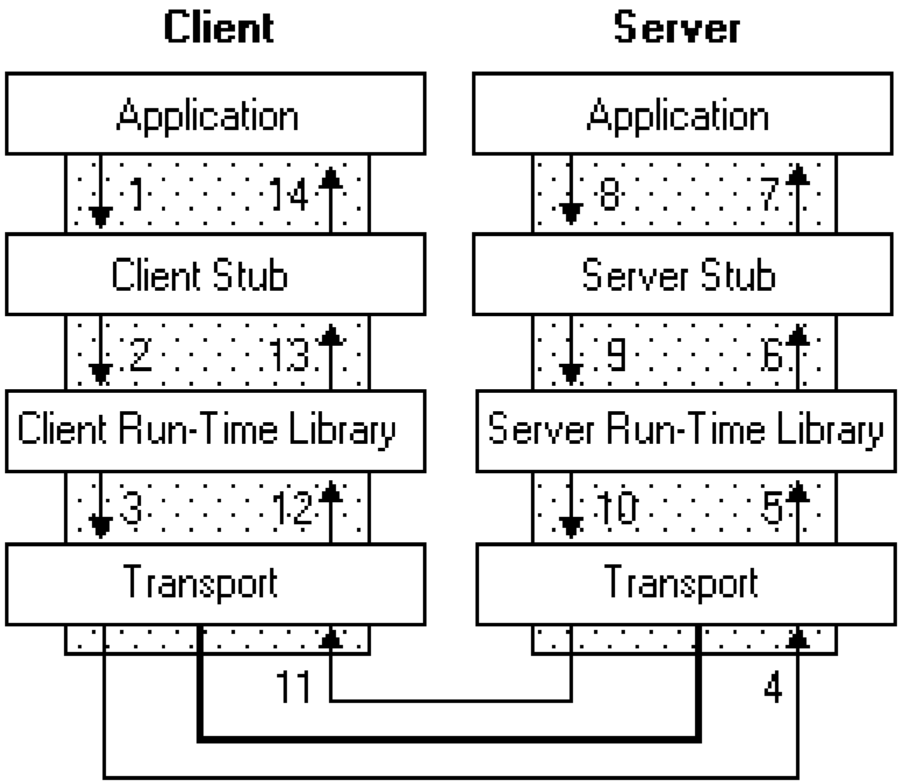
Designing scalable RESTful APIs

Cezar Socoteanu, Cloud Engineer @ CrowdStrike

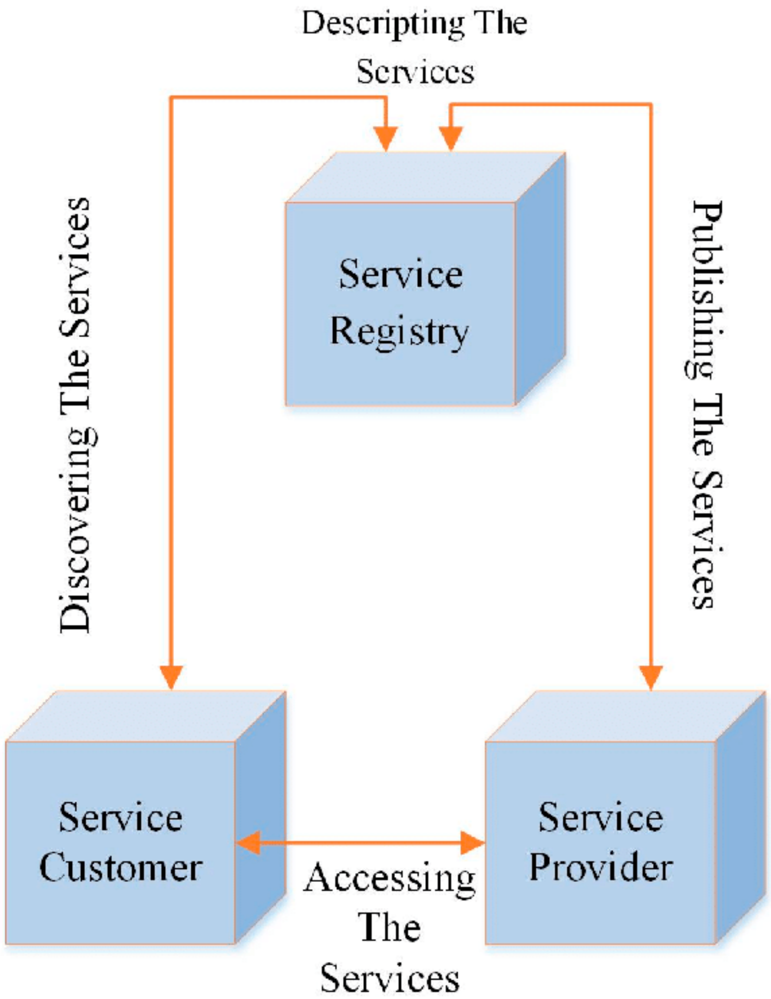
Agenda

- Introduction
- REST overview
- REST over HTTP
- Scaling up
- Conclusion
- QA

A long time ago …



RPC



SOA & SOAP

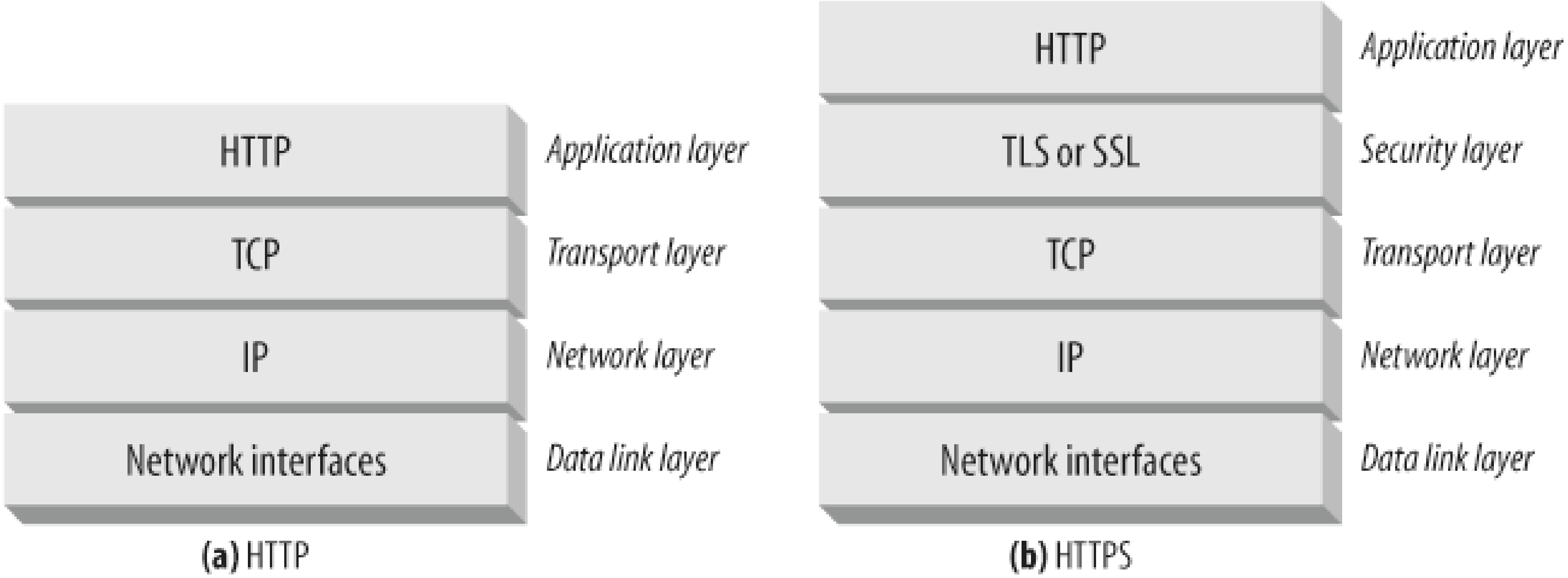
REST concepts

- Resource
- Server
- Client
- Request and Responses
- Representation

REST principles

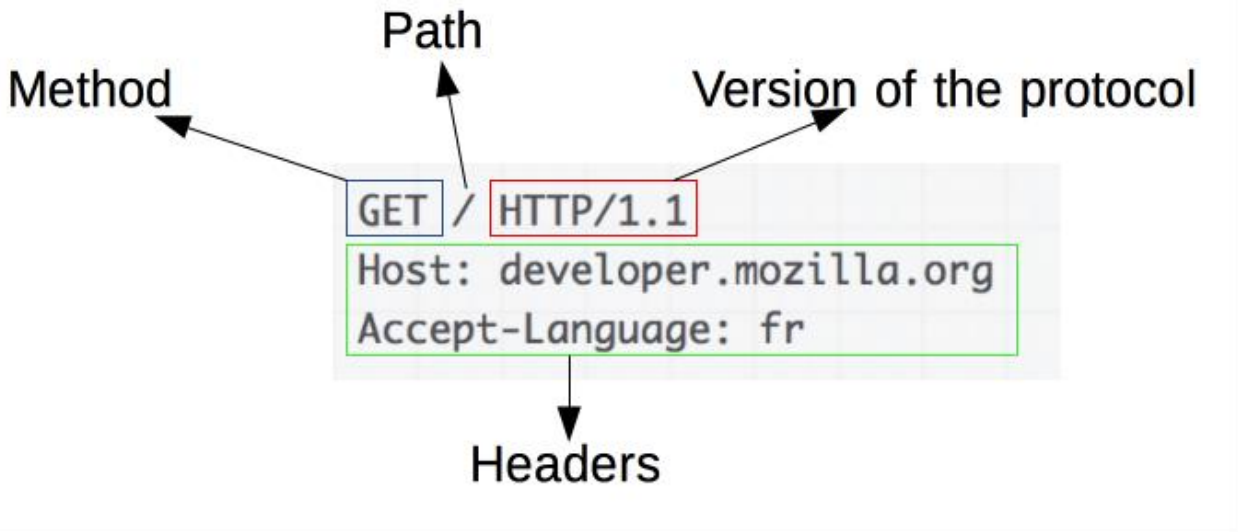
- Give every resource an ID
- Use standard methods
- Communicate statelessly
- Link things together
- Resources with multiple representations

REST over HTTP



HTTP Packet Format

- HTTP verbs: GET POST PUT PATCH DELETE
- URI
- Multiple HTTP headers
- HTTP Body



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP Common Headers

Request
Accept
Content-Type
If-Match
If-None-Match

Response
Content-Type
Content-Length
Status
ETag

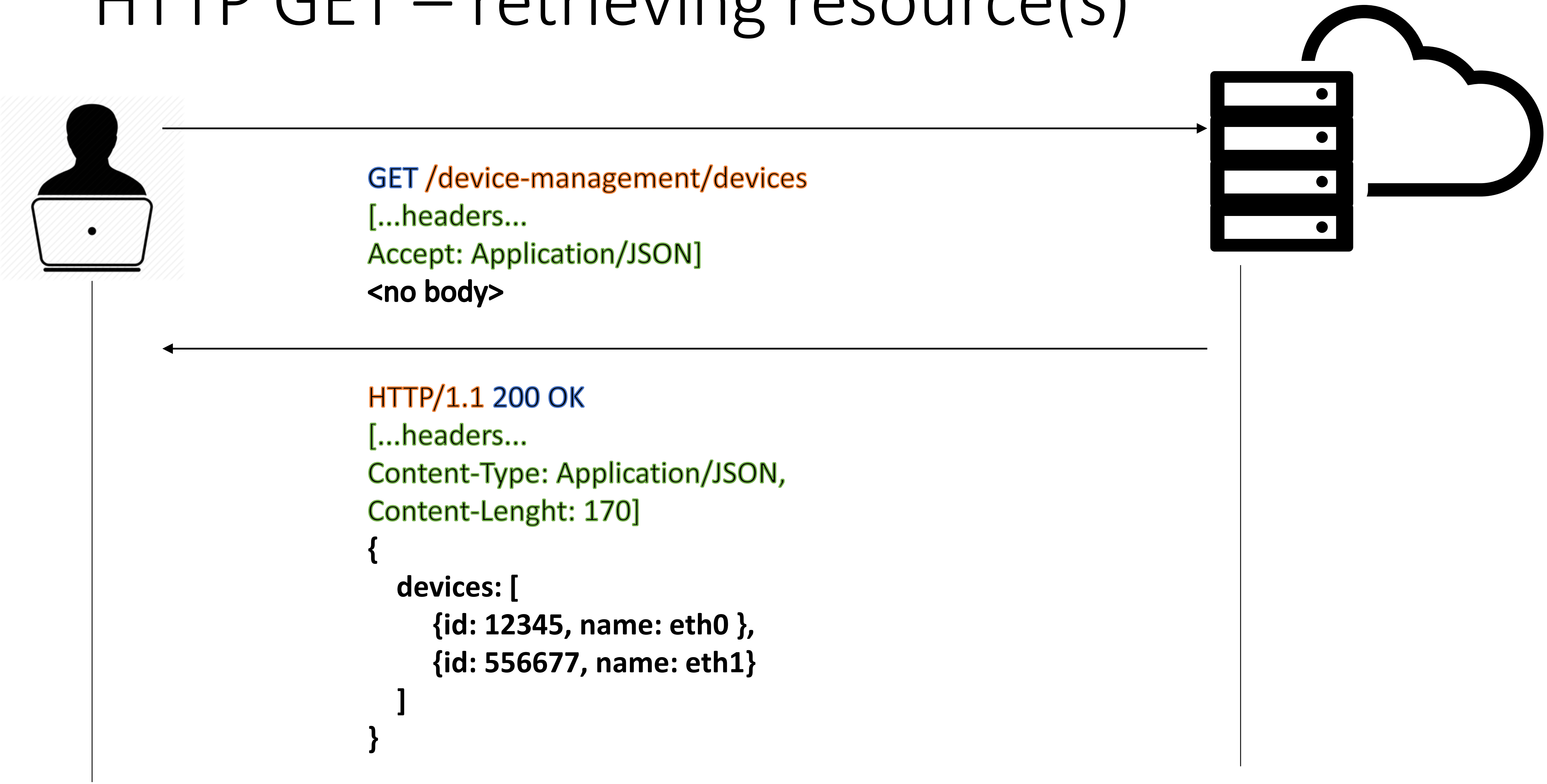
HTTP Common Status Codes

HTTP Code
200 OK
201 Created
202 Accepted
204 No Content
304 Not Modified
400 Bad Request
401 Unauthorized
403 Forbidden
404 Not Found
405 Not Allowed
415 Unsupported Media Type
500 Internal Server Error

Meet our API Endpoints

- /devices-management/devices
- /devices-management/devices/{id}
- /devices-management/devices/{id}/configurations
- /devices-management/devices/{id}/configurations/{id}
- /user-management/users/{id}
- /user-management/users/{id}/roles
- /user-management/users/{id}/roles/{id}

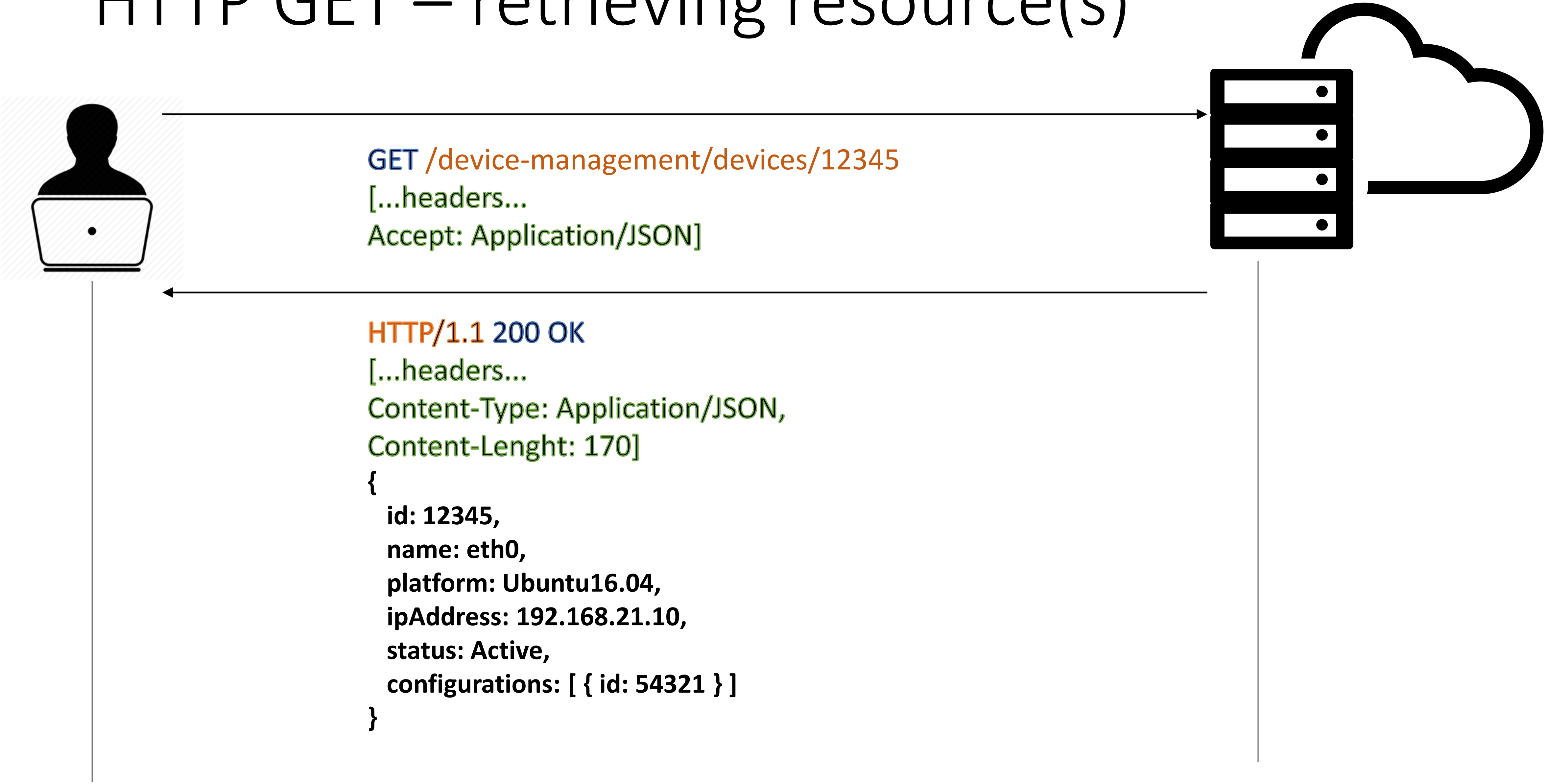
HTTP GET – retrieving resource(s)



```
GET /device-management/devices
[...headers...
Accept: Application/JSON]
<no body>
```

```
HTTP/1.1 200 OK
[...headers...
Content-Type: Application/JSON,
Content-Lenght: 170]
{
   devices: [
       {id: 12345, name: eth0 },
       {id: 556677, name: eth1}
   ]
}
```

HTTP GET – retrieving resource(s)

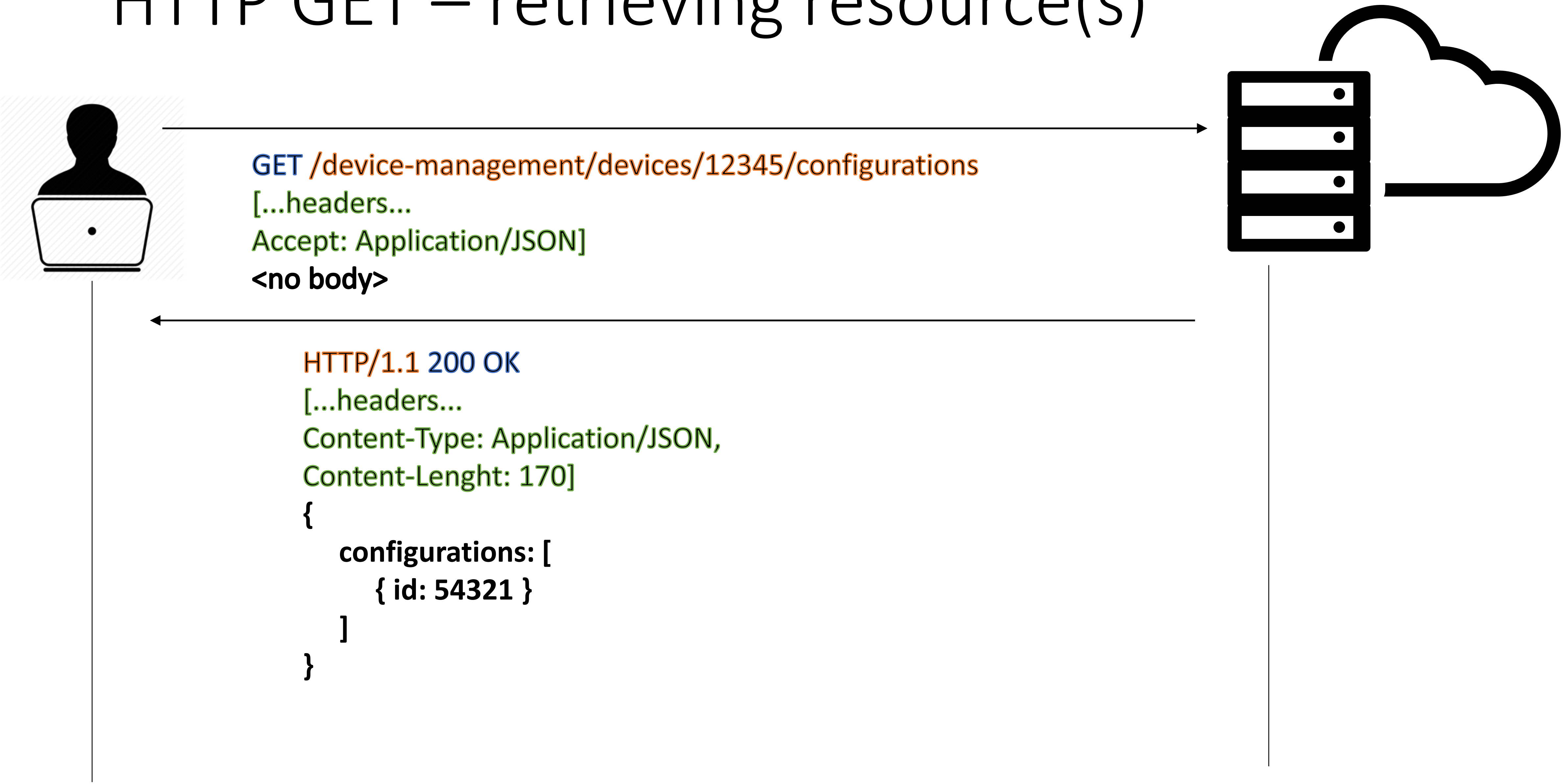


GET /device-management/devices/12345
[...headers...
Accept: Application/JSON]

HTTP/1.1 200 OK
[...headers...
Content-Type: Application/JSON,
Content-Lenght: 170]

```
{
  id: 12345,
  name: eth0,
  platform: Ubuntu16.04,
  ipAddress: 192.168.21.10,
  status: Active,
  configurations: [ { id: 54321 } ]
}
```

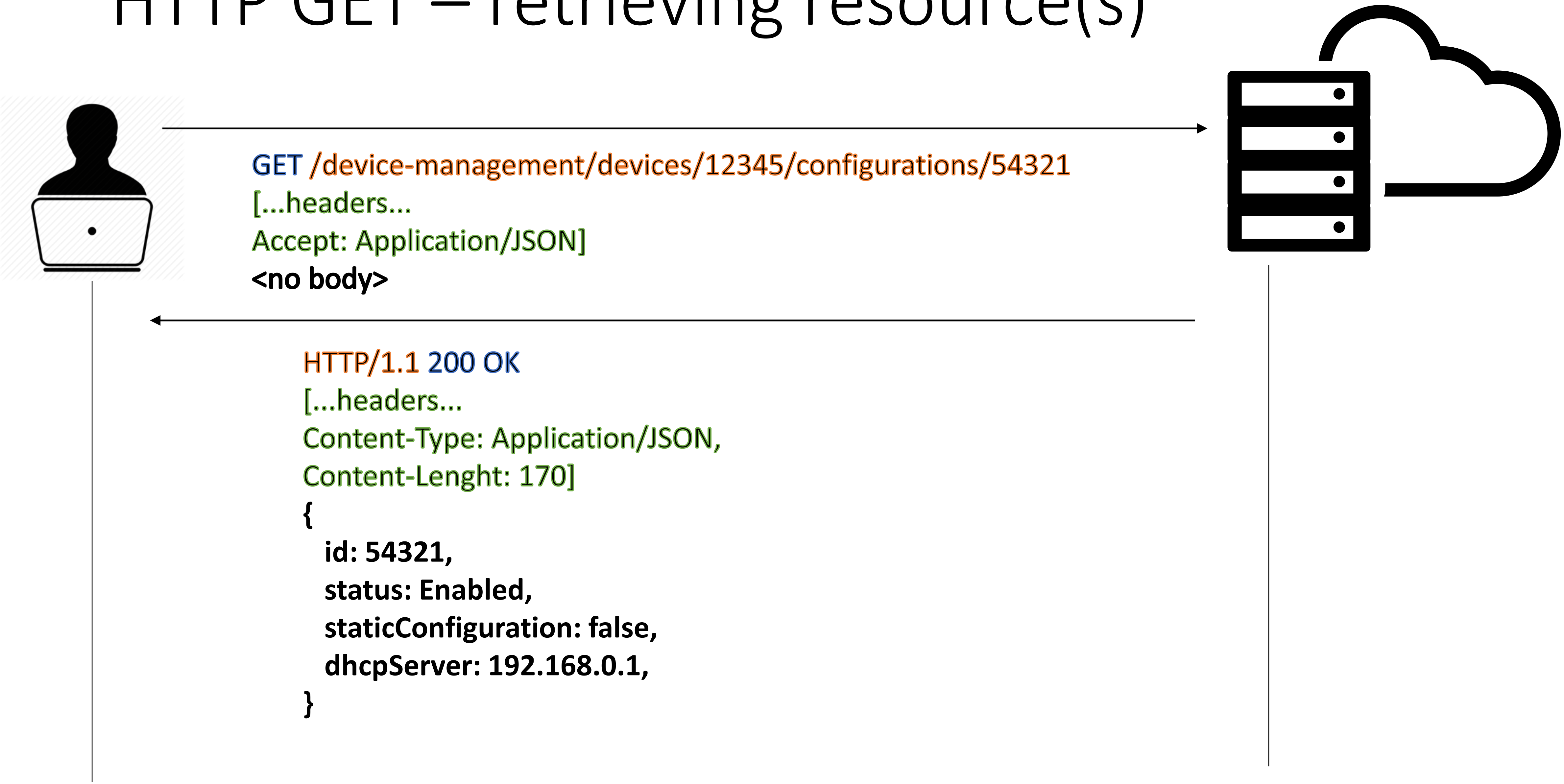
HTTP GET – retrieving resource(s)



```
GET /device-management/devices/12345/configurations
[...headers...
Accept: Application/JSON]
<no body>
```

```
HTTP/1.1 200 OK
[...headers...
Content-Type: Application/JSON,
Content-Lenght: 170]
{
    configurations: [
        { id: 54321 }
    ]
}
```

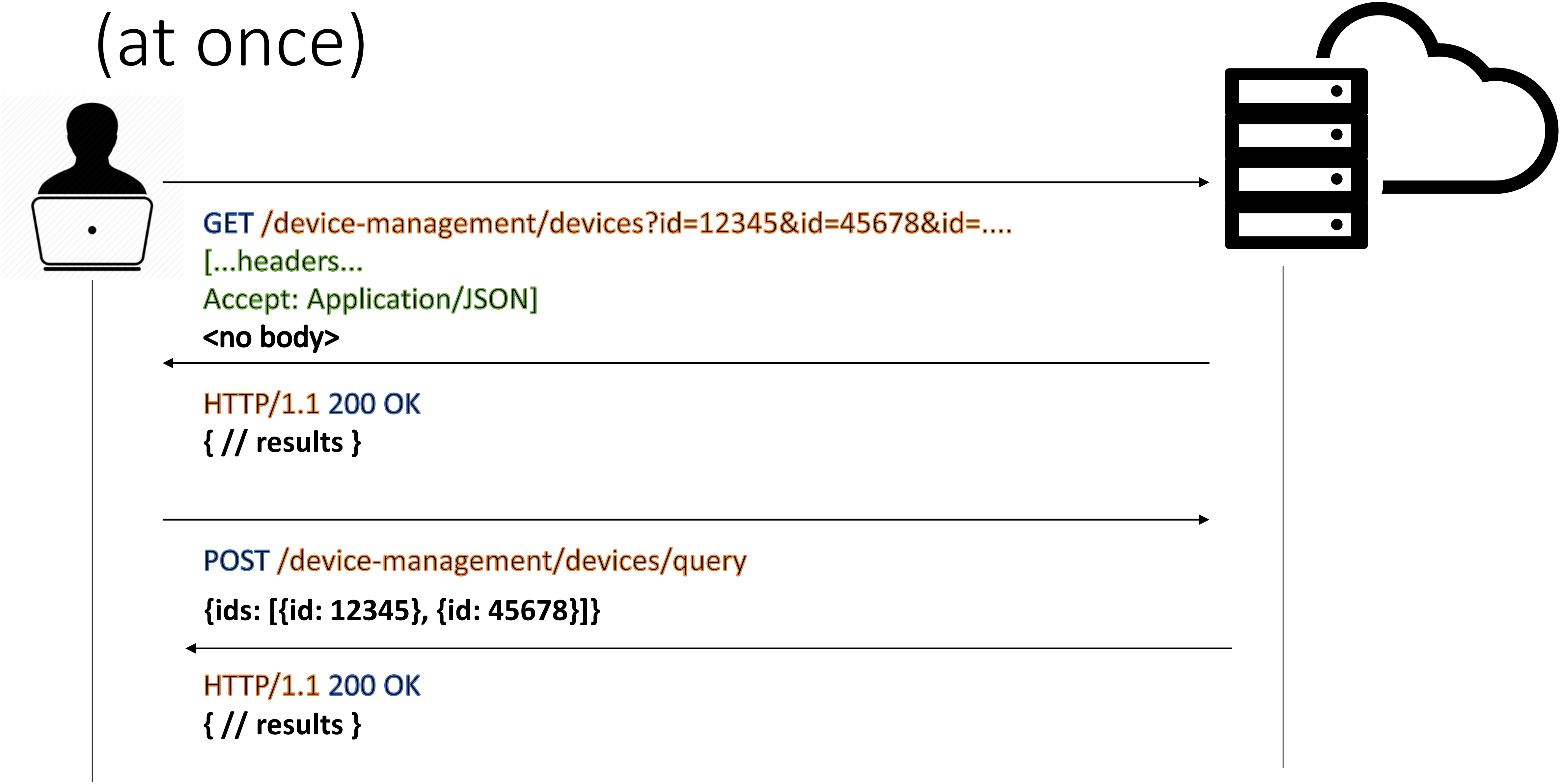
HTTP GET – retrieving resource(s)



```
GET /device-management/devices/12345/configurations/54321
[...headers...
Accept: Application/JSON]
<no body>
```

```
HTTP/1.1 200 OK
[...headers...
Content-Type: Application/JSON,
Content-Lenght: 170]
{
  id: 54321,
  status: Enabled,
  staticConfiguration: false,
  dhcpServer: 192.168.0.1,
}
```

HTTP GET – retrieving more than 1 resource (at once)



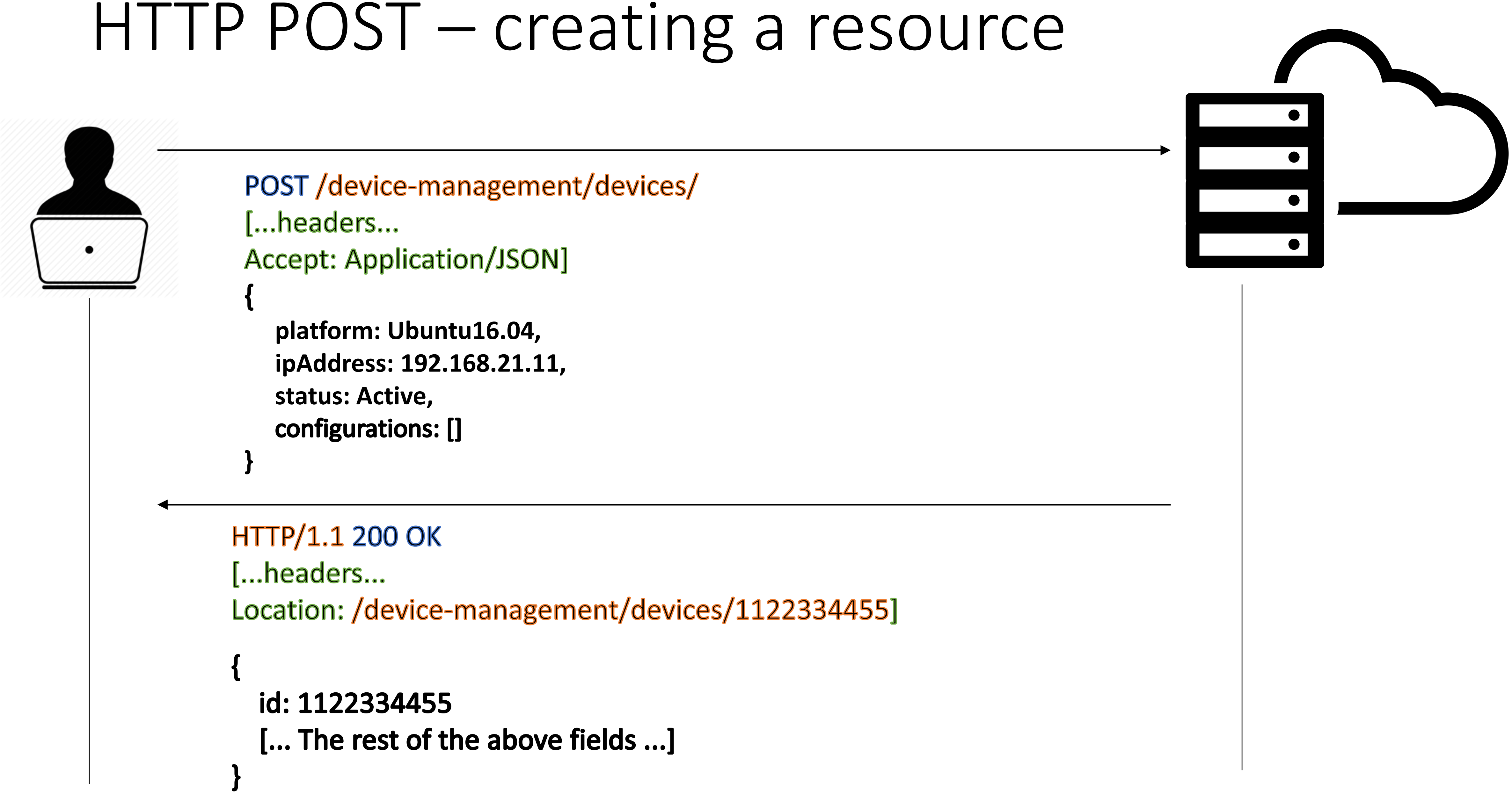
```
GET /device-management/devices?id=12345&id=45678&id=....
[...headers...
Accept: Application/JSON]
<no body>
```

```
HTTP/1.1 200 OK
{ // results }
```

```
POST /device-management/devices/query
{ids: [{id: 12345}, {id: 45678}]}
```

```
HTTP/1.1 200 OK
{ // results }
```

HTTP POST – creating a resource

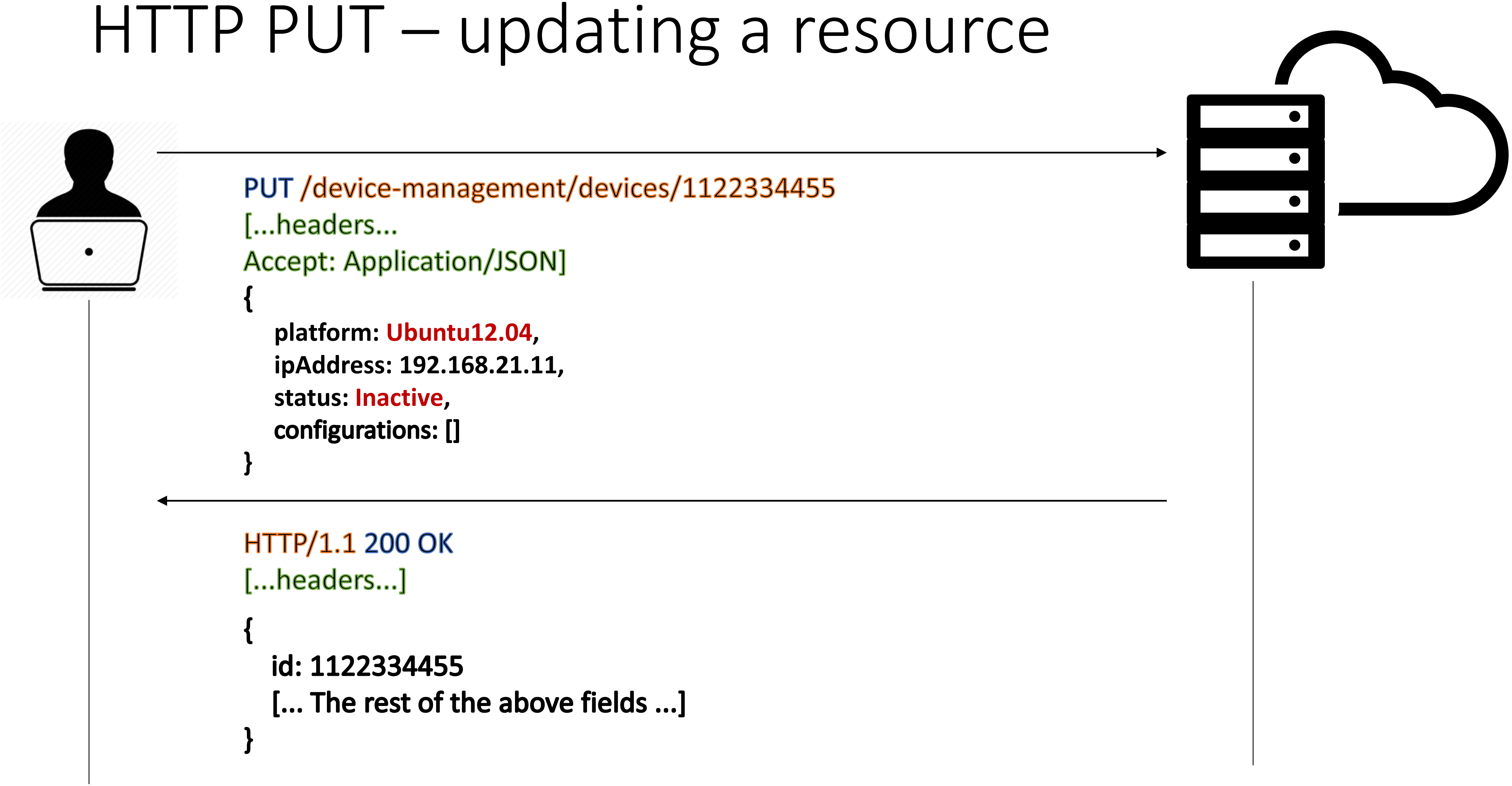


```
POST /device-management/devices/
[...headers...
Accept: Application/JSON]
{
   platform: Ubuntu16.04,
   ipAddress: 192.168.21.11,
   status: Active,
   configurations: []
}
```

```
HTTP/1.1 200 OK
[...headers...
Location: /device-management/devices/1122334455]

{
  id: 1122334455
  [... The rest of the above fields ...]
}
```

HTTP PUT – updating a resource

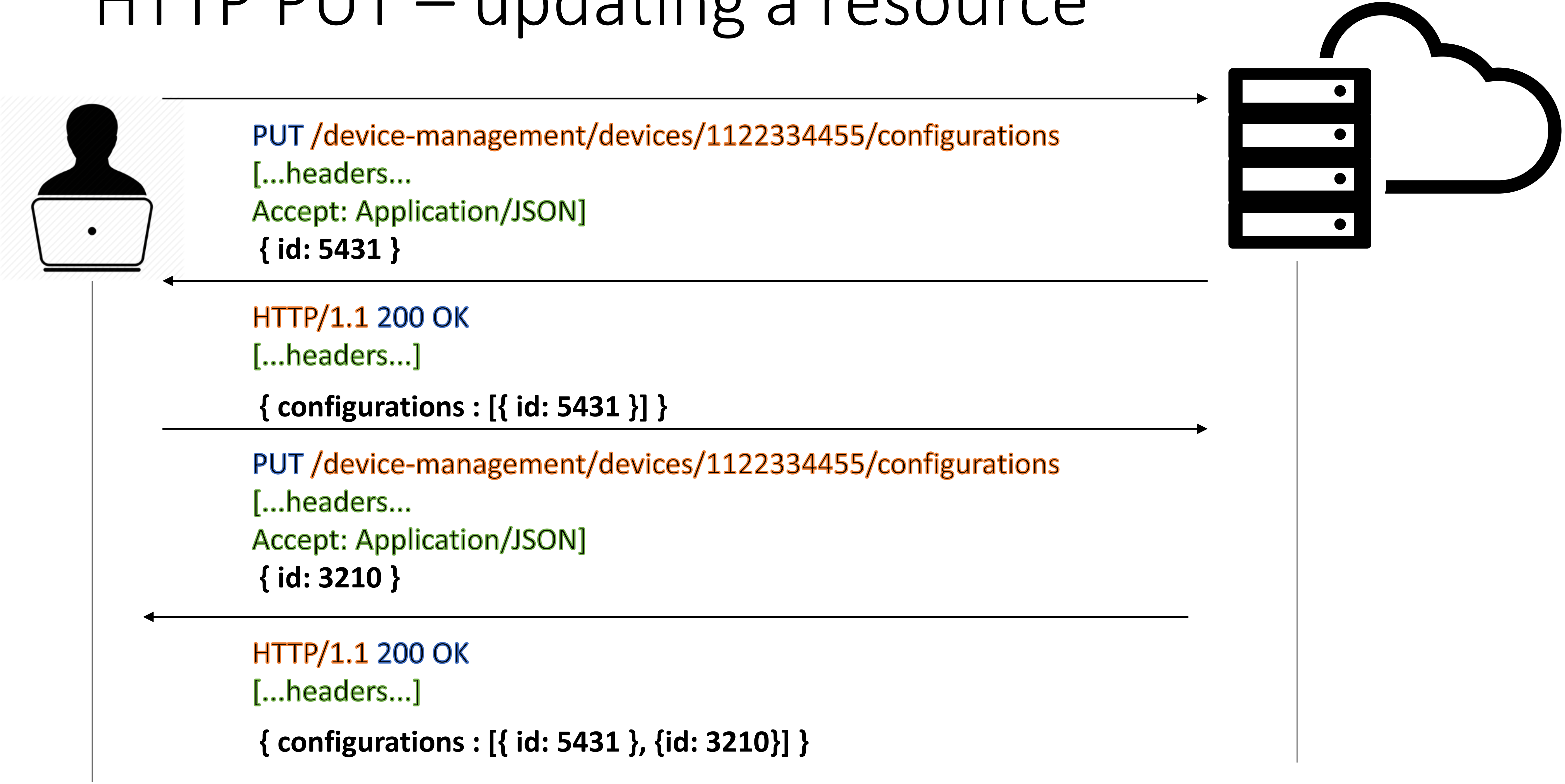


```
PUT /device-management/devices/1122334455
[...headers...
Accept: Application/JSON]
{
    platform: Ubuntu12.04,
    ipAddress: 192.168.21.11,
    status: Inactive,
    configurations: []
}
```

```
HTTP/1.1 200 OK
[...headers...]

{
    id: 1122334455
    [... The rest of the above fields ...]
}
```

HTTP PUT – updating a resource



PUT /device-management/devices/1122334455/configurations
[...headers...
Accept: Application/JSON]
{ id: 5431 }

HTTP/1.1 200 OK
[...headers...]

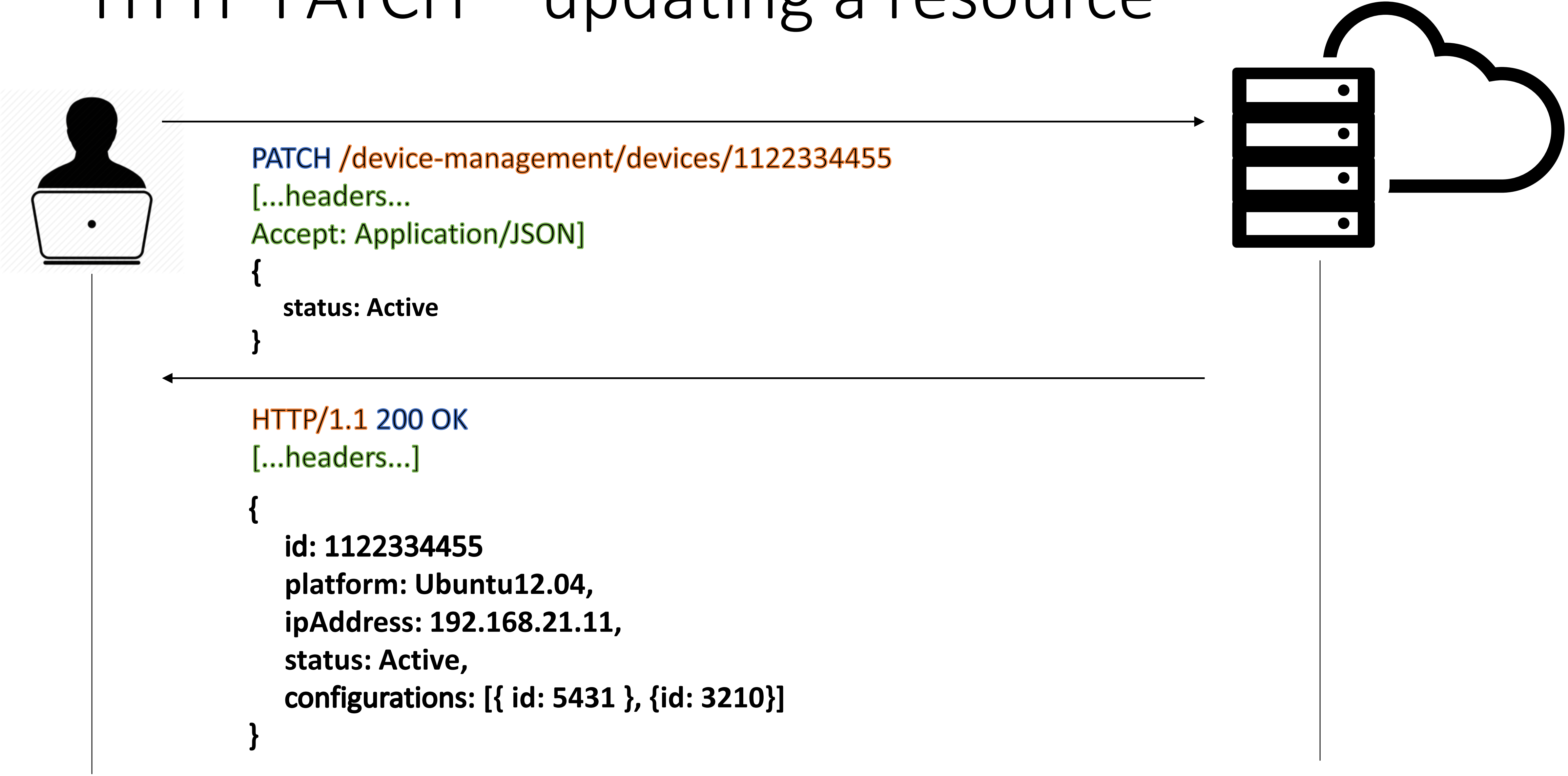
{ configurations : [{ id: 5431 }] }

PUT /device-management/devices/1122334455/configurations
[...headers...
Accept: Application/JSON]
{ id: 3210 }

HTTP/1.1 200 OK
[...headers...]

{ configurations : [{ id: 5431 }, {id: 3210}] }

HTTP PATCH – updating a resource

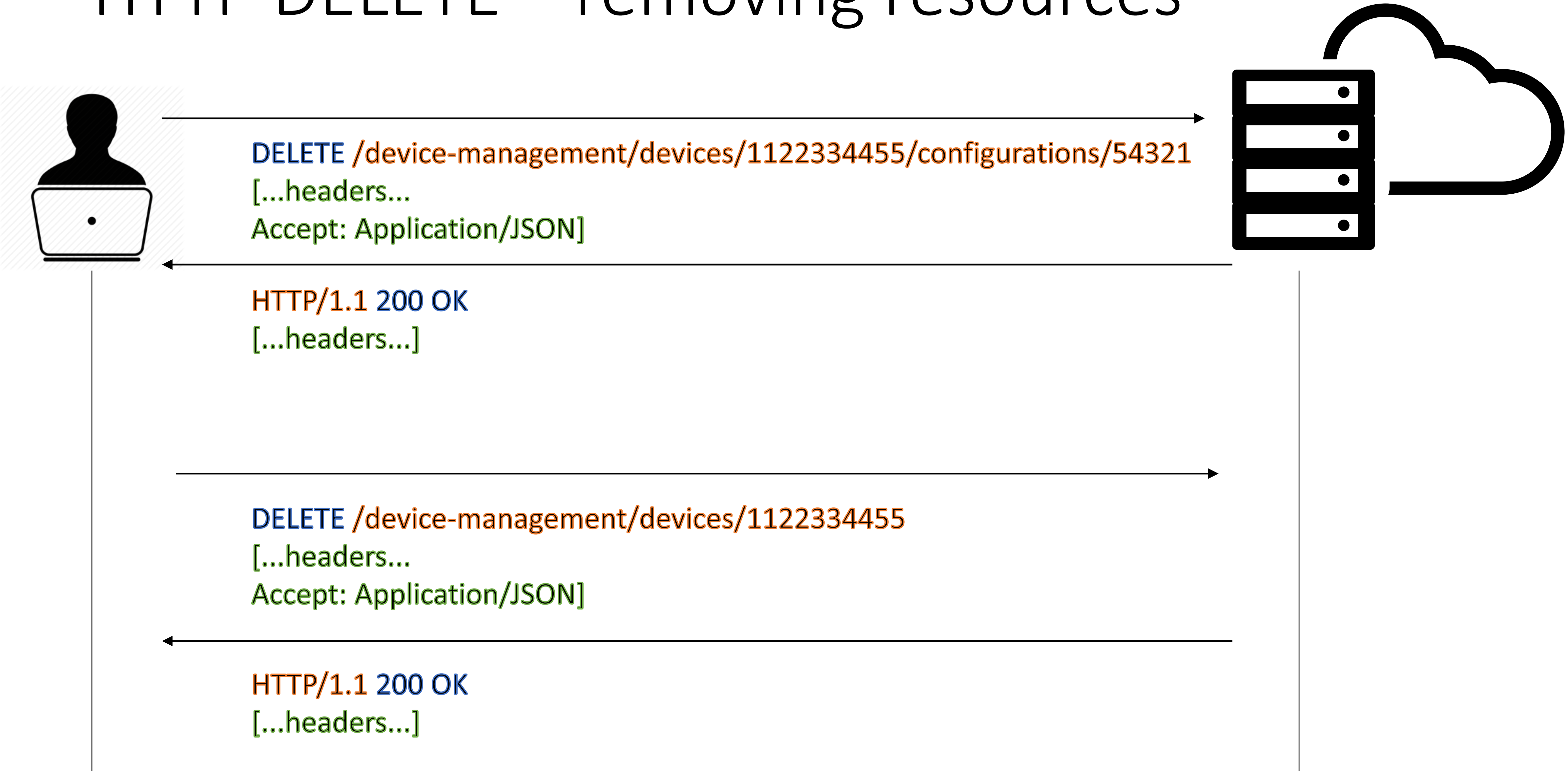


```
PATCH /device-management/devices/1122334455
[...headers...
Accept: Application/JSON]
{
    status: Active
}
```

```
HTTP/1.1 200 OK
[...headers...]

{
    id: 1122334455
    platform: Ubuntu12.04,
    ipAddress: 192.168.21.11,
    status: Active,
    configurations: [{ id: 5431 }, {id: 3210}]
}
```

HTTP DELETE – removing resources



DELETE /device-management/devices/1122334455/configurations/54321
[...headers...
Accept: Application/JSON]

HTTP/1.1 200 OK
[...headers...]

DELETE /device-management/devices/1122334455
[...headers...
Accept: Application/JSON]

HTTP/1.1 200 OK
[...headers...]

HATEOAS – Linking things together

- hypermedia as the engine of application state

```
device: {
   id: 12345,
   name: eth0,
   platform: Ubunt16.04,
   ipAddress: 192.168.21.10,
   status: Active,
   link: {
      href: /devices/12345,
      rel: devices,
      type: GET
   },
   configurations: [{ ... }]
}
```

```
configurations: [{
         id: 54321,
         link: {
            href: /configurations/54321,
            rel: configurations,
            type: GET
         }
      },{
         id: 433210,
         link: {
            href: /configurations/433210,
            rel: configurations,
            type: GET
         }
      }]
```

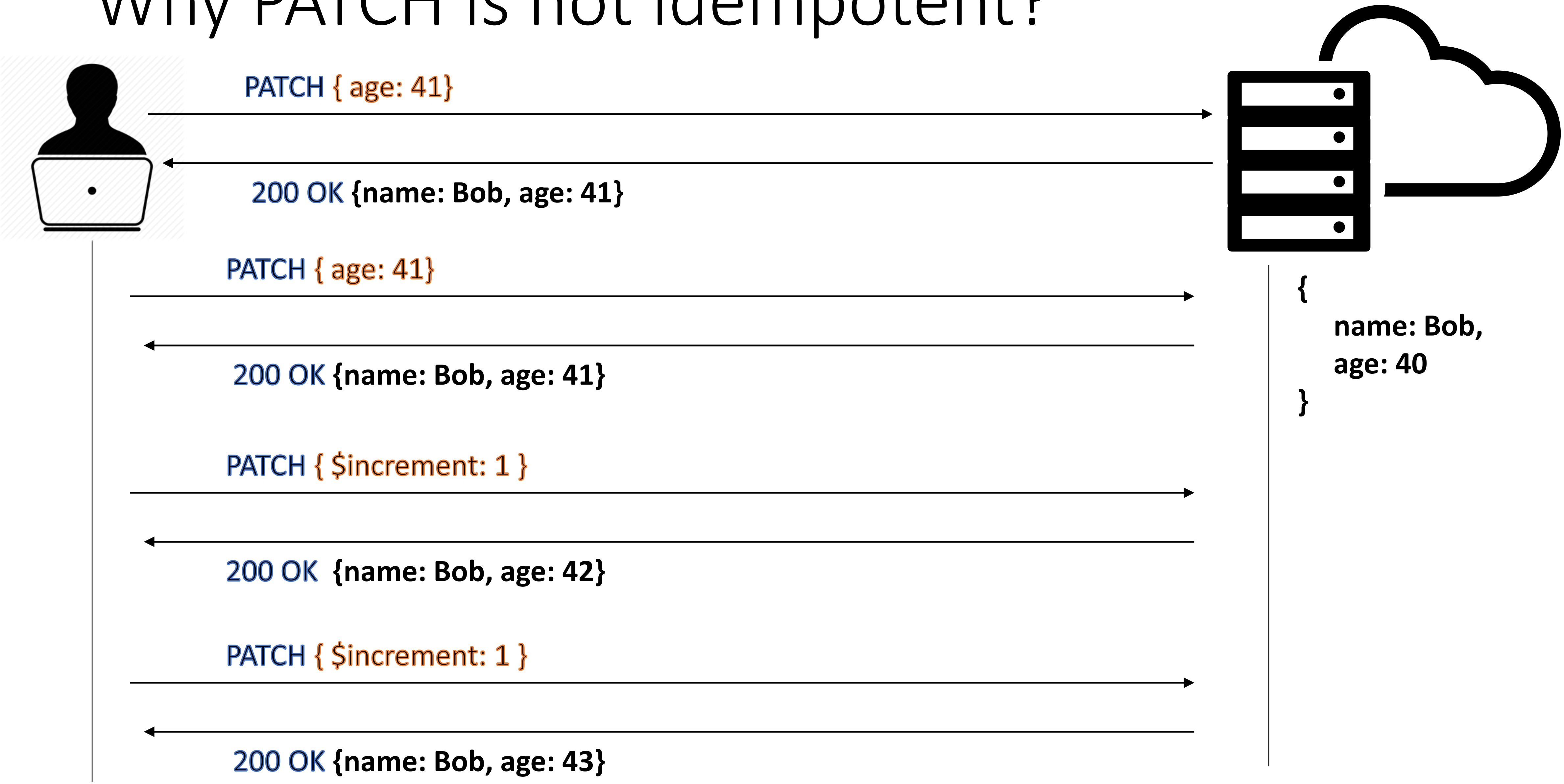
Idempotent and Safe HTTP Methods - Why Do They Matter?

- An idempotent operation produces the same results when executed once or multiple times
- A safe method does not modify resources
- Idempotency and safety make an API fault-tolerant and robust
  - Facilitates caching
- a = 4; // idempotent, but not safe
- a++; // not idempotent

Idempotent and Safe HTTP Methods

Method	Idempotent	Safe
GET	YES	YES
HEAD	YES	YES
OPTIONS	YES	YES
POST	No	No
PUT	YES	No
PATCH	No	No
DELETE	YES	No

Why PATCH is not idempotent?



PATCH { age: 41}

200 OK {name: Bob, age: 41}

PATCH { age: 41}

200 OK {name: Bob, age: 41}

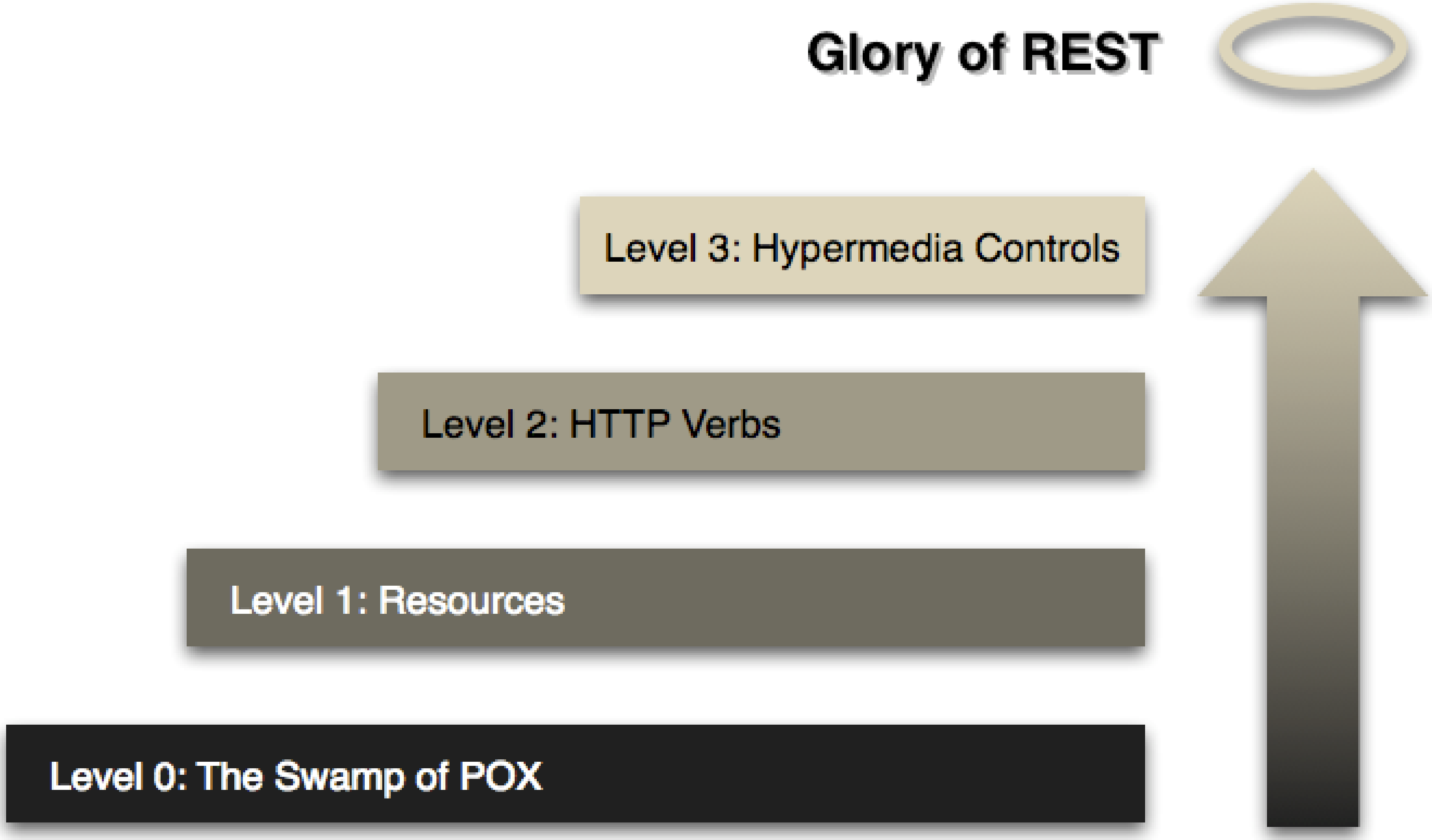
PATCH { $increment: 1 }

200 OK {name: Bob, age: 42}

PATCH { $increment: 1 }

200 OK {name: Bob, age: 43}

```
{
    name: Bob,
    age: 40
}
```



Glory of REST

Level 3: Hypermedia Controls

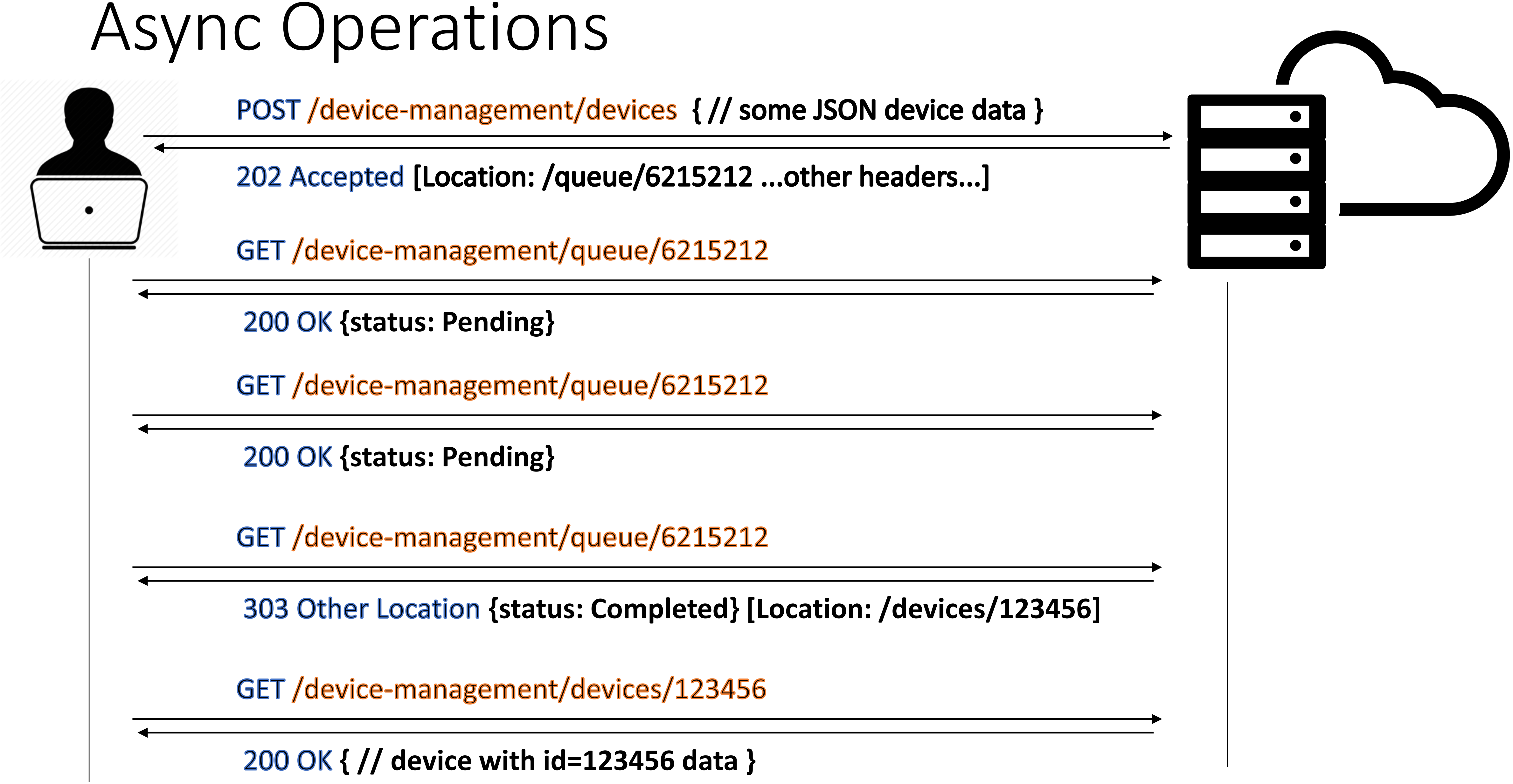
Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

Scaling Up

Async Operations



POST /device-management/devices { // some JSON device data }

202 Accepted [Location: /queue/6215212 ...other headers...]

GET /device-management/queue/6215212

200 OK {status: Pending}

GET /device-management/queue/6215212

200 OK {status: Pending}

GET /device-management/queue/6215212

303 Other Location {status: Completed} [Location: /devices/123456]

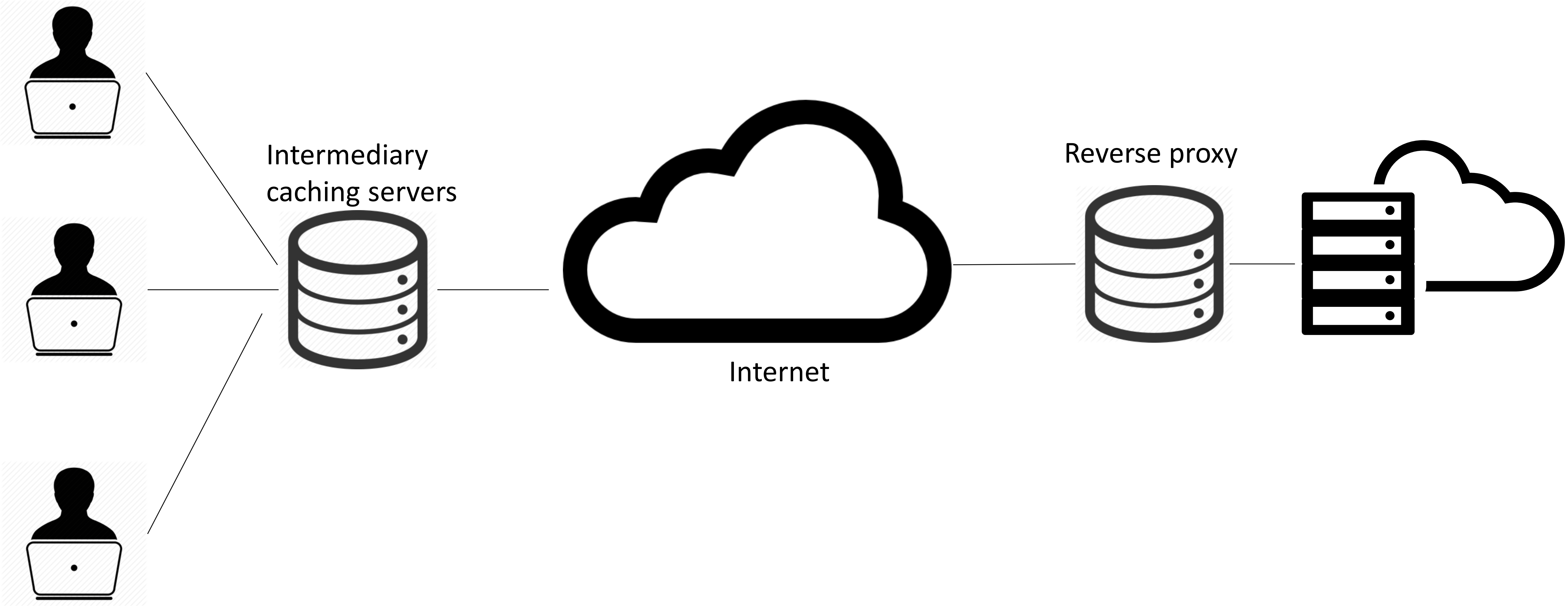
GET /device-management/devices/123456

200 OK { // device with id=123456 data }

Caching

- storing reusable responses in order to make subsequent requests faster
- What to cache:
  - Logos and brand images
  - Style sheets and JS files
  - Downloadable Content and Media Files
  - HTML pages and Content requested with authentication cookies (attention)
- What not to cache:
  - Sensitive data (banking accounts)
  - Frequently changing data (depends by traffic volume)

Where to cache?



Intermediary caching servers

Reverse proxy

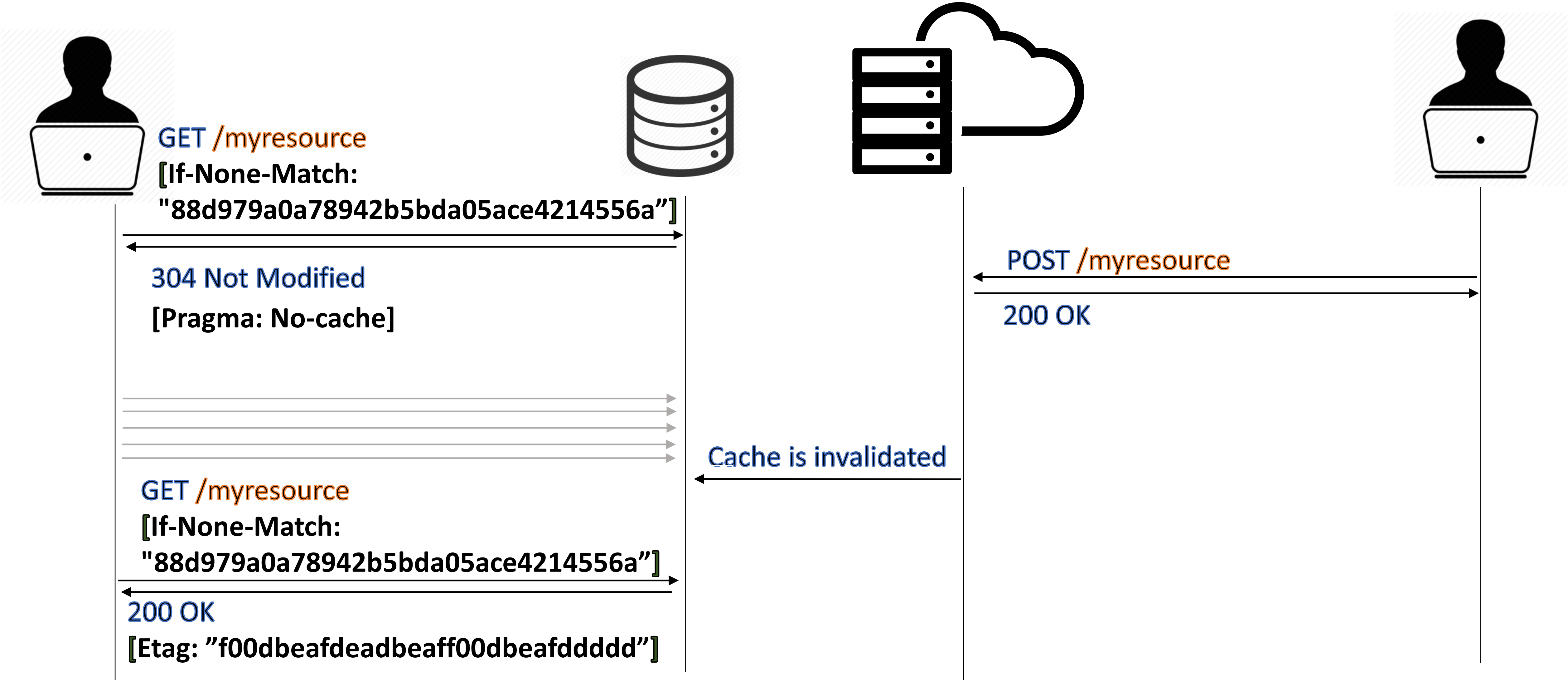
Internet

Client caches (browsers)

Caching techniques

- Write-through
  - Data written to cache and backing store at the same time
  - I/O completion when both operations complete
- Write-around
  - Data written only to the backing store
- Write-back
  - Data written to cache
  - Data written in the background at the backing store
- Cache invalidation

Etag (entity tag) HTTP headers



GET /myresource

[If-None-Match:
"88d979a0a78942b5bda05ace4214556a”]

304 Not Modified

[Pragma: No-cache]

POST /myresource

200 OK

Cache is invalidated

GET /myresource

[If-None-Match:
"88d979a0a78942b5bda05ace4214556a”]

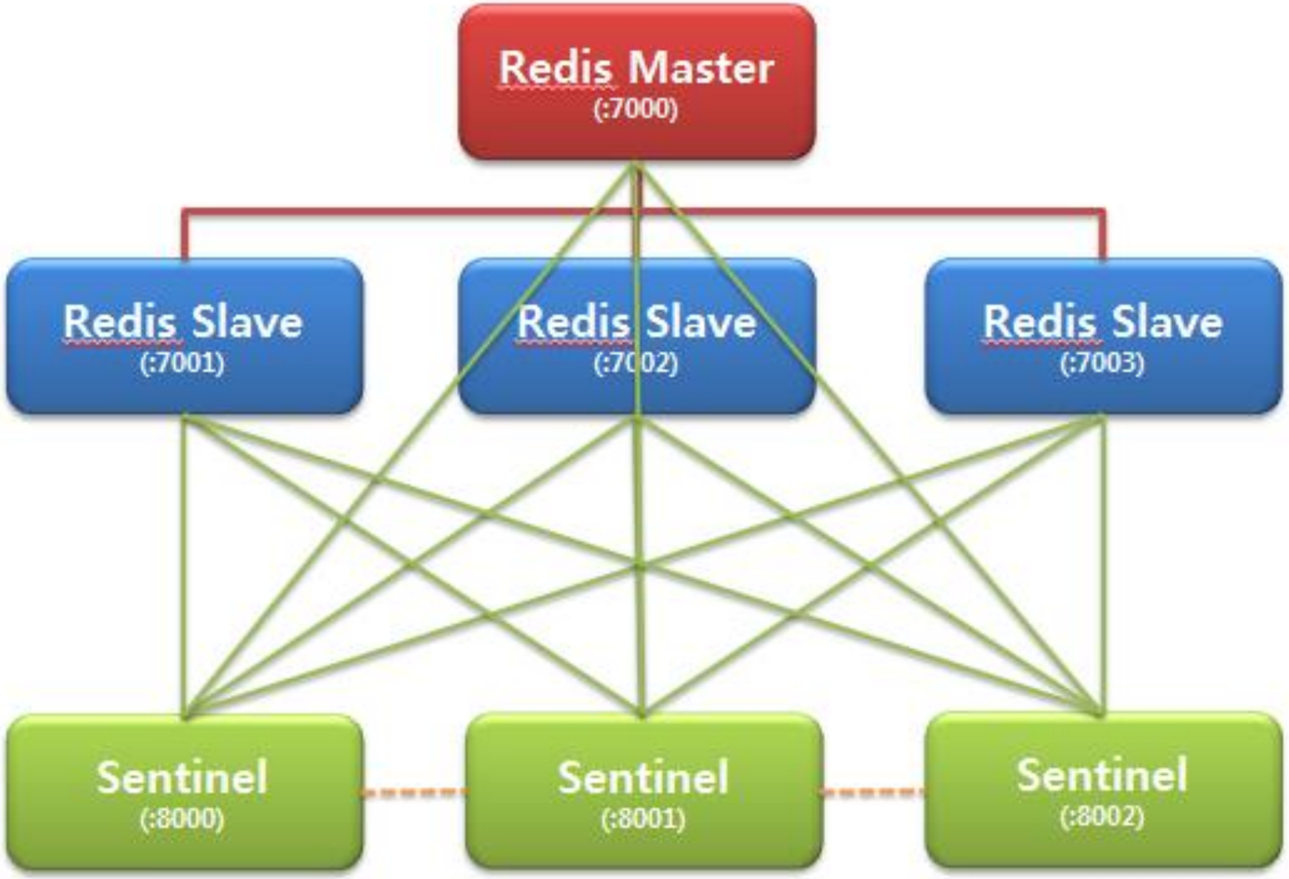
200 OK

[Etag: ”f00dbeafdeadbeaff00dbeafddddd”]

Some real caching headers

- If-Modified-Since (client) Last-Modified(server)
- Cache-control: max-age
- Expires

Caching as a service with REDIS



Conclusion

- Core REST concepts:
  - Resource & Resource representation
  - Client - Server
  - Request – Response
- REST over HTTP
  - Standard RESTfull status codes
  - Keep operations safe & idempotent
- Scalable API
  - Async operations
  - Cache: What to cache? Where? For how long?
    - Get some expected throughput beforehand
    - Premature optimization is the root of all evil!

Q&A