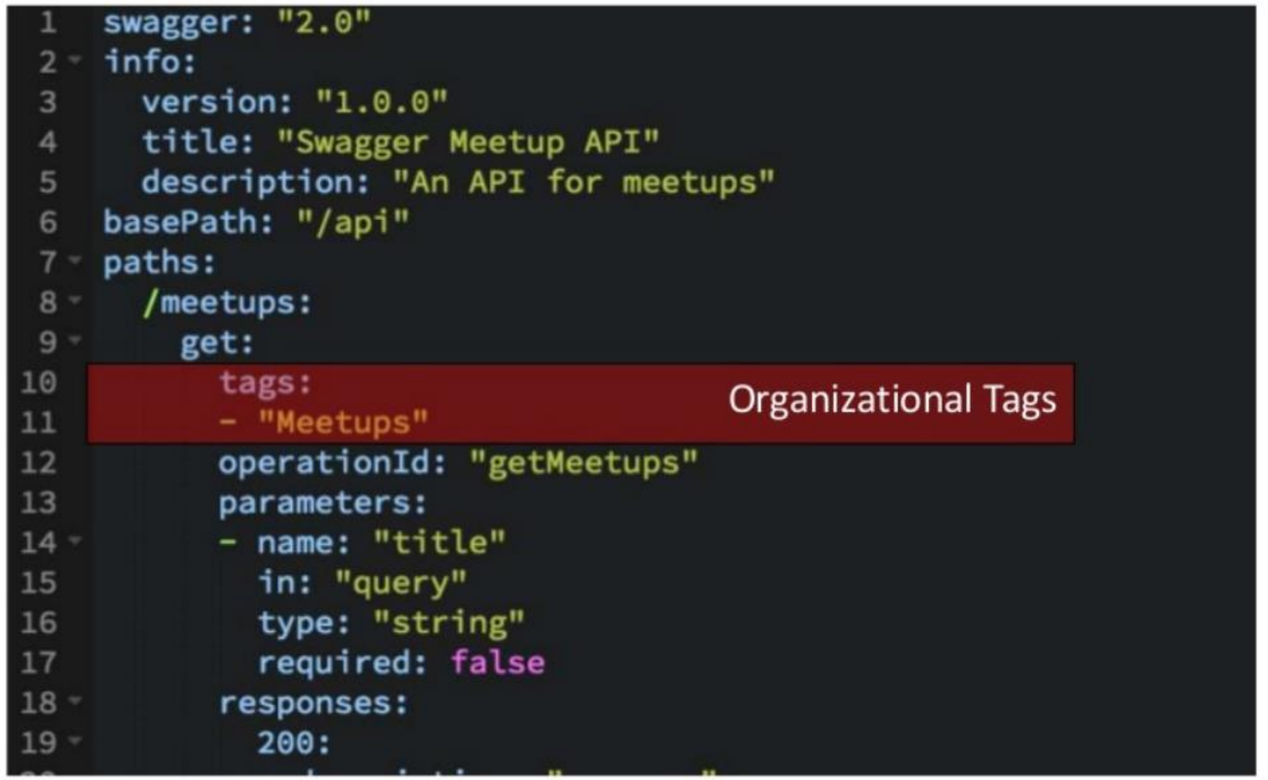
Host Metadata



Resources

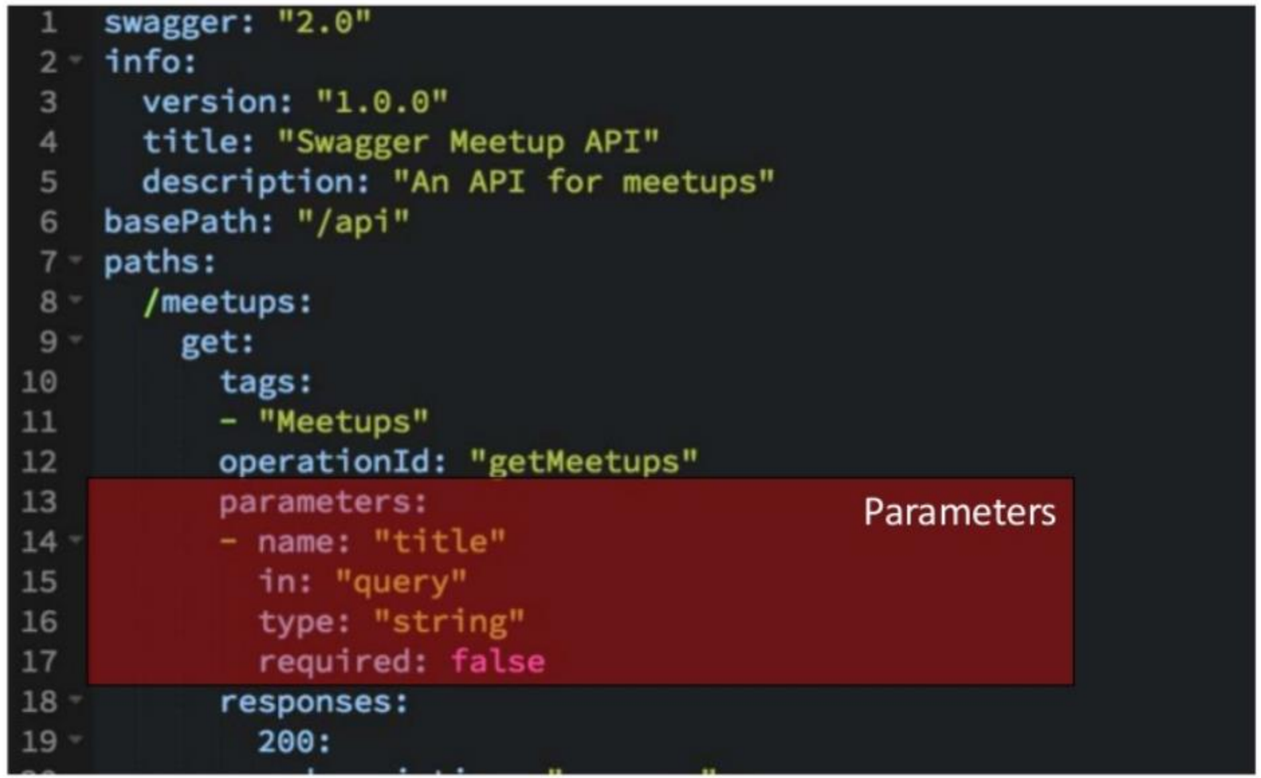

swagger: "2.0"
info:
version: "1.0.0"
title: "Swagger Meetup API"
description: "An API for meetups"
basePath: "/api"
paths:
/meetups:
get:
tags:
- "Meetups"
operationId: "getMeetups"
parameters:
- name: "title"
in: "query"
type: "string"
required: false
responses:
200:
Resources

Organizational Tags

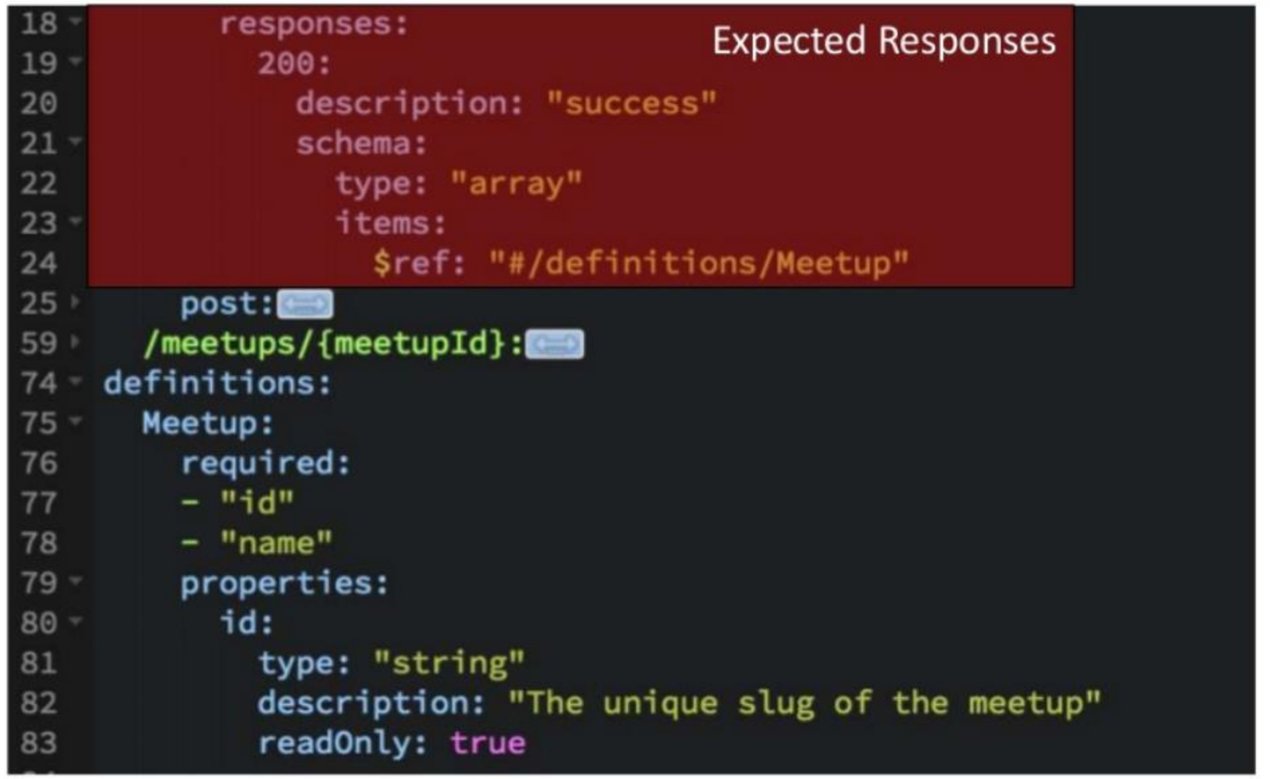


Codegen Hints

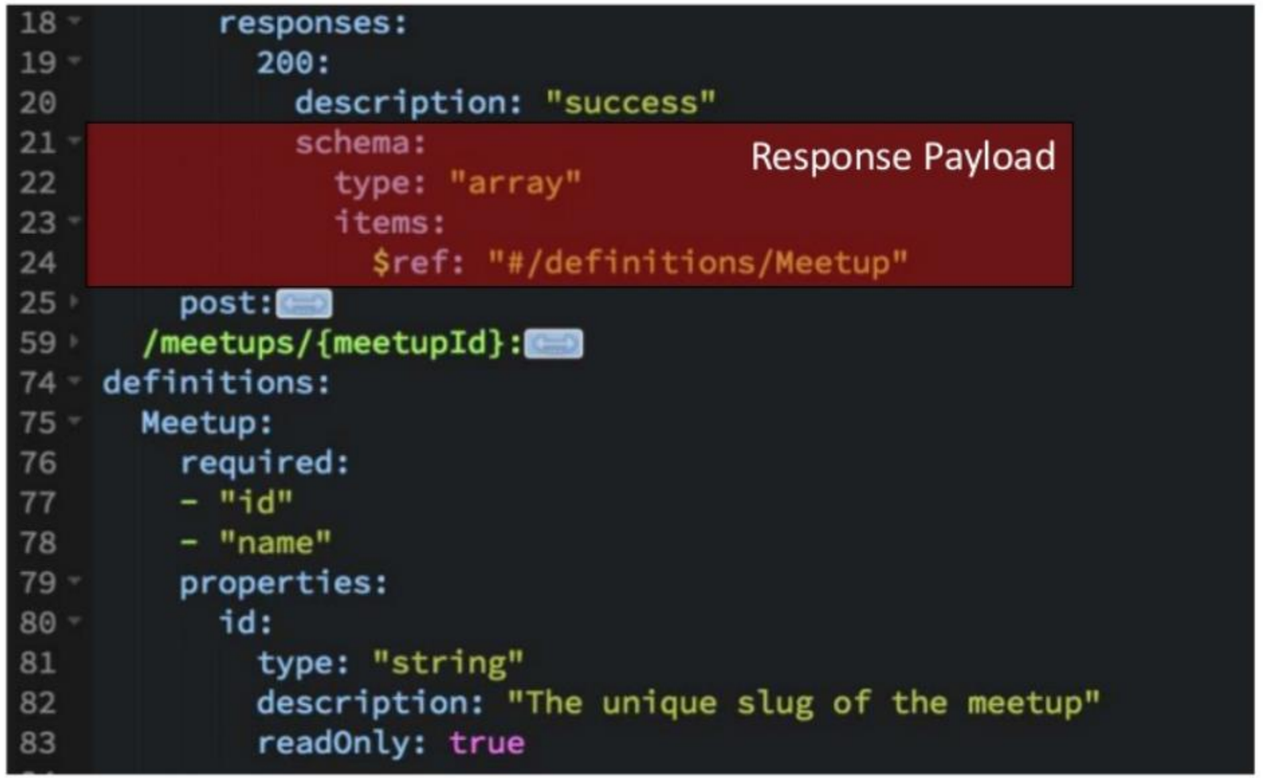
Parameters


swagger: "2.0"
info:
version: "1.0.0"
title: "Swagger Meetup API"
description: "An API for meetups"
basePath: "/api"
paths:
/meetups:
get:
tags:
- "Meetups"
operationId: "getMeetups"
parameters:
- name: "title"
in: "query"
type: "string"
required: false
responses:
200:
Parameters

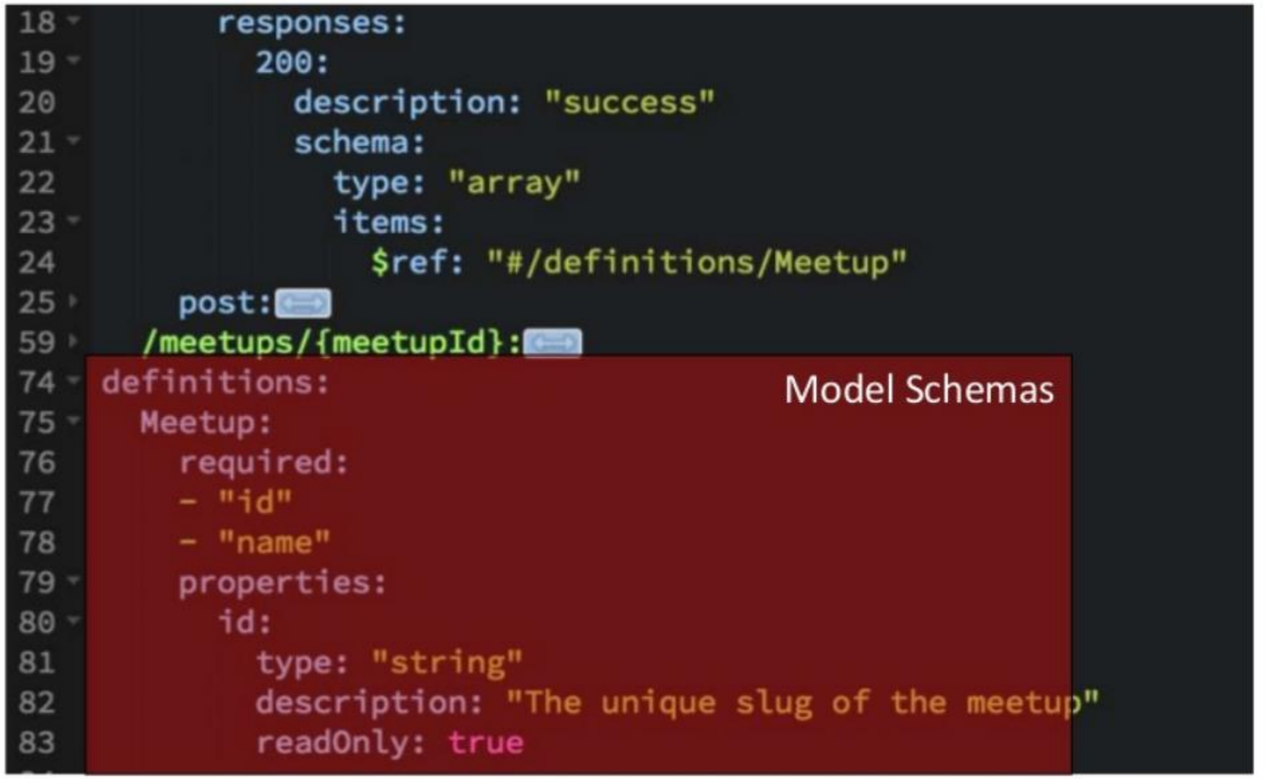
Expected Responses



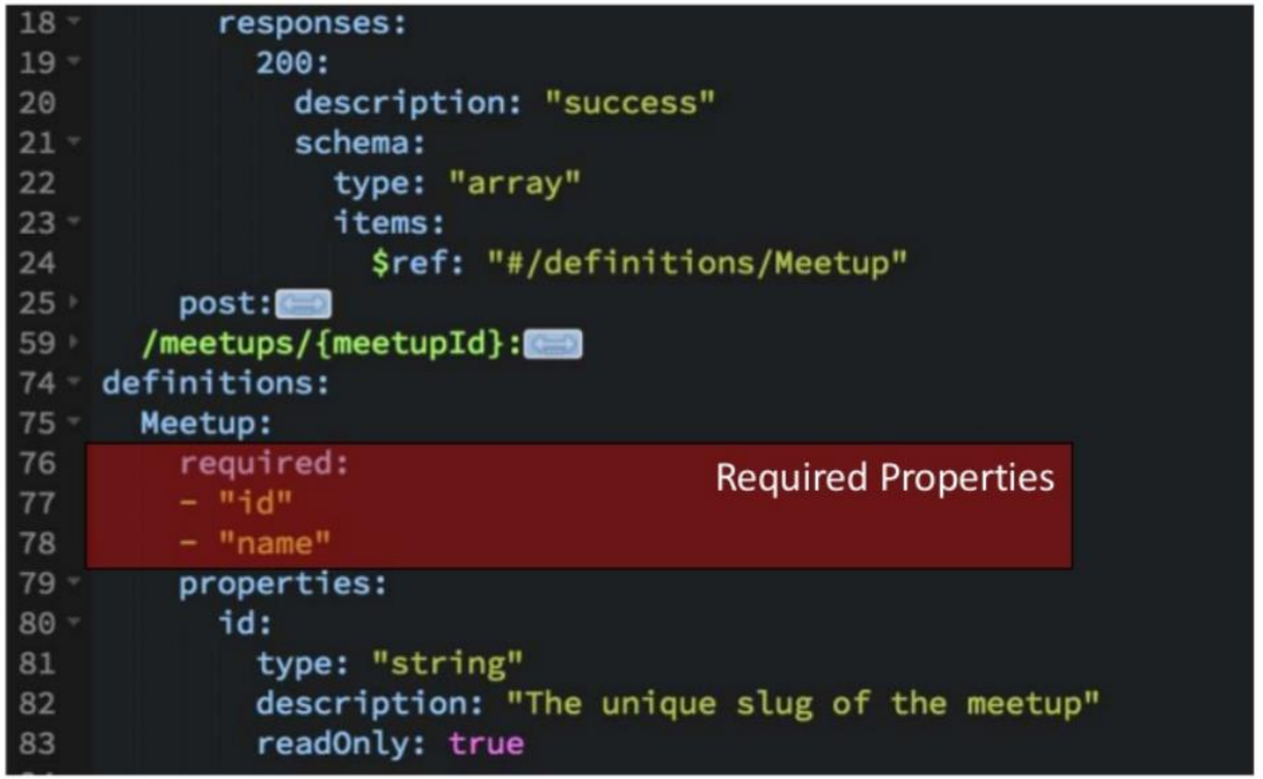
Response Payload



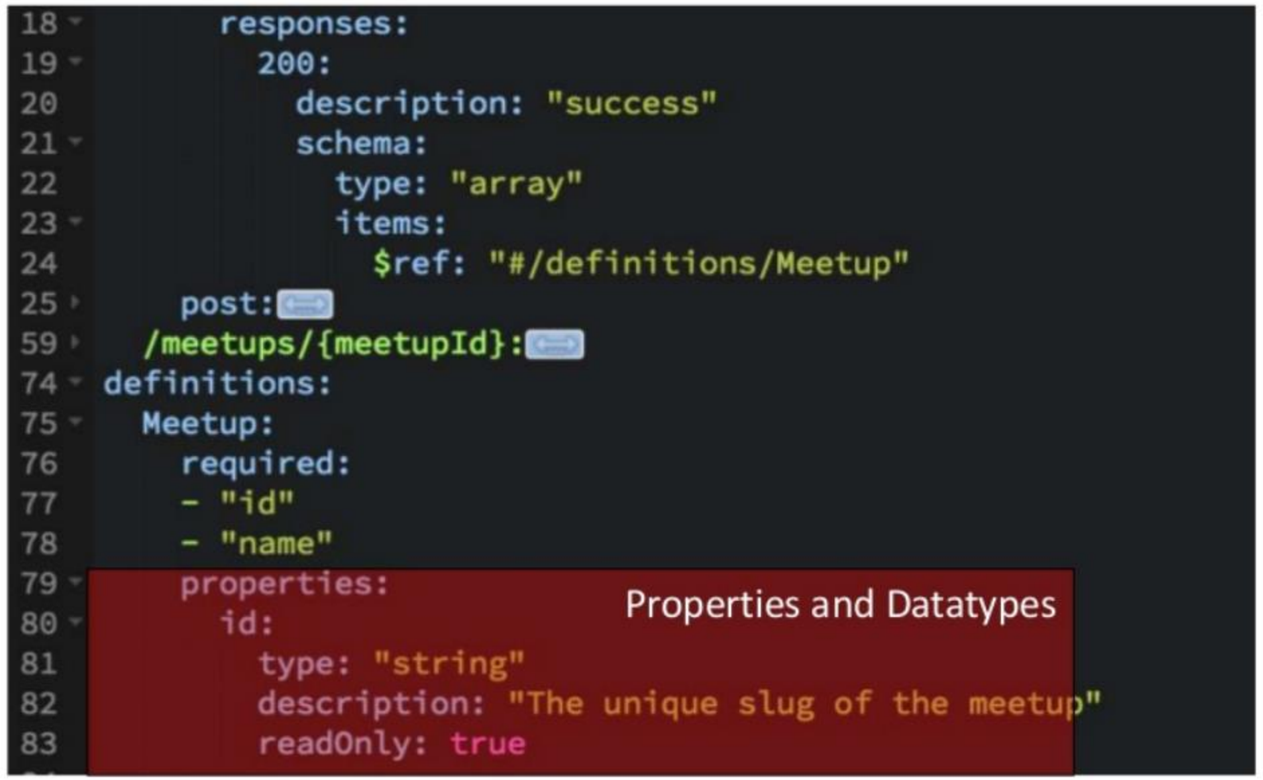
Model Schema



Required Properties



Properties and datatypes



Code First

- Code Annotations, comments for embedded documentation
  - Processed at compile-time, post-compile, runtime
  - Docs are always right! (code *is* the documentation)
    - Like Javadocs for REST APIs (but tons better)

Code first

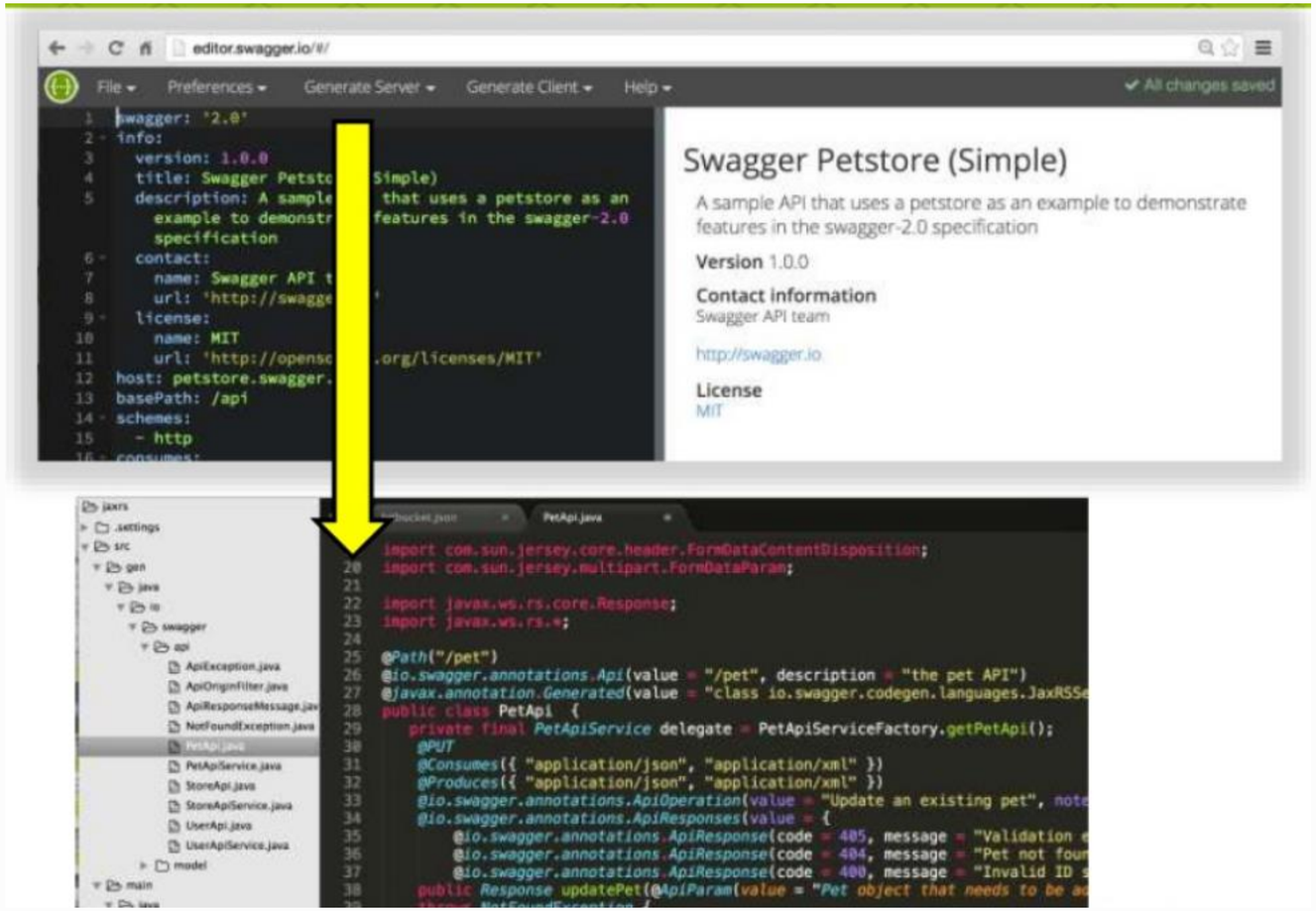
```
@Path("/pet")
@Api(value = "/pet", description = "Operations about pets", authorizations = {
  @Authorization(value = "petstore_auth",
  scopes = {
    @AuthorizationScope(scope = "write:pets", description = "modify pets in your account"),
    @AuthorizationScope(scope = "read:pets", description = "read your pets")
  })
}, tags = "pet")
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
public class PetResource {
  static PetData petData = new PetData();

  @GET
  @Path("/{petId}")
  @ApiOperation(value = "Find pet by ID",
    notes = "Returns a pet when ID <= 10.  ID > 10 or nonintegers will simulate API error conditions",
    response = Pet.class,
    authorizations = @Authorization(value = "api_key")
  )
  @ApiResponses(value = { @ApiResponse(code = 400, message = "Invalid ID supplied"),
      @ApiResponse(code = 404, message = "Pet not found") })
  public Response getPetById(
      @ApiParam(value = "ID of pet that needs to be fetched", allowableValues = "range[1,10]", required =
      throws NotFoundException {
```

Design First

- Start with the Swagger Definition as a blueprint for the implementation
  - Code from the definition
    - Manual! Swagger Definition is just the implementation *guideline*
    - Automated with swagger-codegen

Design First



Design vs Code mismatch

```java
@Path("/")
@io.swagger.annotations.Api(value = "/", tags = "Job"
public class JobApi  {
    private final JobApiService delegate = JobApiServ

    @Context HttpServletRequest request;
    @Context HttpServletResponse response;

    @POST
    @Path("/job")
    @io.swagger.annotations.ApiOperation(value = "",
    @io.swagger.annotations.ApiResponses(value = {
        @io.swagger.annotations.ApiResponse(code = 20

        @io.swagger.annotations.ApiResponse(code = 40
    public Response addJob(@ApiParam(value = "Job reg
    throws NotFoundException {
        final Timer.Context context = Service.metrics
        final Timer.Context defaultTimer = Service.me

        try {
            return delegate.addJob(request, response,
        } finally {
            context.stop();
            defaultTimer.stop();
        }
    }

    @DELETE
    @Path("/job/{jobId}")
    @io.swagger.annotations.ApiOperation(value = "",
    @io.swagger.annotations.ApiResponses(value = {
        @io.swagger.annotations.ApiResponse(code = 20
    public Response deleteJob(@ApiParam(value = "desc
    throws NotFoundException {
        final Timer.Context context = Service.metrics
        final Timer.Context defaultTimer = Service.me
```

```yaml
{
  swagger: "2.0",
- info: {
    description: "This is a sample server Petstc
    href="http://swagger.io">http://swagger.io</
    test the authorization filters",
    version: "1.0.0",
    title: "Swagger Petstore YAML",
    termsOfService: "http://swagger.io/terms/",
  - contact: {
      email: "apiteam@swagger.io"
    },
  - license: {
      name: "Apache 2.0",
      url: "http://www.apache.org/licenses/LIC
    }
  },
  basePath: "/v2",
- tags: [
  - {
      name: "pet",
      description: "Everything about your Pets
    - externalDocs: {
        description: "Find out more",
        url: "http://swagger.io"
      }
    },
  - {
      name: "store",
      description: "Operations about user"
    },
  - {
      name: "user",
      description: "Access to Petstore orders"
    - externalDocs: {
        description: "Find out more about our
        url: "http://swagger.io"
      }
```

Problem

- The Swagger Specification *should* be the source of truth
  - No round-trips to/from codegen
  - No out-of-date server based on code changes

Inflector

- Write your Specification
- Inflector wires your spec to the server
- Endpoints have automatic sample data
- Implementation is simply writing controllers
- Based on JAX-RS 2.0, Jersey 2.6

Inflector

```
controllerPackage: io.swagger.sample.controllers
modelPackage: io.swagger.sample.models
swaggerUrl: ./src/main/swagger/swagger.yaml
modelMappings:
  Pet: io.swagger.sample.models.Pet
```

inflector.yaml

swagger.yaml

```
paths:
  /pet:
    post:
      tags:
        - pet
      summary: Add a new pet to the store
      x-swagger-router-controller: SampleController
      description: ""
      operationId: addPet
      consumes:
        - application/json
        - application/xml
      produces:
        - application/xml
        - application/json
      parameters:
        - in: body
          name: body
          description: Pet object that needs to be a
          required: false
          schema:
            $ref: "#/definitions/Pet"
```

```
package io.swagger.sample.controllers;

public class SampleController {
    public ResponseContext addPet(RequestContext request, Pet body) {
```

8. API Gateway implementation

Comparison

Monolith vs. Microservices

A monolithic app

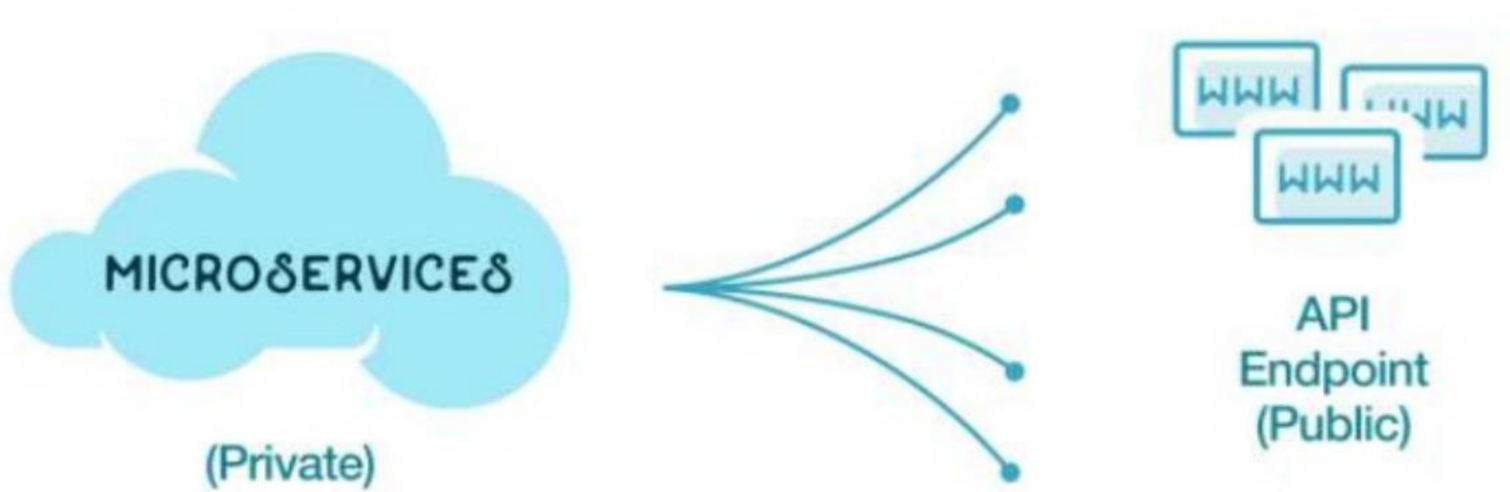
All or nothing — a big, giant thing that can't fit through the shipping door.

vs.

A microservice app

Continuous updating in parts.

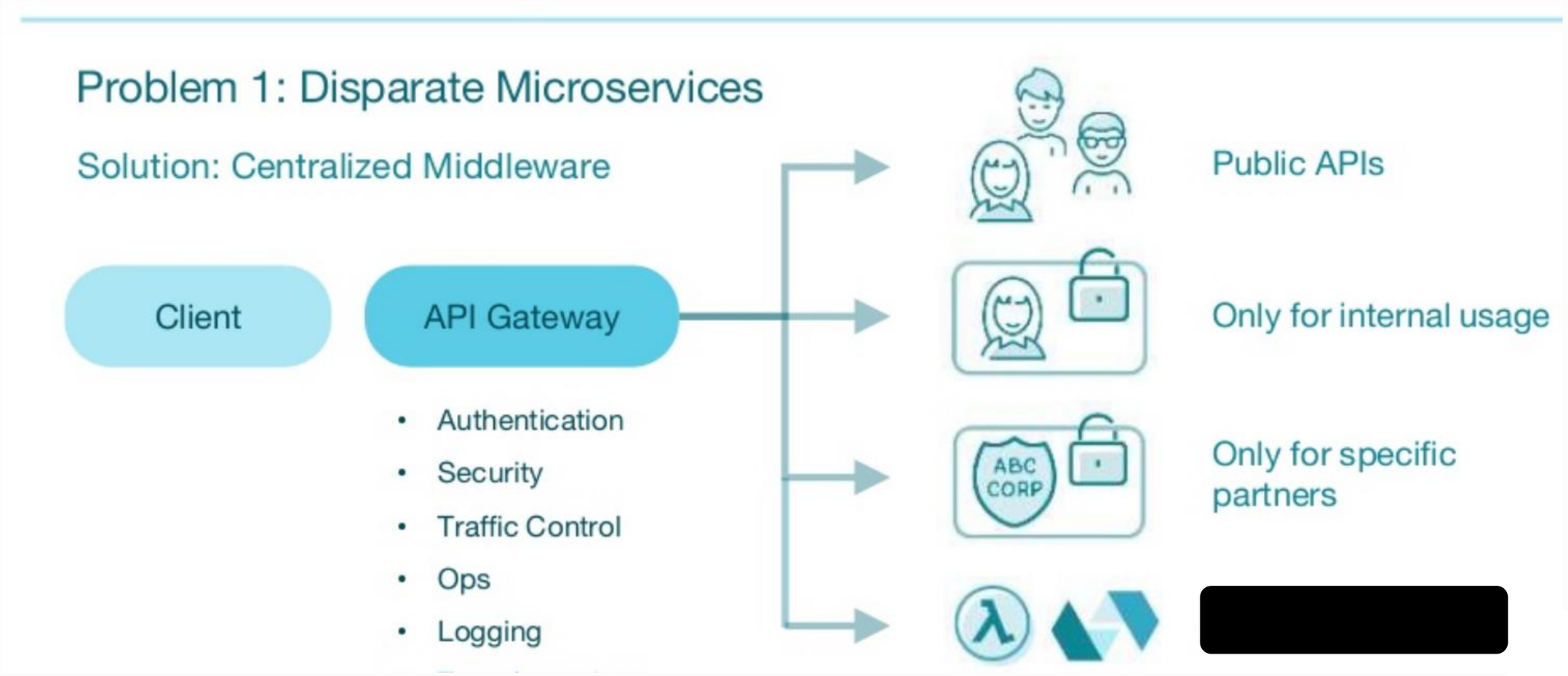
API Evolution



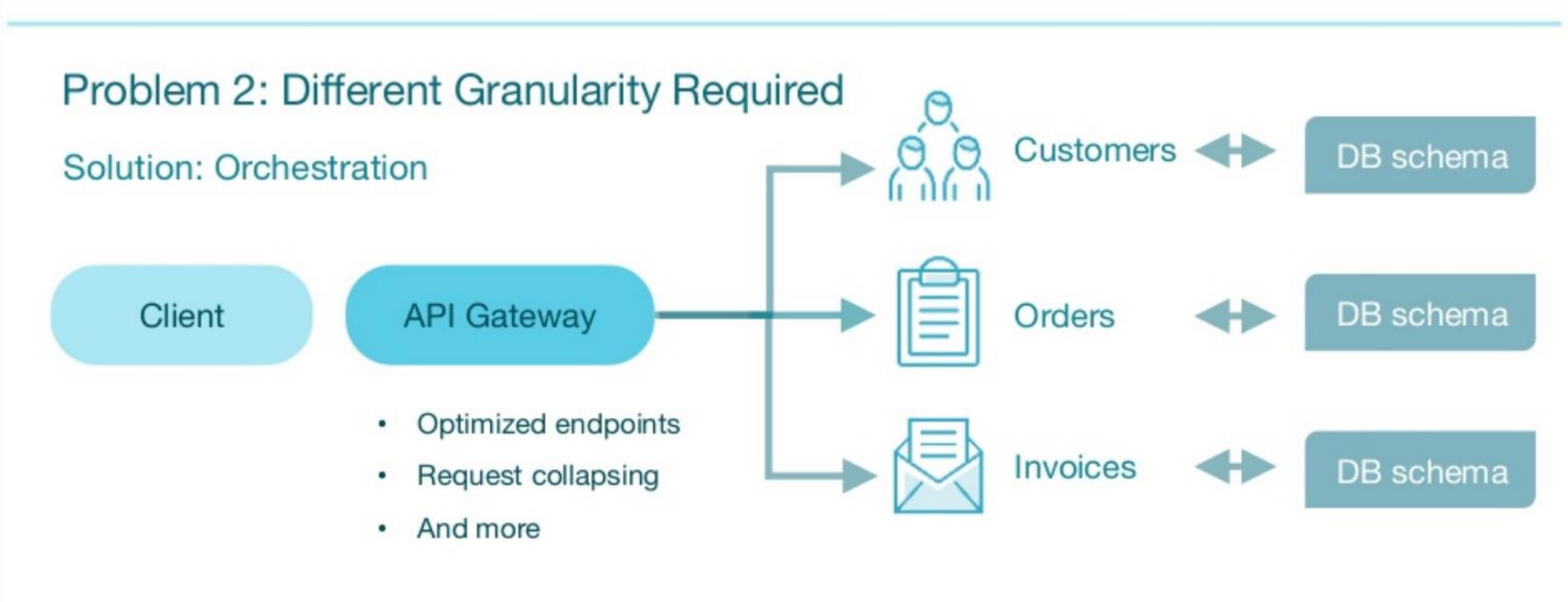
The natural evolution of microservices are APIs to the “N” client - client traditionally was built by the company, but now can be built by anyone

APIs = business leverage and amplification

Problems


Problem 1: Disparate Microservices
Solution: Centralized Middleware
Client
API Gateway
Authentication
Security
Traffic Control
Ops
Logging
Public APIs
Only for internal usage
ABC CORP
Only for specific partners
[redacted]

Problems


Problem 2: Different Granularity Required
Solution: Orchestration
Client
API Gateway
Optimized endpoints
Request collapsing
And more
Customers
Orders
Invoices
DB schema
DB schema
DB schema

API Manager Features

Centralized Declarative Configuration - Manage, Control and Visualize

API Consumer Management (users, apps, credentials, tokens etc.) – Get What You Need Most

Express Middleware Driven Plugins – Getting Started is Dead Simple

Security – Peace of Mind

- Basic Authorization
- Key Authorization
- Oauth 2.0
- CORS

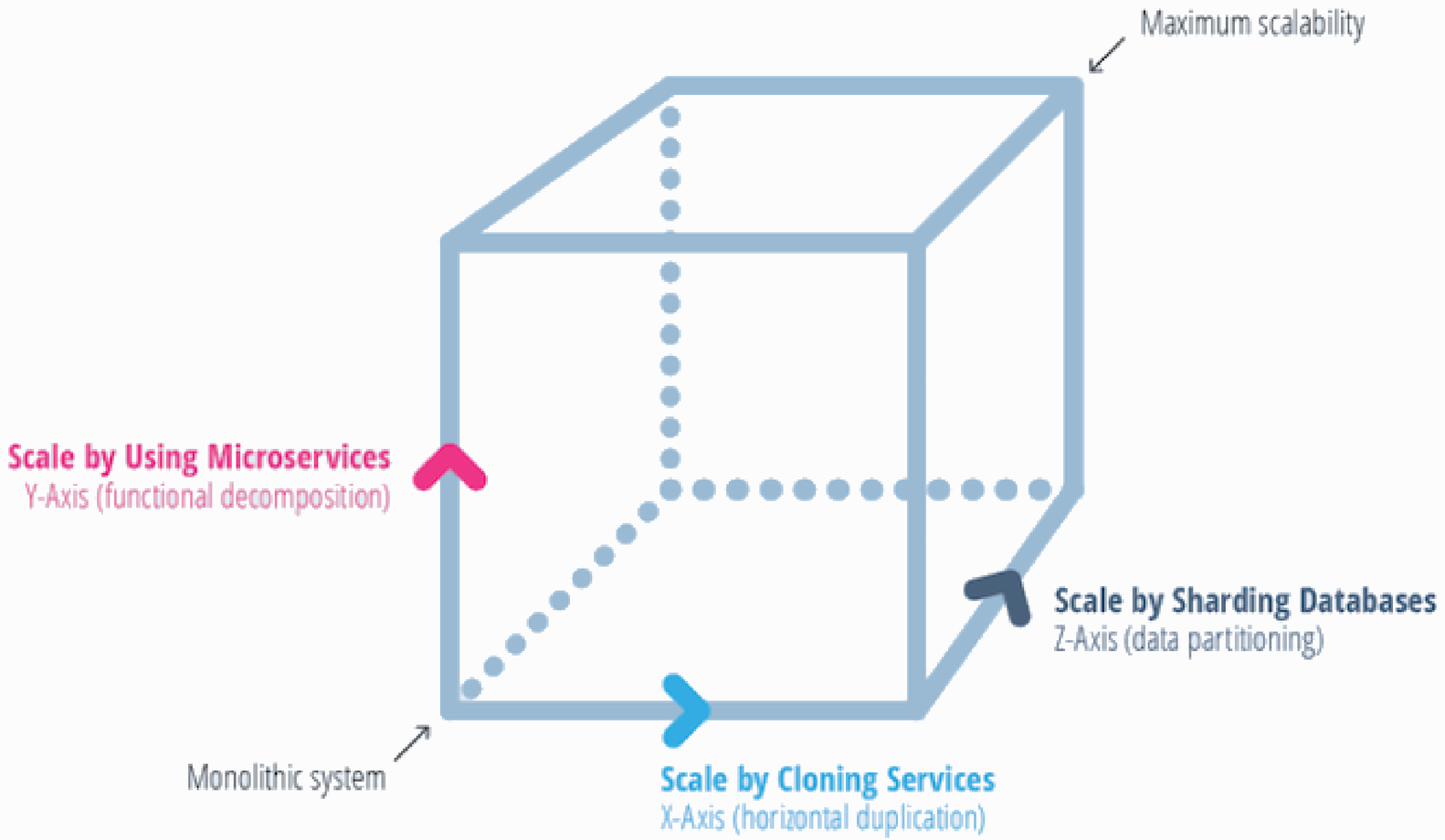
Quality of Services –Your Way

- Dynamic Routing
- Simple Logger
- Proxy with Load Balancer (coming soon)
- Rate Limiter

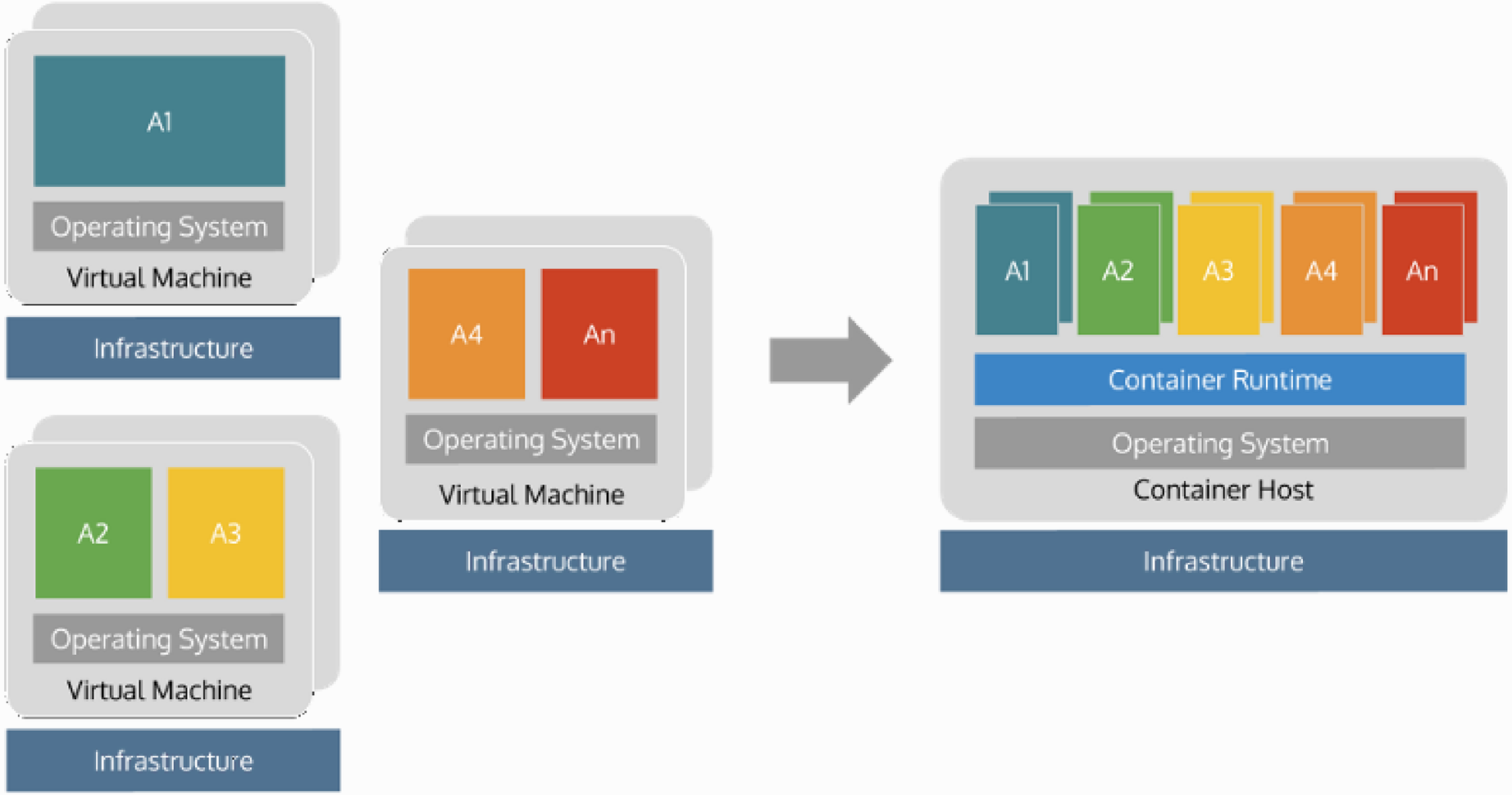
APIGEE

12. Scalability for your microservices.

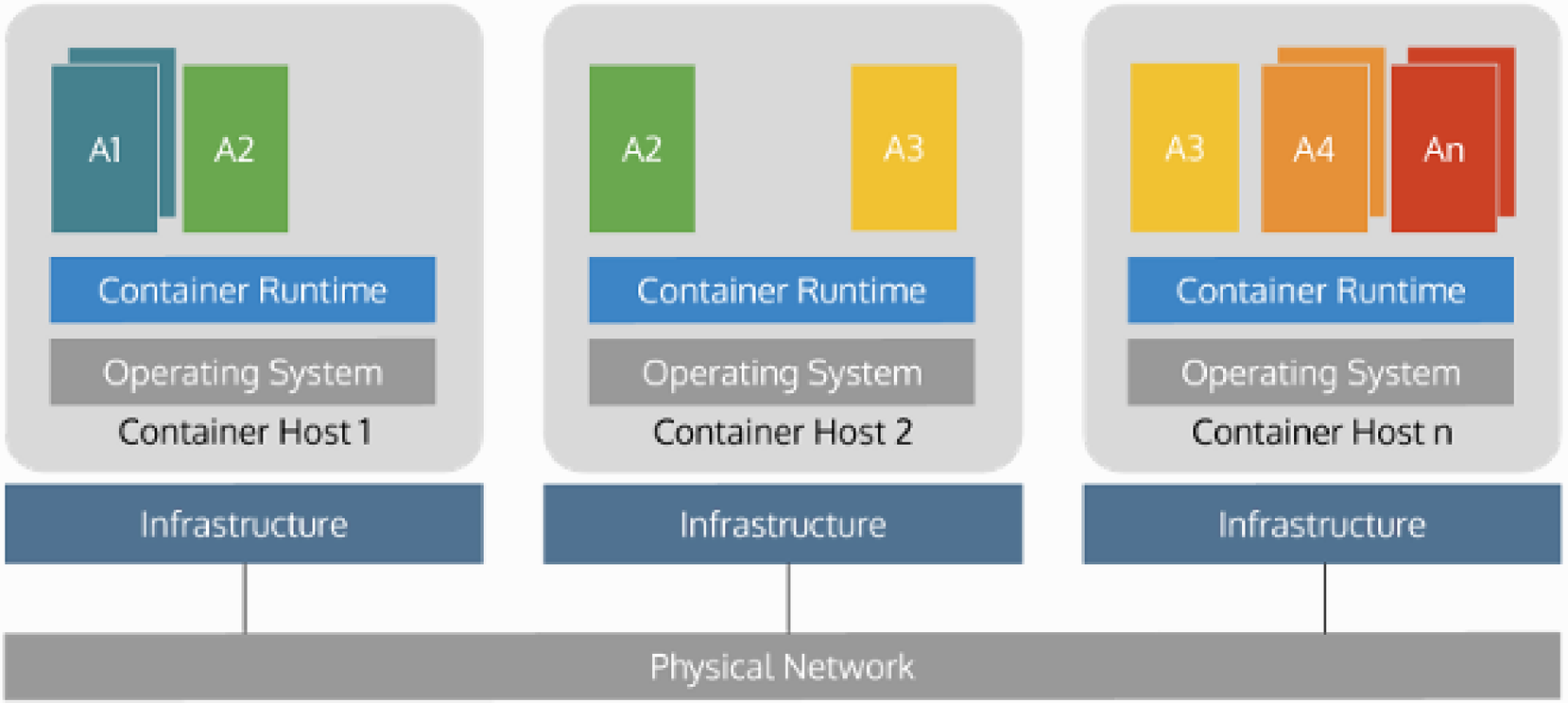
Scalability



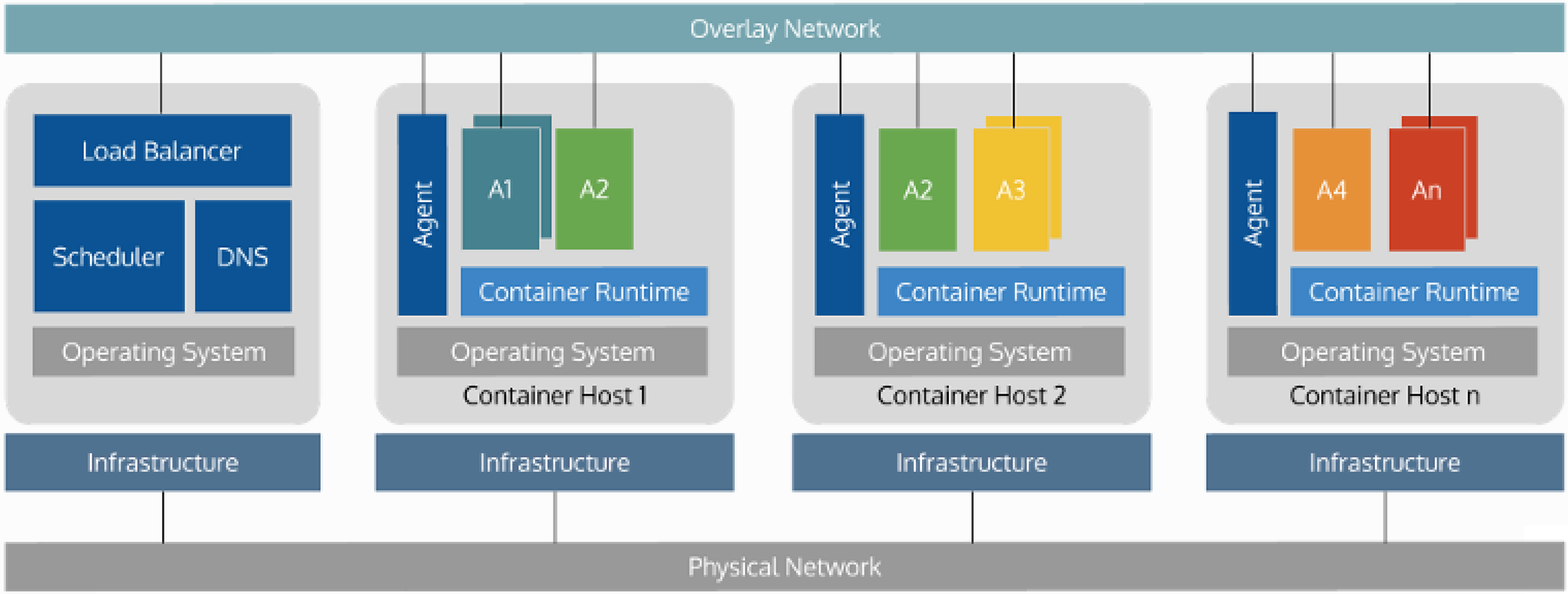
Cluster Management



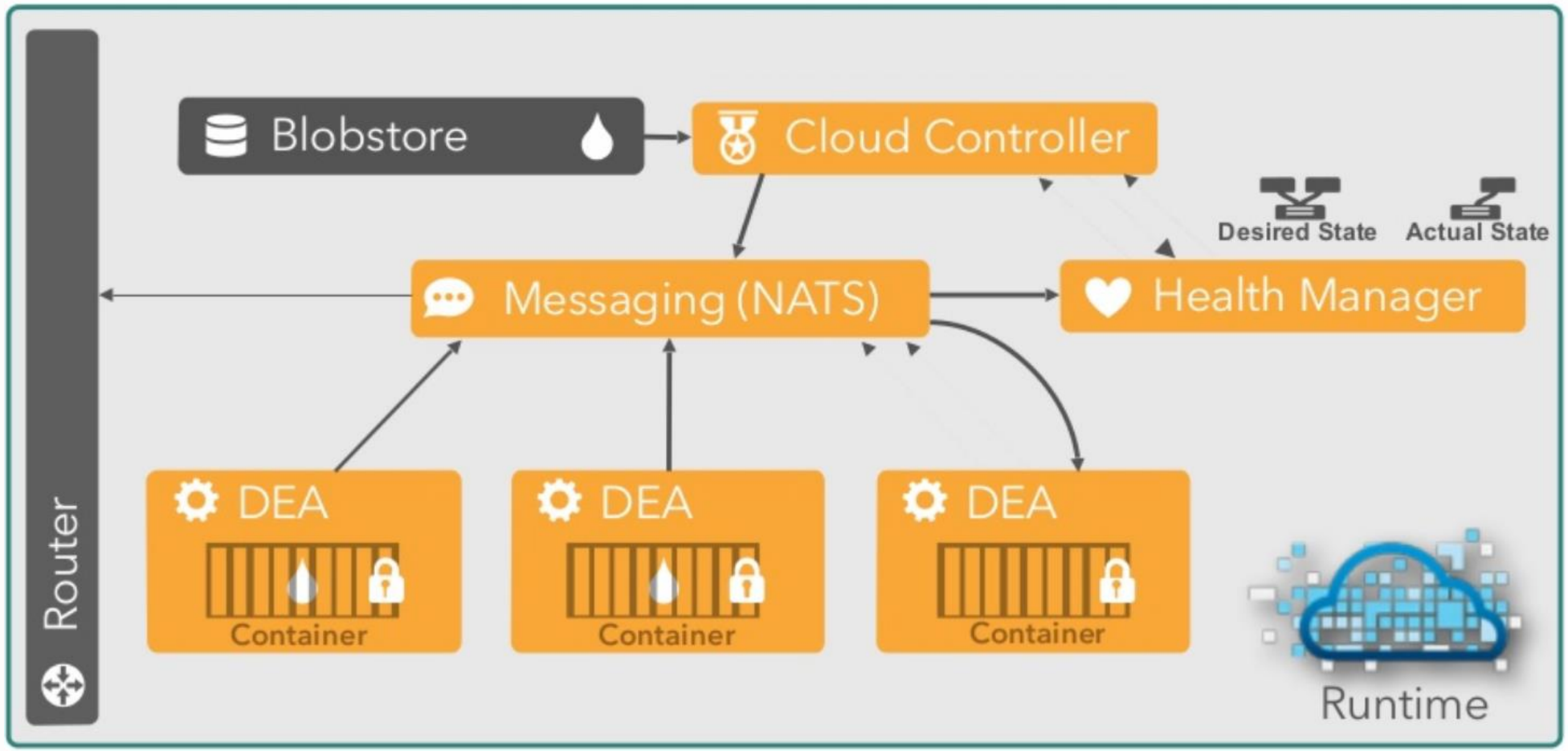
Cluster Management



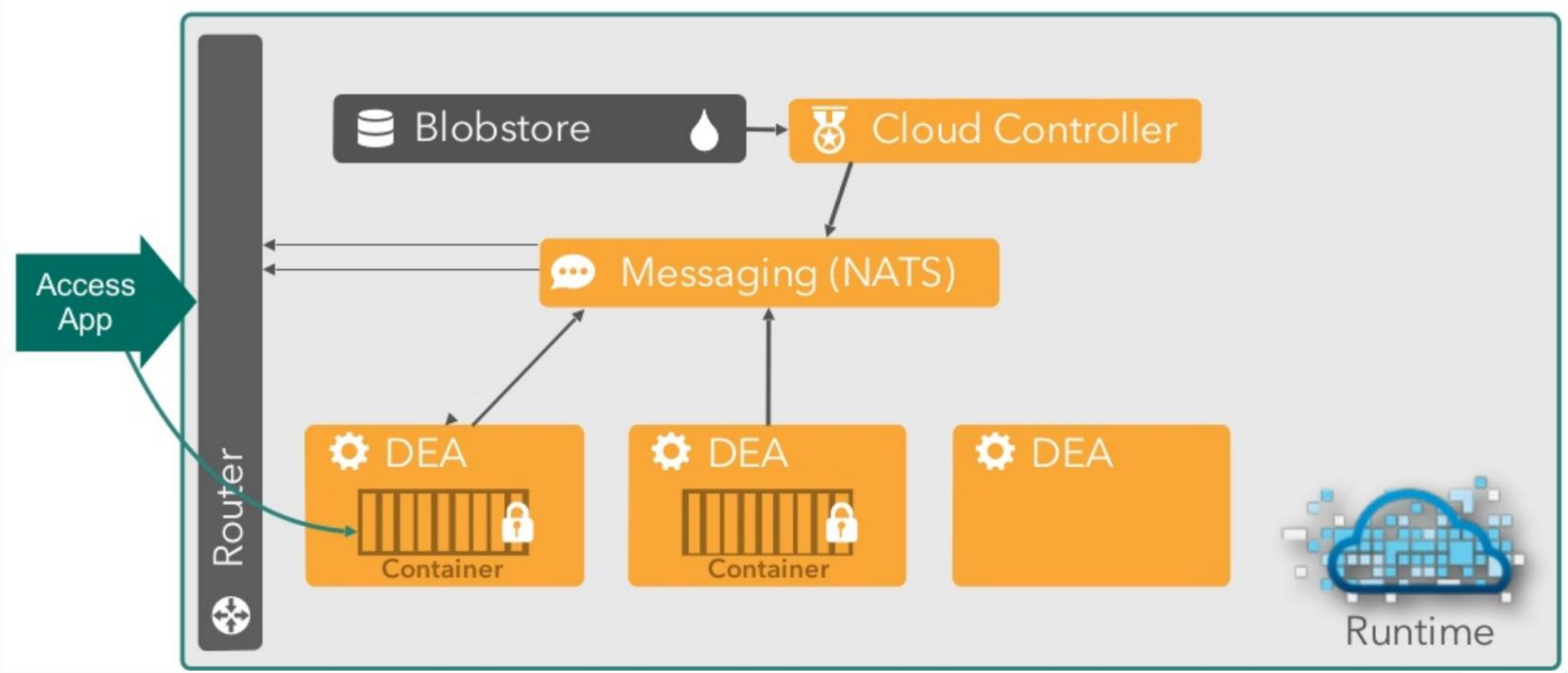
Cluster Management



Failover


Blobstore
Cloud Controller
Desired State
Actual State
Messaging (NATS)
Health Manager
Router
DEA
Container
DEA
Container
DEA
Container
Runtime

Load Balancing



GRACIAS