

Desining and implementing Microservices Tools



AGENDA

**7. Introduction
to APIs Lifecycle**

**8. API Gateway
implementation**

9. APIs design

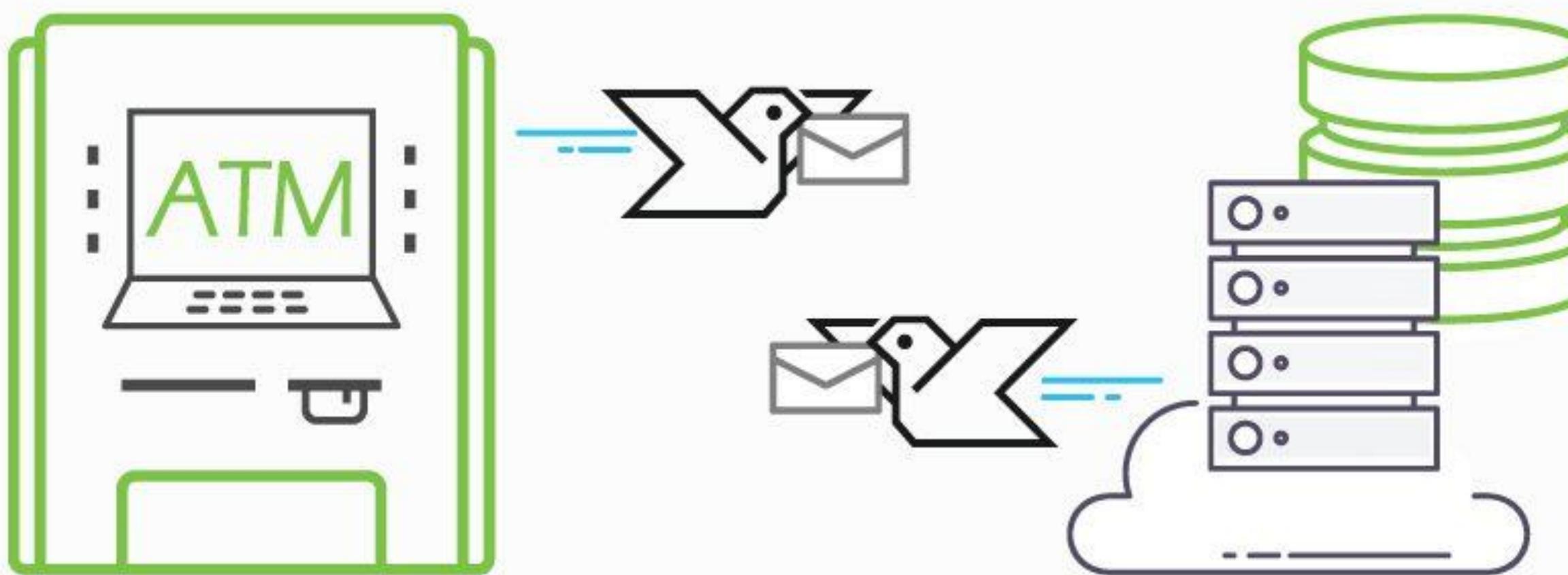
**10. Introduction
to API Blueprint
and Swagger**

**11. Deployment
of APIs on
microservices**

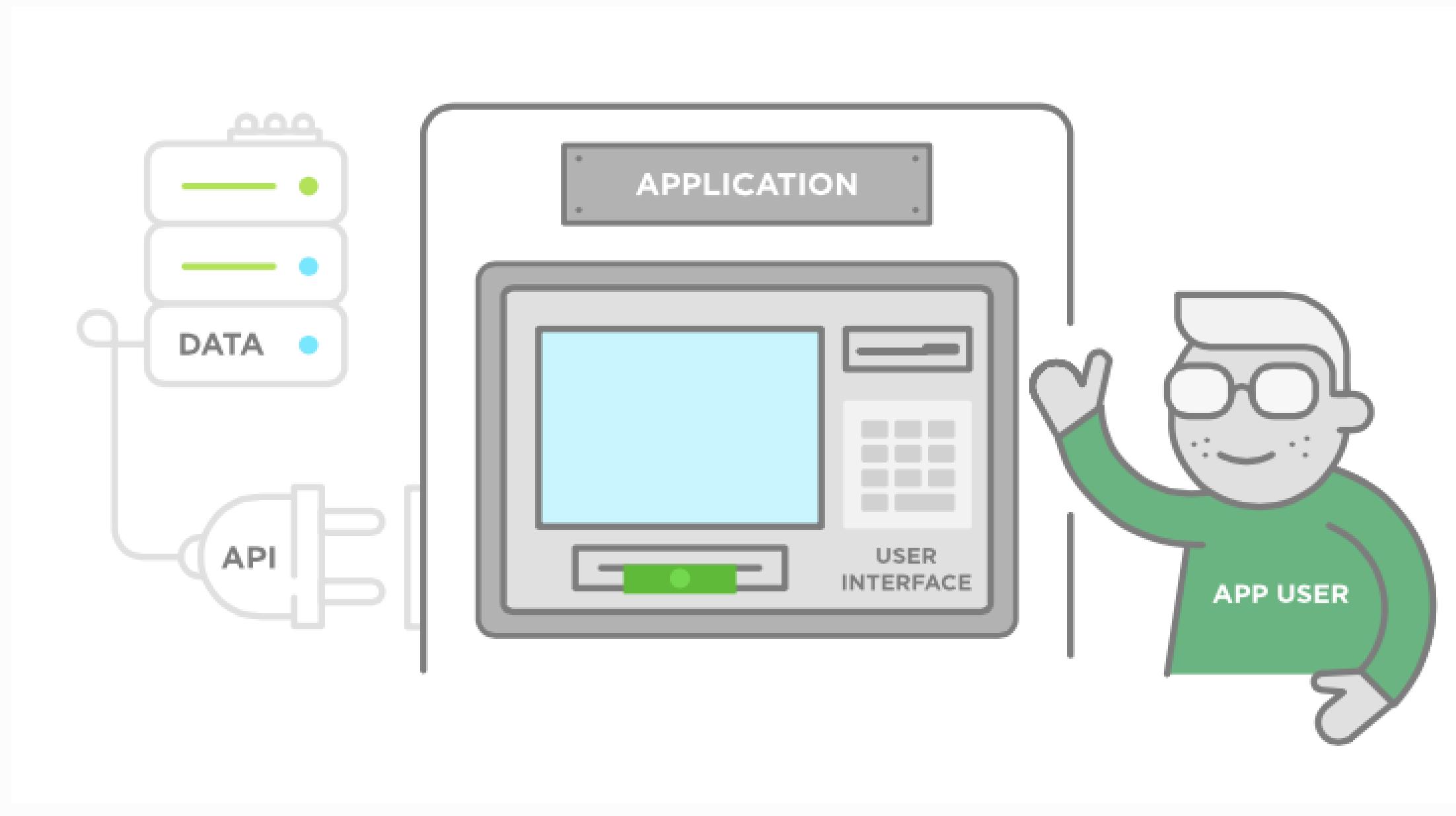
**12. Scalability
for your
microservices.**

7. Introduction to APIs Lifecycle

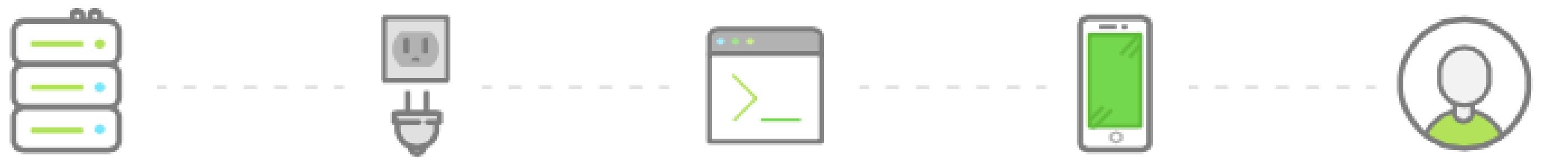
Messaging system



API flow



API Actors



ASSETS

Data, Software,
Services, Products

APIs

DEVELOPERS

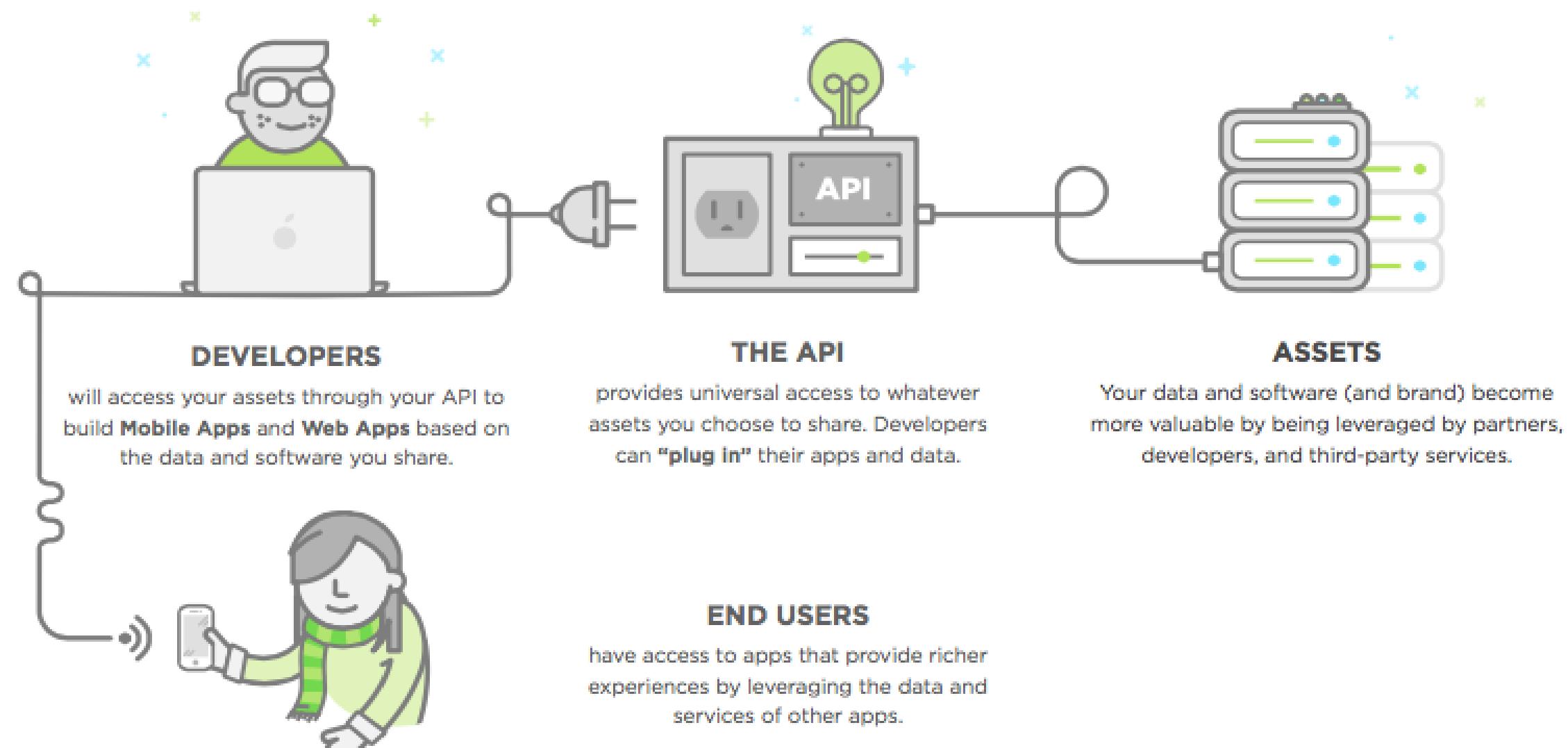
Use APIs
to create apps

SOFTWARE & APPLICATIONS

powered by APIs

END USERS

API Lifecycle



Types of API

The three basic types of APIs

APIs take three basic forms: local, web-like and program-like. Here's a look at each type.

Local APIs

The original API, created to provide operating system or middleware services to application programs.

Web APIs

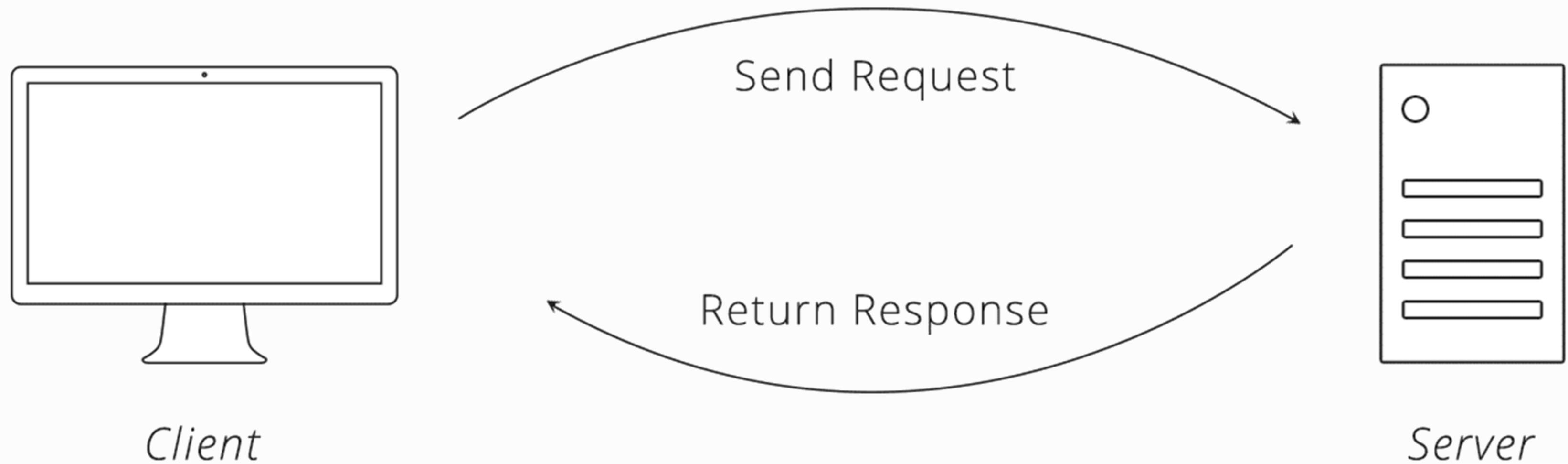
Designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Often called REST APIs or RESTful APIs.

Program APIs

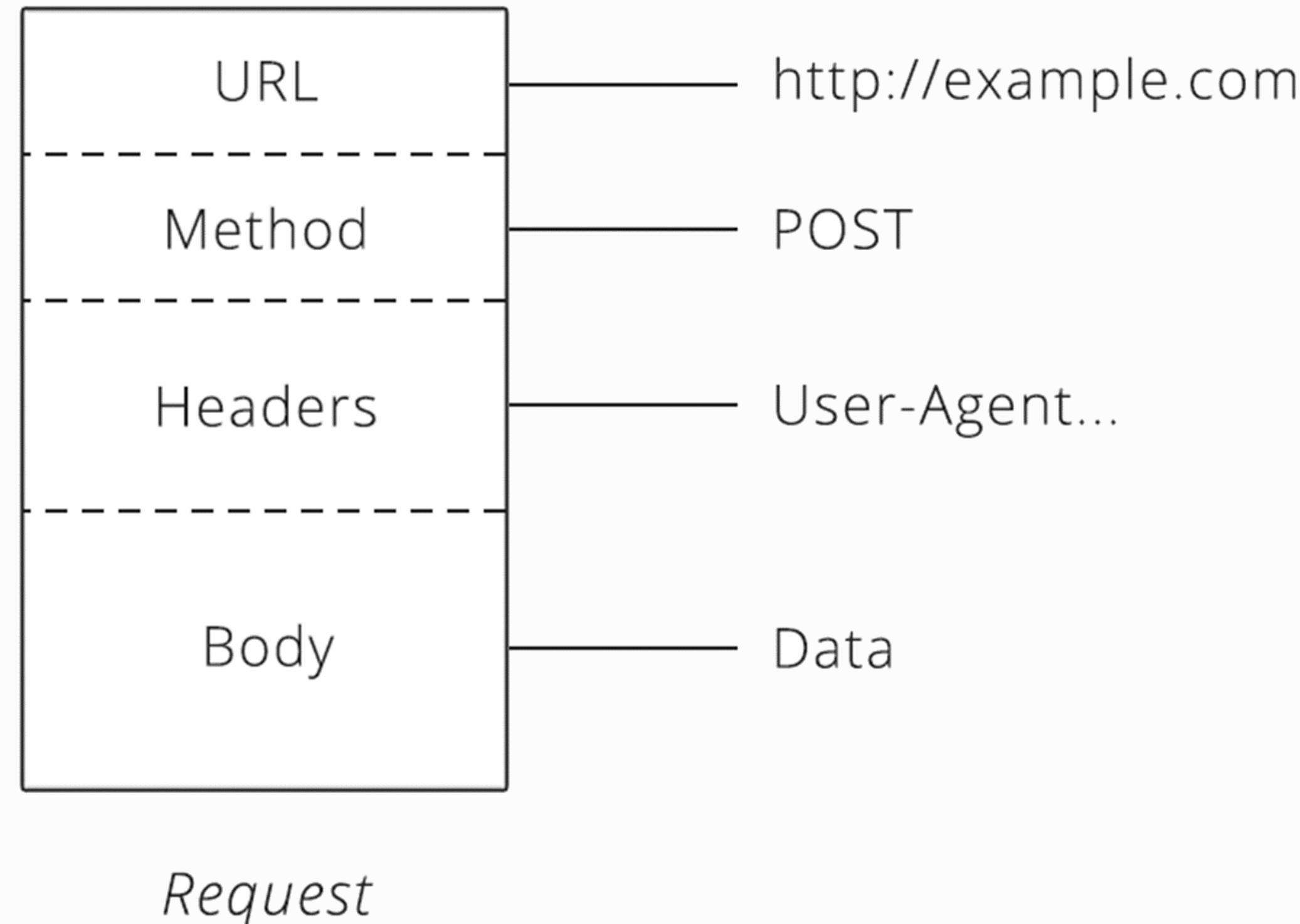
Based on RPC technology that makes a remote program component appear to be local to the rest of the software.

9. APIs design

Message Exchange Protocol



API Request

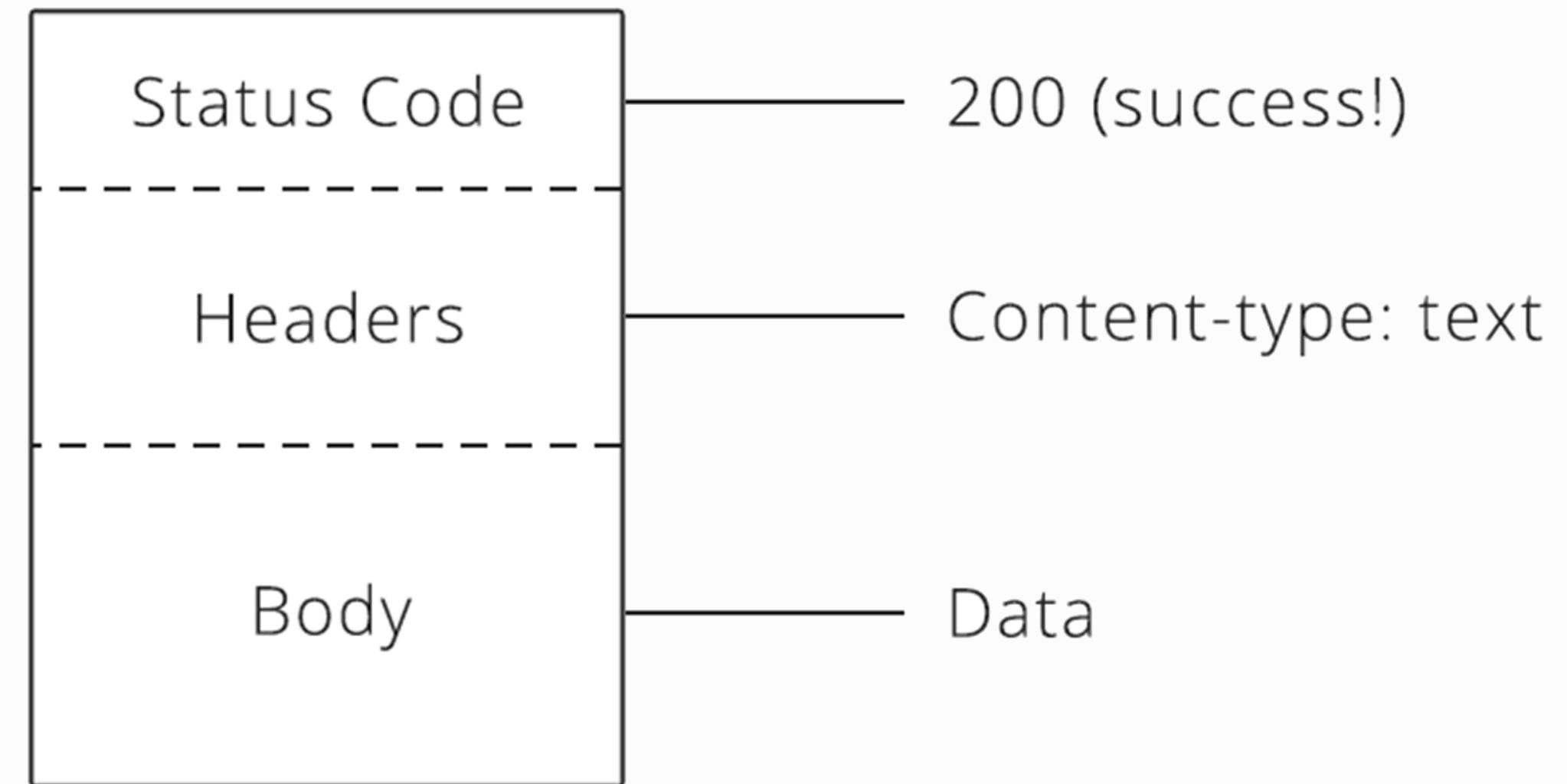


API Response

Not Found

The requested URL / was not found on this server.

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



Response

JSON

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

key



value



```
{ "crust": "original" }
```

XML

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

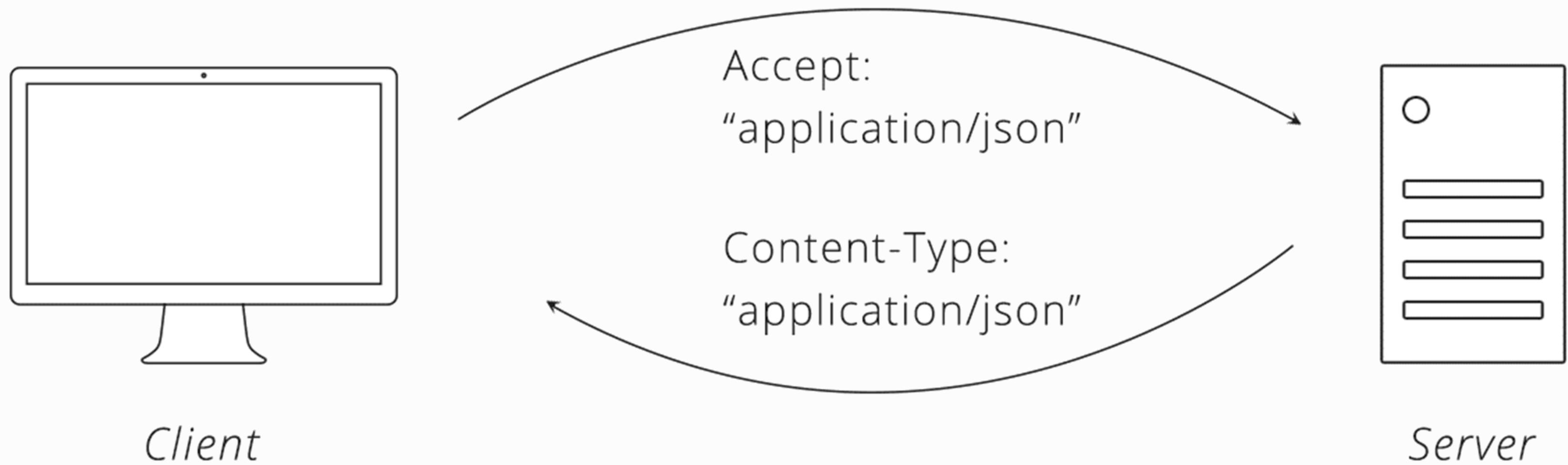
node opening tag

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

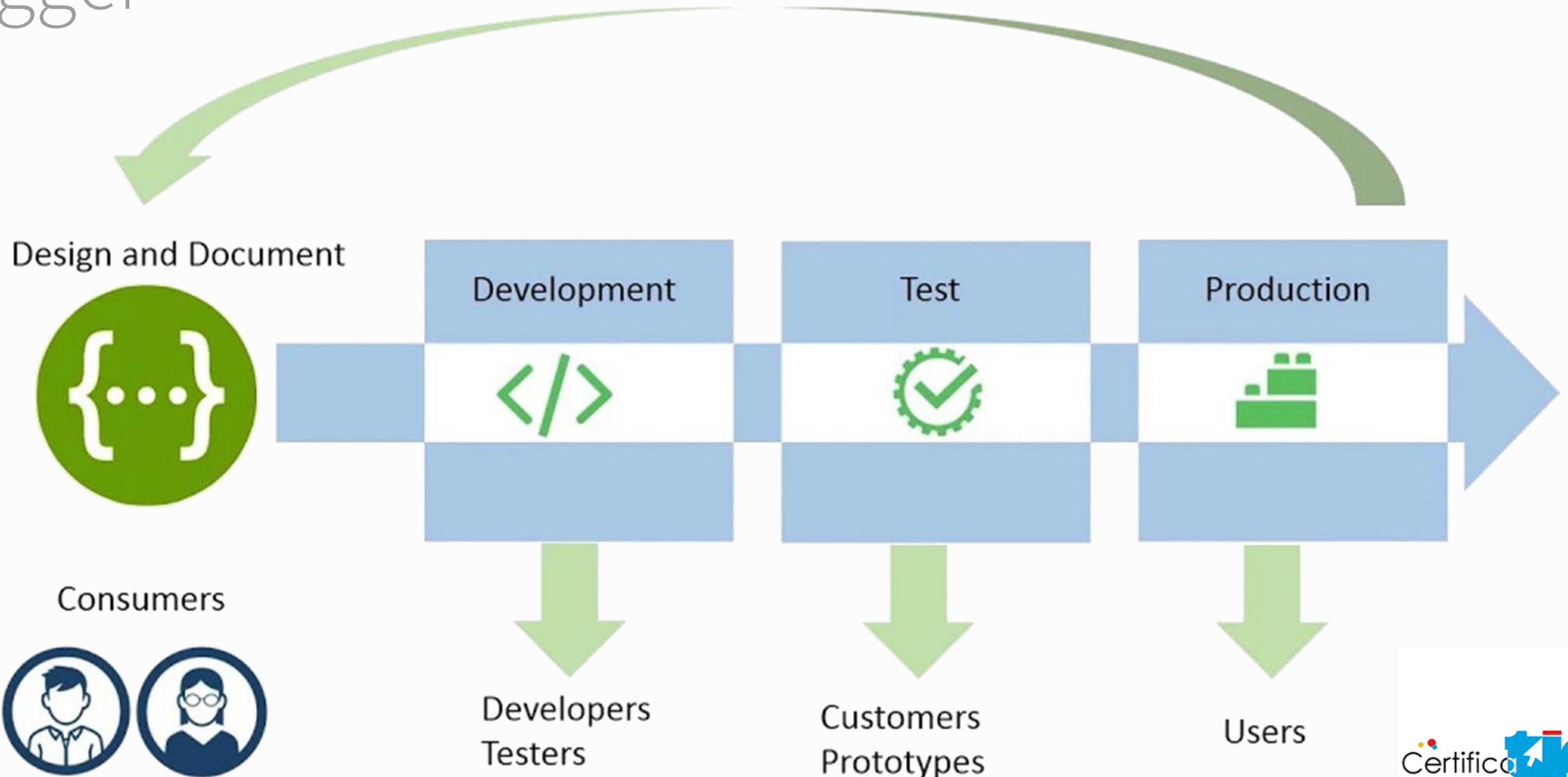
```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "name": "Shop Entry",
  "type": "object",
  "properties": {
    "id": {
      "type": "number"
    },
    "name": {
      "type": "string"
    },
    "price": {
      "type": "number"
    },
    "tags": {
      "type": "array",
      "items":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



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

```
1  swagger: "2.0"
2  info:
3    version: "1.0.0"                                     API Metadata
4    title: "Swagger Meetup API"
5    description: "An API for meetups"
6  basePath: "/api"
7  paths:
8    /meetups:
9      get:
10        tags:
11          - "Meetups"
12        operationId: "getMeetups"
13        parameters:
14          - name: "title"
15            in: "query"
16            type: "string"
17            required: false
18        responses:
19          200:
```

Host Metadata

```
1  swagger: "2.0"
2  info:
3    version: "1.0.0"
4    title: "Swagger Meetup API"
5    description: "An API for meetups"
6    basePath: "/api"                                Host Metadata
7  paths:
8    /meetups:
9      get:
10        tags:
11          - "Meetups"
12        operationId: "getMeetups"
13        parameters:
14          - name: "title"
15            in: "query"
16            type: "string"
17            required: false
18        responses:
19          200:
```

Resources

```
1  swagger: "2.0"
2  info:
3    version: "1.0.0"
4    title: "Swagger Meetup API"
5    description: "An API for meetups"
6    basePath: "/api"
7  paths:
8    /meetups:
9      get:
10        tags:
11          - "Meetups"
12        operationId: "getMeetups"
13        parameters:
14          - name: "title"
15            in: "query"
16            type: "string"
17            required: false
18        responses:
19          200:
```

Resources

Organizational Tags

```
1  swagger: "2.0"
2  info:
3    version: "1.0.0"
4    title: "Swagger Meetup API"
5    description: "An API for meetups"
6  basePath: "/api"
7  paths:
8    /meetups:
9      get:
10        tags:
11          - "Meetups"                                     Organizational Tags
12        operationId: "getMeetups"
13        parameters:
14          - name: "title"
15            in: "query"
16            type: "string"
17            required: false
18        responses:
19          200:
```

Codegen Hints

```
1  swagger: "2.0"
2  info:
3    version: "1.0.0"
4    title: "Swagger Meetup API"
5    description: "An API for meetups"
6  basePath: "/api"
7  paths:
8    /meetups:
9      get:
10        tags:
11          - "Meetups"
12        operationId: "getMeetups"          Codegen Hints
13        parameters:
14          - name: "title"
15            in: "query"
16            type: "string"
17            required: false
18        responses:
19          200:
```

Parameters

```
1  swagger: "2.0"
2  info:
3    version: "1.0.0"
4    title: "Swagger Meetup API"
5    description: "An API for meetups"
6    basePath: "/api"
7  paths:
8    /meetups:
9      get:
10        tags:
11          - "Meetups"
12        operationId: "getMeetups"
13        parameters: Parameters
14          - name: "title"
15            in: "query"
16            type: "string"
17            required: false
18        responses:
19          200:
```

Expected Responses

```
18 responses:          Expected Responses
19   200:
20     description: "success"
21     schema:
22       type: "array"
23     items:
24       $ref: "#/definitions/Meetup"
25   post: ➔
26   /meetups/{meetupId}: ➔
27 definitions:
28   Meetup:
29     required:
30       - "id"
31       - "name"
32     properties:
33       id:
34         type: "string"
35         description: "The unique slug of the meetup"
36         readOnly: true
```

Response Payload

```
18      responses:
19        200:
20          description: "success"
21        schema:                                     Response Payload
22          type: "array"
23          items:
24            $ref: "#/definitions/Meetup"
25      post: ➔
26      /meetups/{meetupId}: ➔
27    definitions:
28      Meetup:
29        required:
30          - "id"
31          - "name"
32        properties:
33          id:
34            type: "string"
35            description: "The unique slug of the meetup"
36            readOnly: true
37
```

Model Schema

```
18      responses:
19        200:
20          description: "success"
21          schema:
22            type: "array"
23            items:
24              $ref: "#/definitions/Meetup"
25        post: ➔
26        /meetups/{meetupId}: ➔
27      definitions:
28        Meetup:
29          required:
30            - "id"
31            - "name"
32          properties:
33            id:
34              type: "string"
35              description: "The unique slug of the meetup"
36              readOnly: true
```

Model Schemas

Required Properties

```
18      responses:
19        200:
20          description: "success"
21        schema:
22          type: "array"
23        items:
24          $ref: "#/definitions/Meetup"
25      post: ➔
59      /meetups/{meetupId}: ➔
74    definitions:
75      Meetup:
76        required: Required Properties
77        - "id"
78        - "name"
79      properties:
80        id:
81          type: "string"
82          description: "The unique slug of the meetup"
83          readOnly: true
```

Properties and datatypes

```
18    responses:
19      200:
20        description: "success"
21        schema:
22          type: "array"
23          items:
24            $ref: "#/definitions/Meetup"
25      post: ➔
26      /meetups/{meetupId}: ➔
27    definitions:
28      Meetup:
29        required:
30          - "id"
31          - "name"
32        properties: Properties and Datatypes
33          id:
34            type: "string"
35            description: "The unique slug of the meetup"
36            readOnly: true
```

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

```
32 @Path("/pet")
33 @Api(value = "/pet", description = "Operations about pets", authorizations = {
34     @Authorization(value = "petstore_auth",
35     scopes = {
36         @AuthorizationScope(scope = "write:pets", description = "modify pets in your account"),
37         @AuthorizationScope(scope = "read:pets", description = "read your pets")
38     })
39 }, tags = "pet")
40 @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
41 public class PetResource {
42     static PetData petData = new PetData();
43
44     @GET
45     @Path("/{petId}")
46     @ApiOperation(value = "Find pet by ID",
47         notes = "Returns a pet when ID <= 10. ID > 10 or nonintegers will simulate API error conditions",
48         response = Pet.class,
49         authorizations = @Authorization(value = "api_key")
50     )
51     @ApiResponses(value = { @ApiResponse(code = 400, message = "Invalid ID supplied"),
52         @ApiResponse(code = 404, message = "Pet not found") })
53     public Response getPetById(
54         @ApiParam(value = "ID of pet that needs to be fetched", allowableValues = "range[1,10]", required =
55         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

The screenshot shows the Swagger Editor interface with a yellow arrow pointing from the top section to the bottom section.

Top Section (Swagger Editor):

```
swagger: '2.0'
info:
  version: 1.0.0
  title: Swagger Petstore
  description: A sample example to demonstrate features in the swagger-2.0 specification
  contact:
    name: Swagger API team
    url: 'http://swagger.io'
  license:
    name: MIT
    url: 'http://opensource.org/licenses/MIT'
host: petstore.swagger.io
basePath: /api
schemes:
  - http
consumes:
  - application/json
```

Middle Section (Swagger Petstore Simple):

A sample API that uses a petstore as an example to demonstrate features in the swagger-2.0 specification

Version 1.0.0

Contact information
Swagger API team
<http://swagger.io>

License
MIT

Bottom Section (Generated Java Code):

```
import com.sun.jersey.core.header.FormDataContentDisposition;
import com.sun.jersey.multipart.FormDataParam;
import javax.ws.rs.core.Response;
import javax.ws.rs.*;
import io.swagger.annotations.Api;
import io.swagger.annotations.Generated(value = "class io.swagger.codegen.languages.JaxRSServerCodegen");
public class PetApi {
    private final PetApiService delegate = PetApiServiceFactory.getPetApi();
    @PUT
    @Consumes({ "application/json", "application/xml" })
    @Produces({ "application/json", "application/xml" })
    @io.swagger.annotations.ApiOperation(value = "Update an existing pet", note = "This can also be done without id in body by making pet.id null")
    @io.swagger.annotations.ApiResponses(value = {
        @io.swagger.annotations.ApiResponse(code = 405, message = "Validation error"),
        @io.swagger.annotations.ApiResponse(code = 404, message = "Pet not found"),
        @io.swagger.annotations.ApiResponse(code = 400, message = "Invalid ID specified")
    })
    public Response updatePet(@ApiParam(value = "Pet object that needs to be updated") Pet pet)
```

Design vs Code mismatch



```

32  @Path("/")
33  @io.swagger.annotations.Api(value = "/", tags = "Job"
34  public class JobApi {
35      private final JobApiService delegate = JobApiServ
36
37      @Context HttpServletRequest request;
38      @Context HttpServletResponse response;
39
40      @POST
41      @Path("/job")
42      @io.swagger.annotations.ApiOperation(value = "",
43      @io.swagger.annotations.ApiResponses(value = {
44          @io.swagger.annotations.ApiResponse(code = 20
45
46          @io.swagger.annotations.ApiResponse(code = 40
47      public Response addJob(@ApiParam(value = "Job req
48      throws NotFoundException {
49          final Timer.Context context = Service.metrics
50          final Timer.Context defaultTimer = Service.me
51
52          try {
53              return delegate.addJob(request, response,
54          } finally {
55              context.stop();
56              defaultTimer.stop();
57          }
58      }
59
60      @DELETE
61      @Path("/job/{ jobId}")
62      @io.swagger.annotations.ApiOperation(value = "",
63      @io.swagger.annotations.ApiResponses(value = {
64          @io.swagger.annotations.ApiResponse(code = 20
65      public Response deleteJob(@ApiParam(value = "desc
66      throws NotFoundException {
67          final Timer.Context context = Service.metrics
68          final Timer.Context defaultTimer = Service.me

```

```
{
    swagger: "2.0",
    info: {
        description: "This is a sample server Petstore API for testing 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/LICENSE-2.0"
        }
    },
    basePath: "/v2",
    tags: [
        {
            name: "pet",
            description: "Everything about your Pets"
        },
        {
            name: "store",
            description: "Operations about user"
        },
        {
            name: "user",
            description: "Access to Petstore orders"
        }
    ],
    externalDocs: {
        description: "Find out more",
        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



8. API Gateway implementation

Comparison

Monolith vs. Microservices



vs.



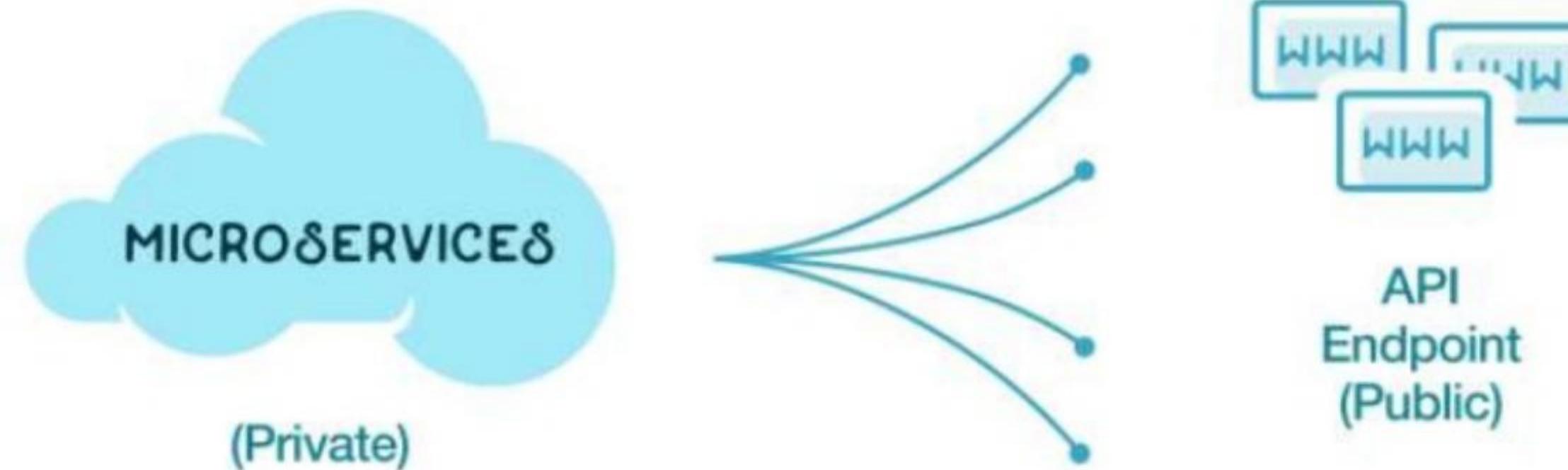
A monolithic app

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

A microservice app

Continuous updating in parts.

API Evolution



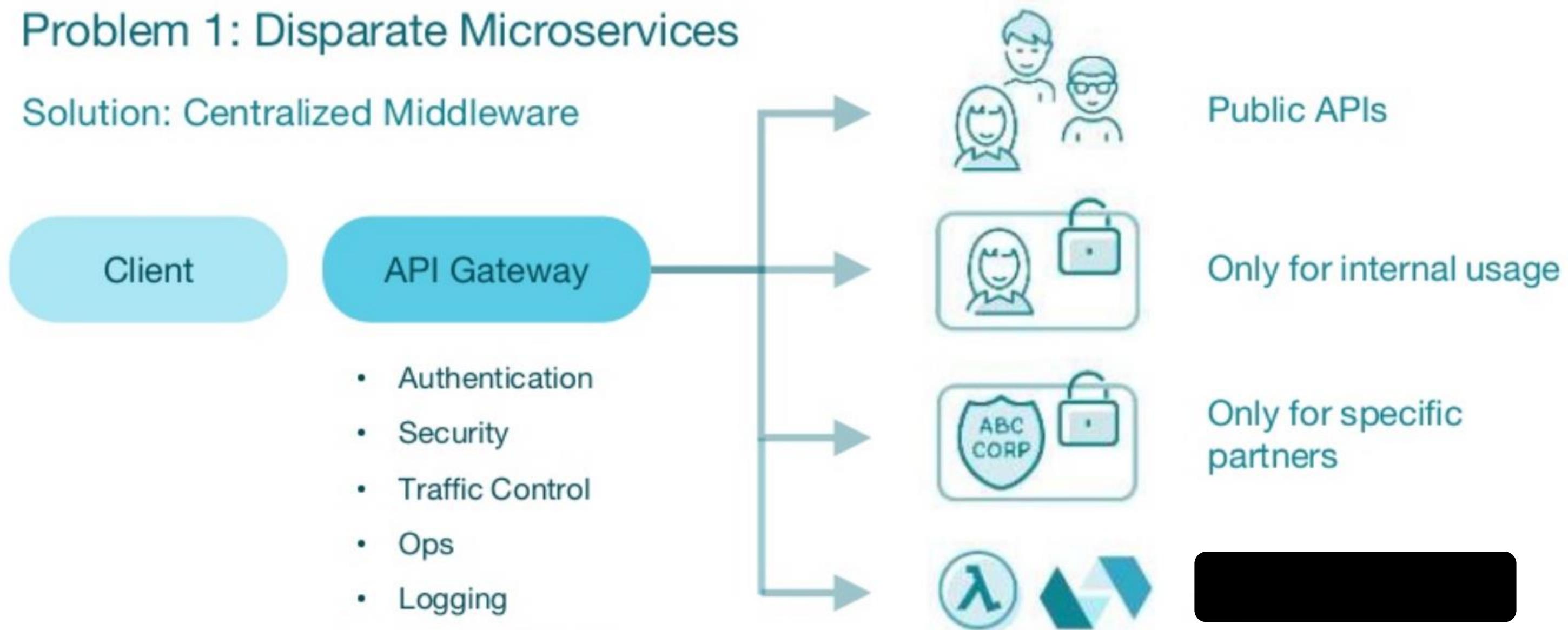
The natural evolution of microservices are APIs to the “N” client - client traditionally was built by the company, but now can be built by anyone

APIs = business leverage and amplification

Problems

Problem 1: Disparate Microservices

Solution: Centralized Middleware



Problems

Problem 2: Different Granularity Required

Solution: Orchestration



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

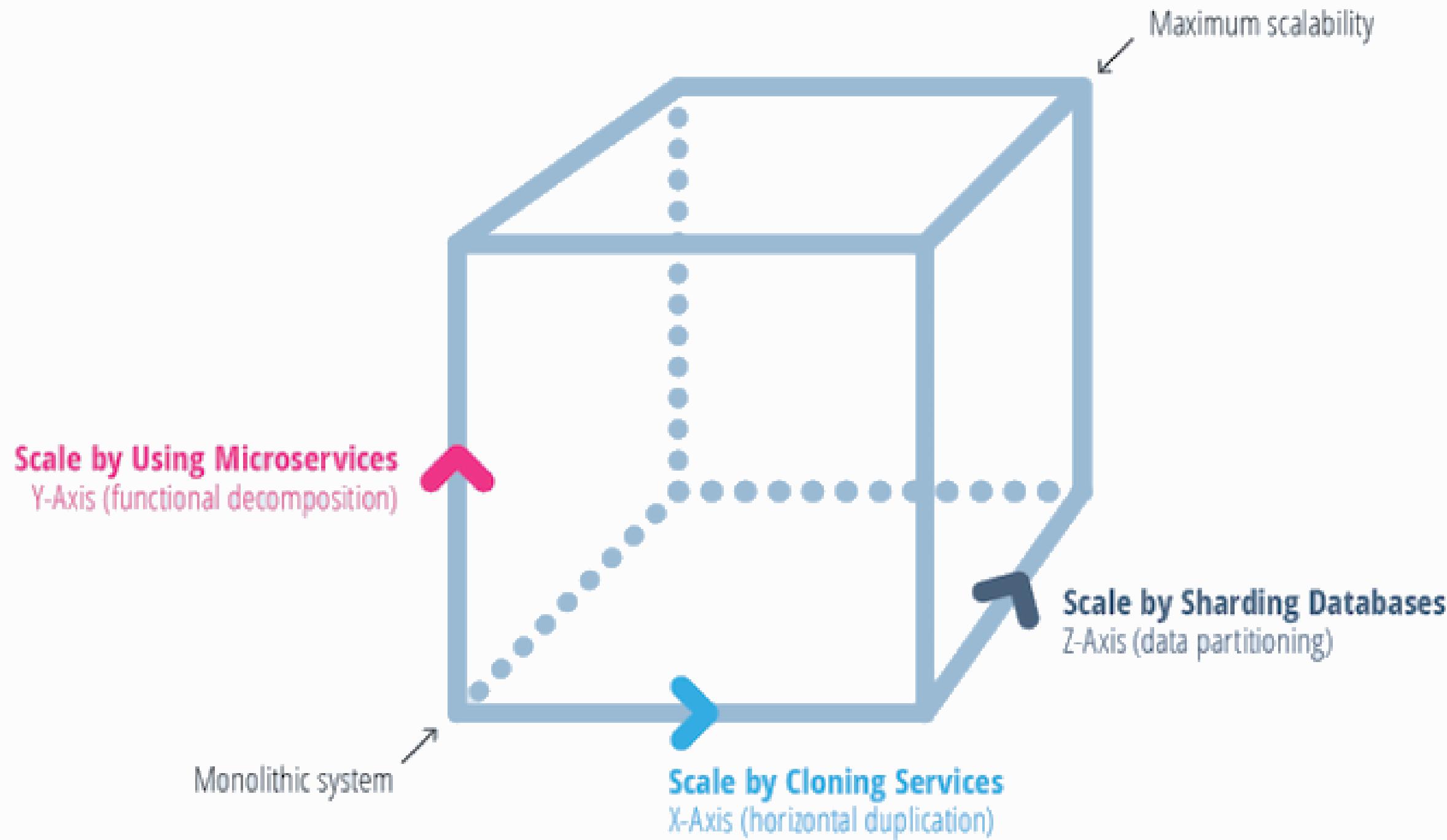
- Proxy with Load Balancer (coming soon)

- Data Limiter

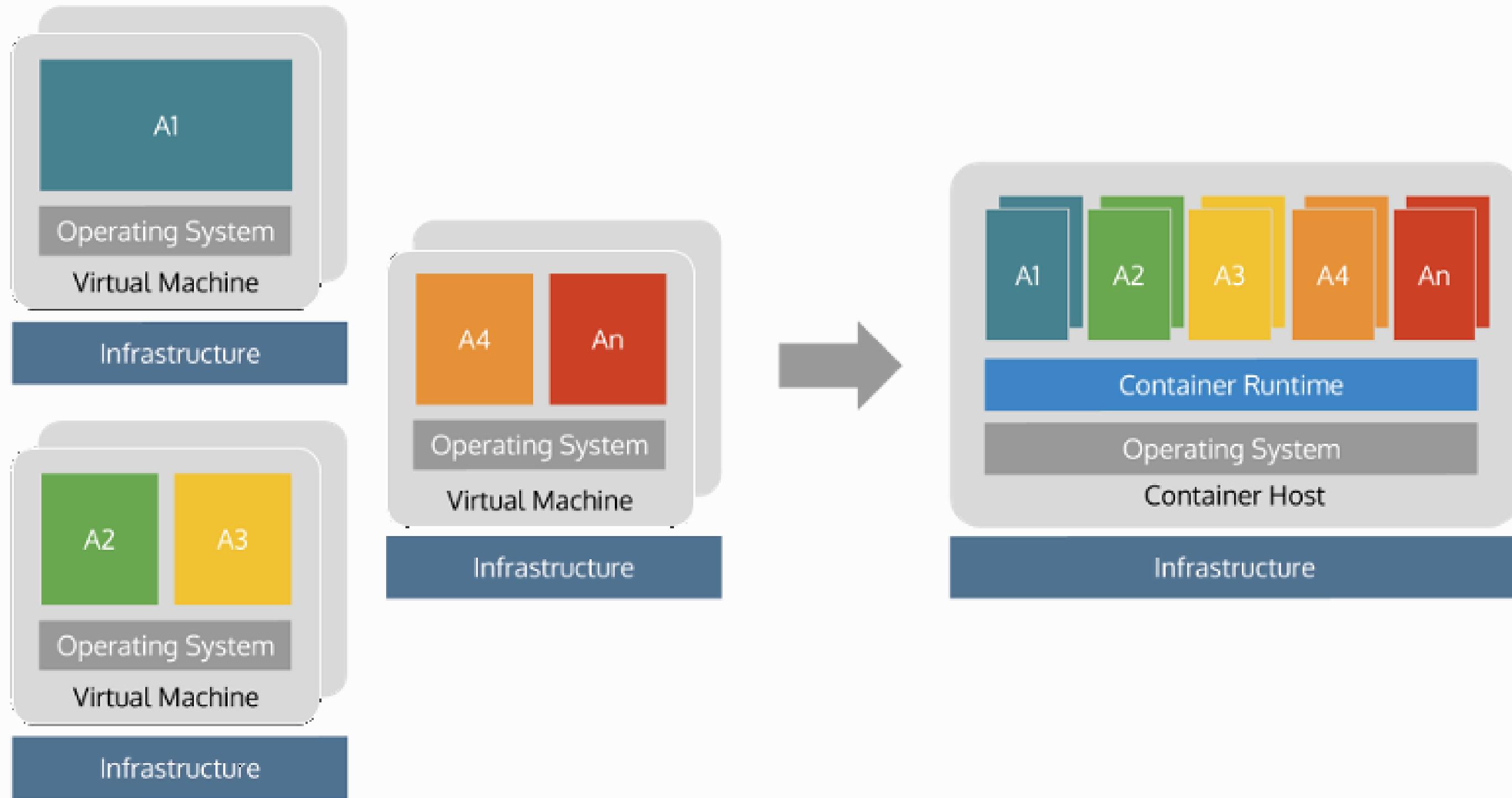
APIGEE

12. Scalability for your microservices.

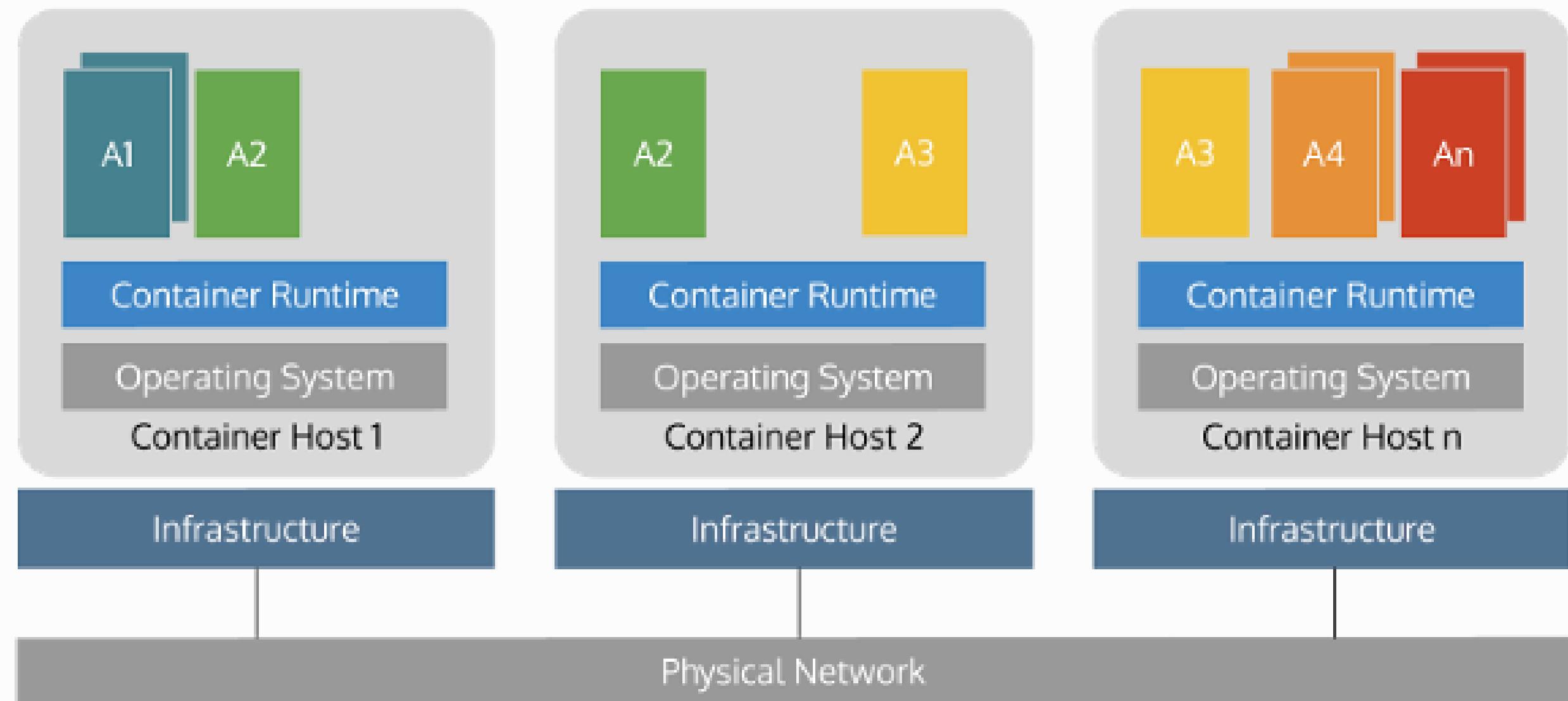
Scalability



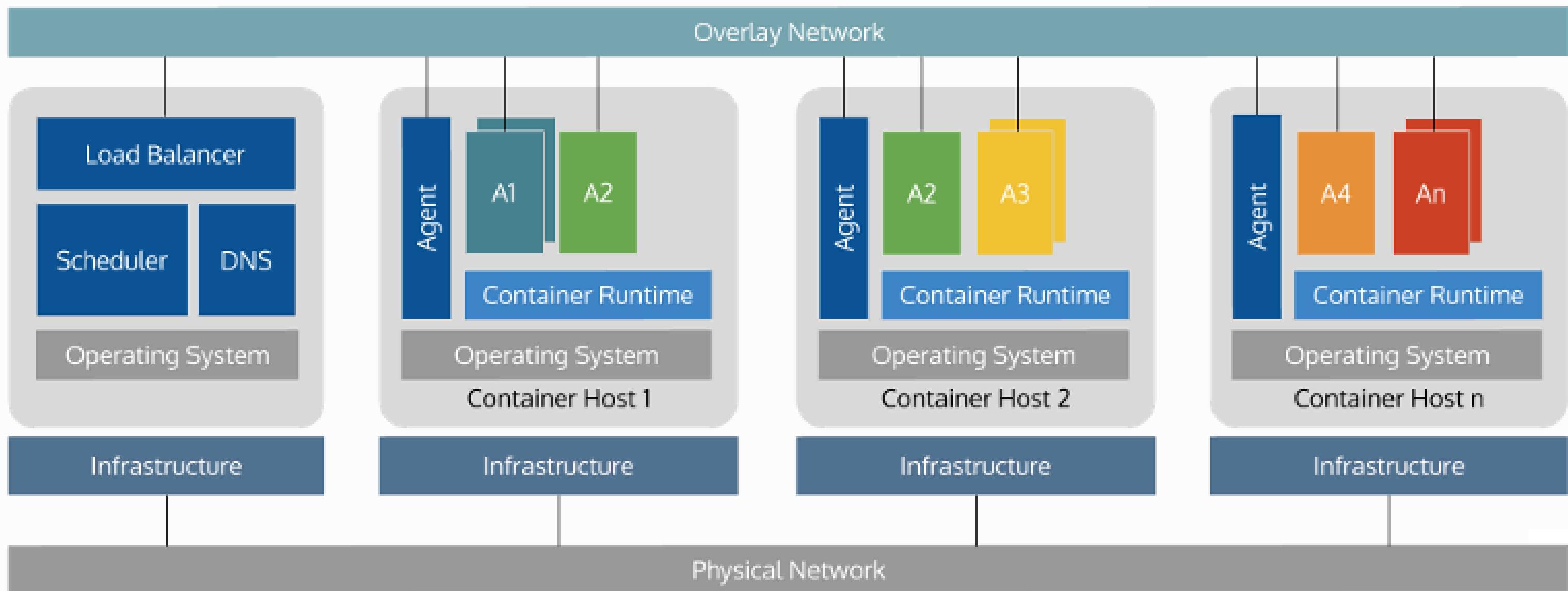
Cluster Management



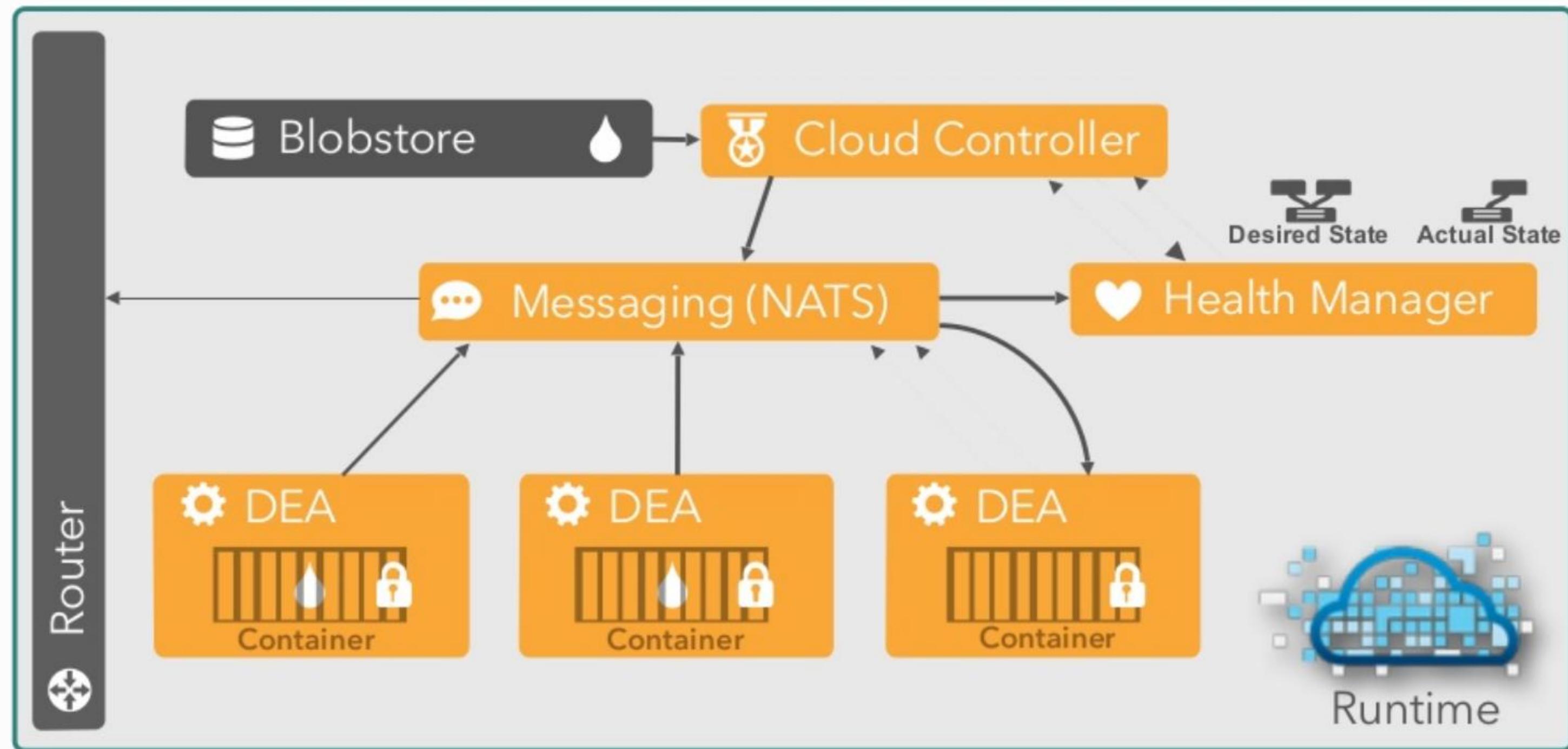
Cluster Management



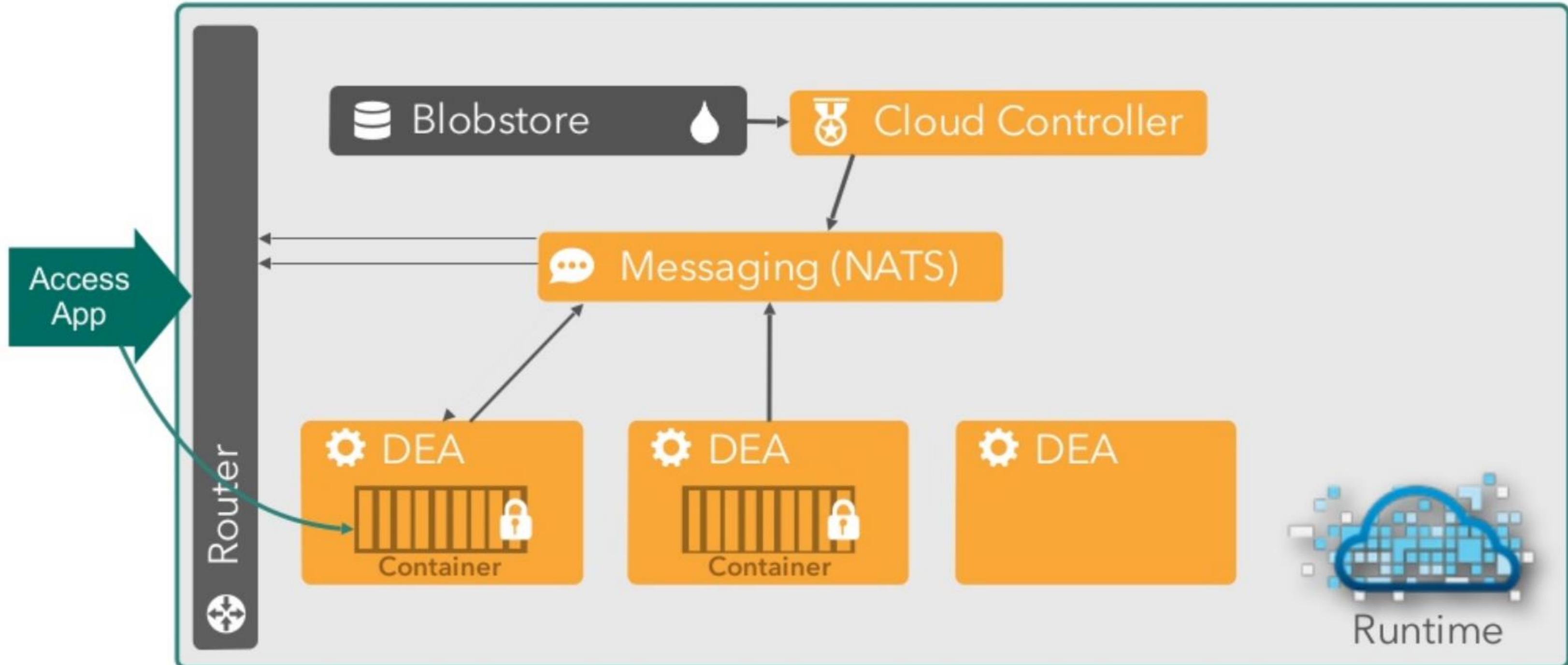
Cluster Management



Failover



Load Balancing



GRACIAS