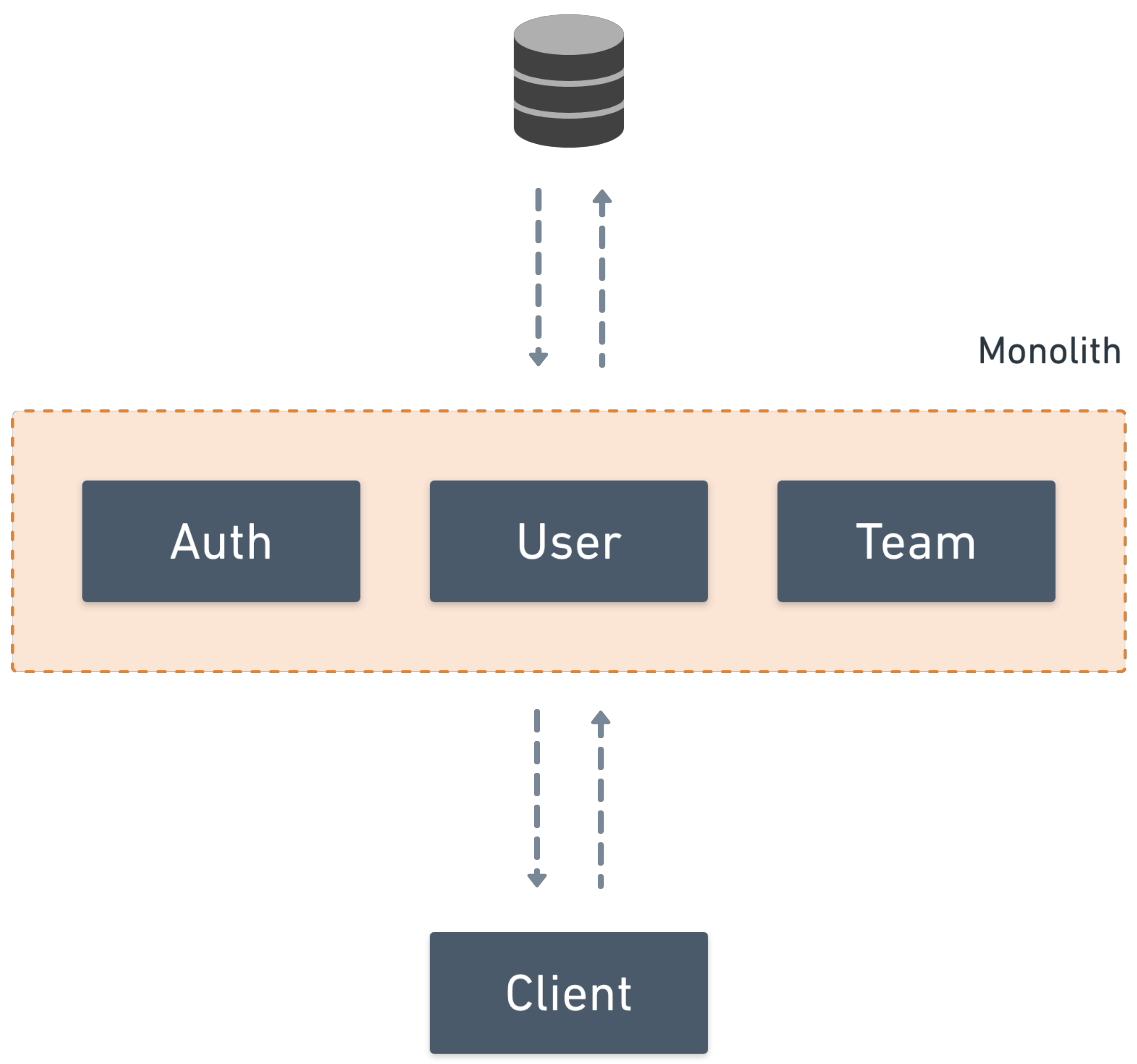
Building a secure BFF

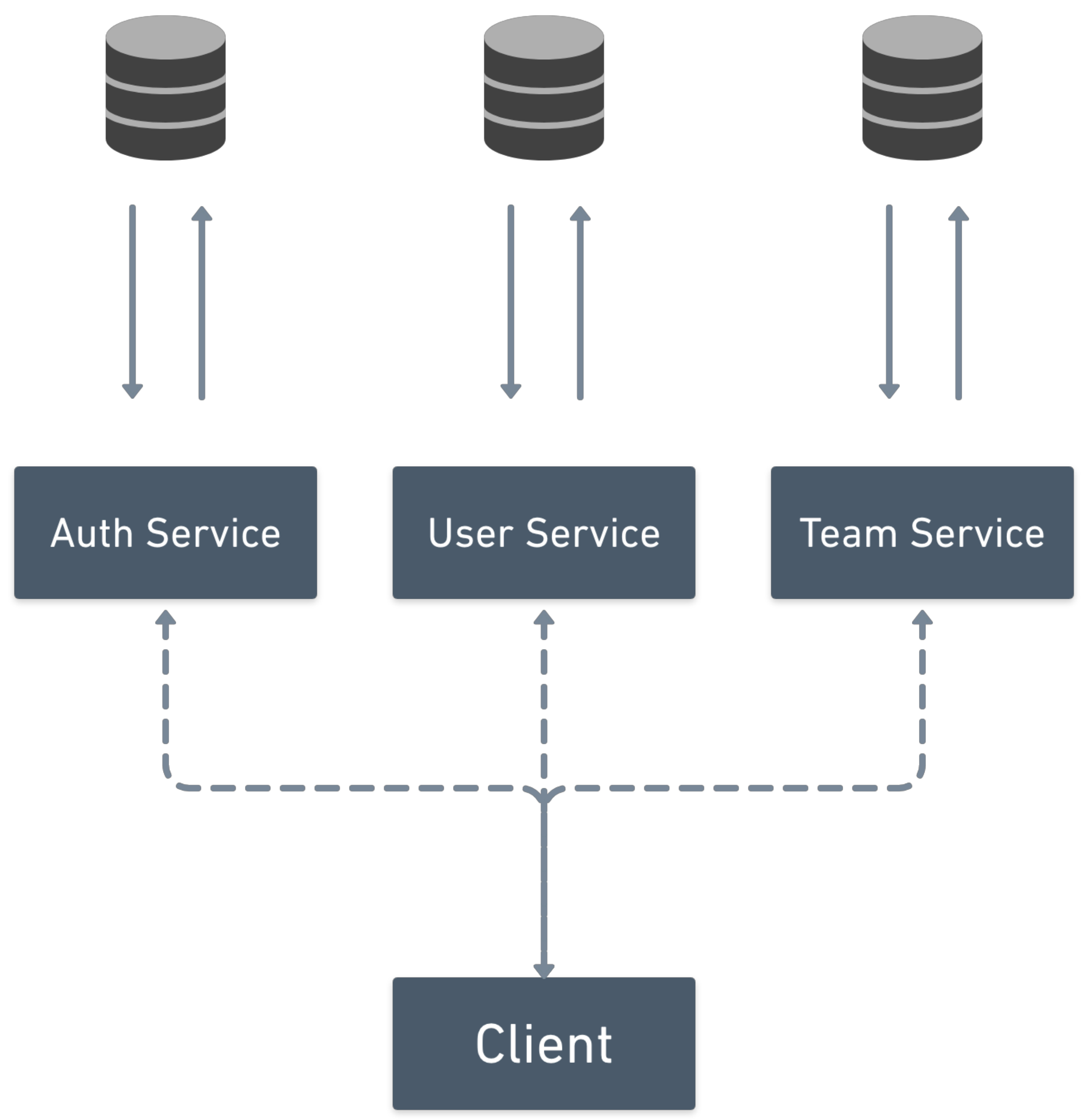
Ankit Muchhala - Postman

ankit_muchhala

Monolith

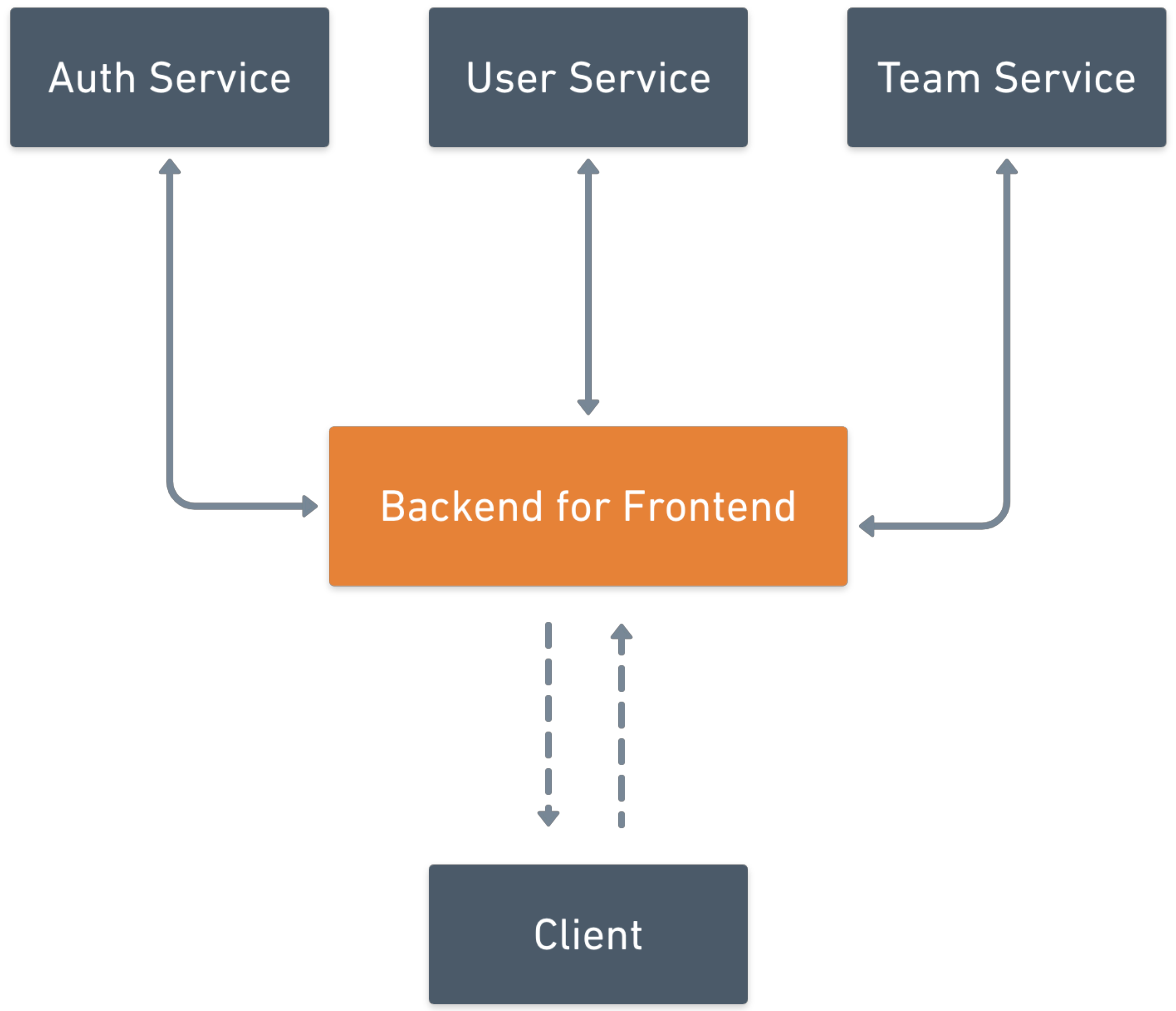


Microservices



Backend For Frontend

- BFF is an **API Gateway** designed for a **specific UI** to interact with microservices.
- Abstracts away implementation details from client.
- Reduces network *chatter* and improves performance.



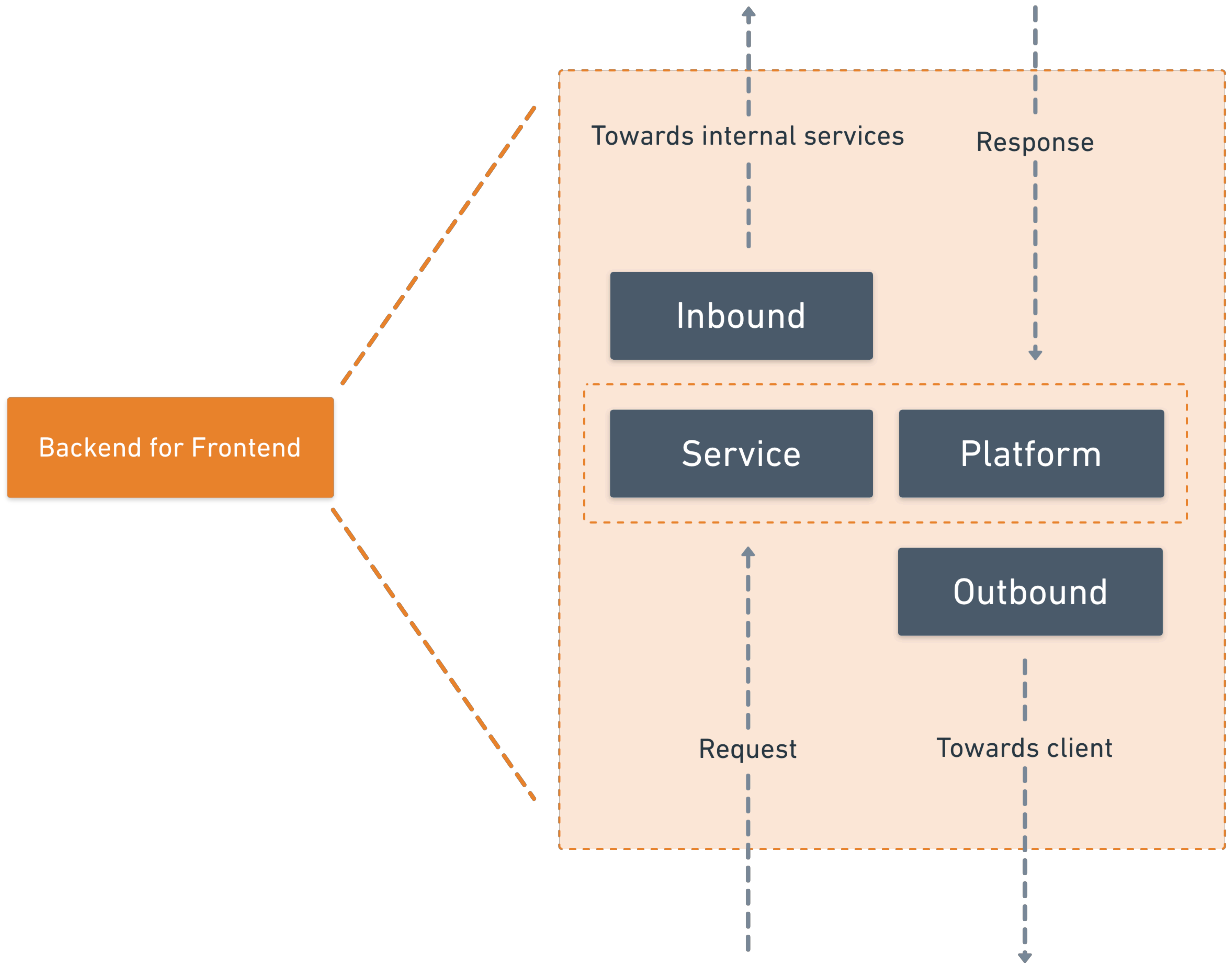
Security concerns?

- Single point of failure and attack.
- Public facing service.
- Handles user input.

- How to **quantify** these for an API?

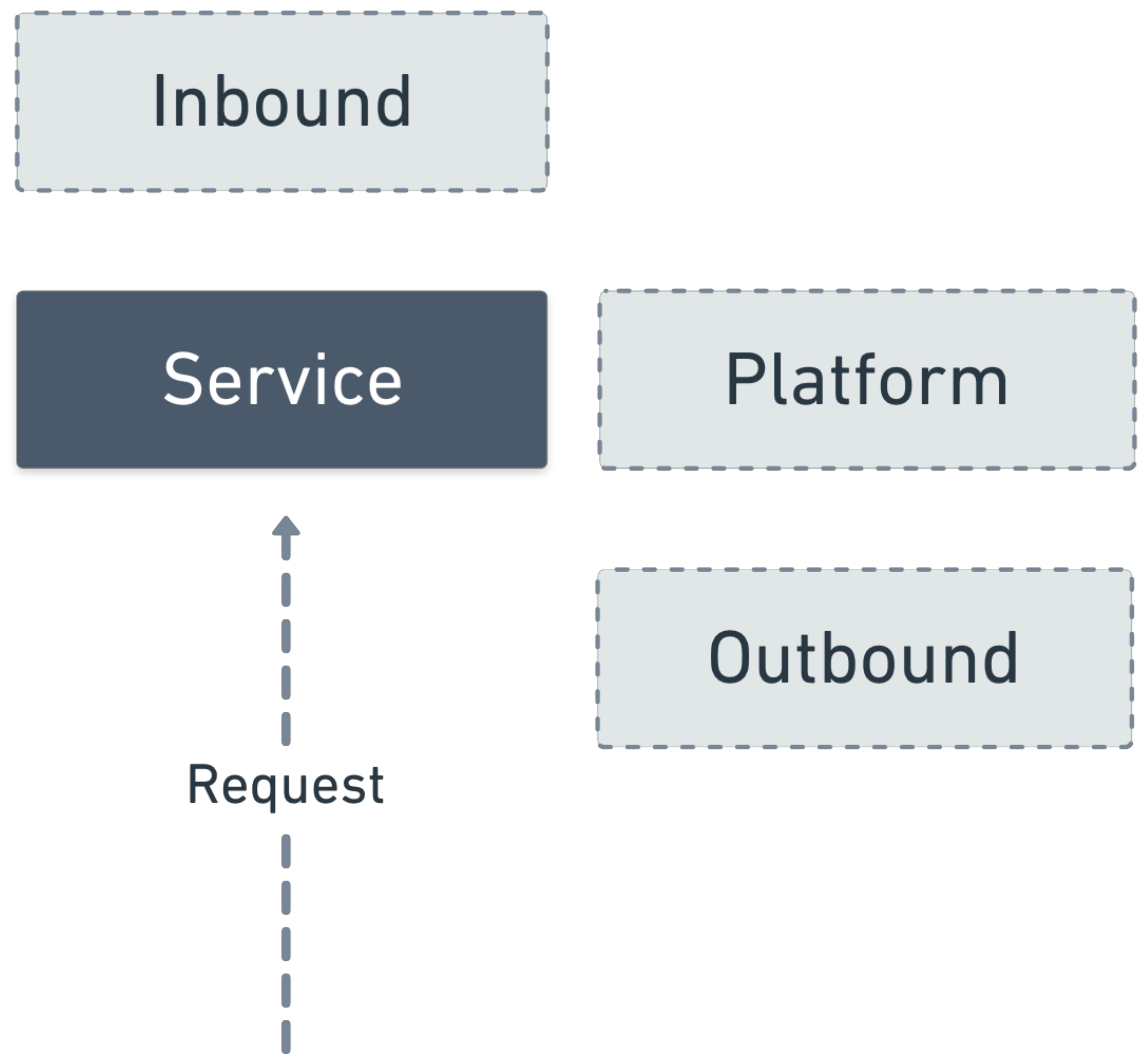
Security Parameters

Confidentiality	Integrity	Availability
Only **authorised** people can access appropriate data.	Data delivered by your service is **not tampered** with.	Content is available to authorised users to **on demand**.


Backend for Frontend
Towards internal services
Response
Inbound
Service
Platform
Outbound
Request
Towards client

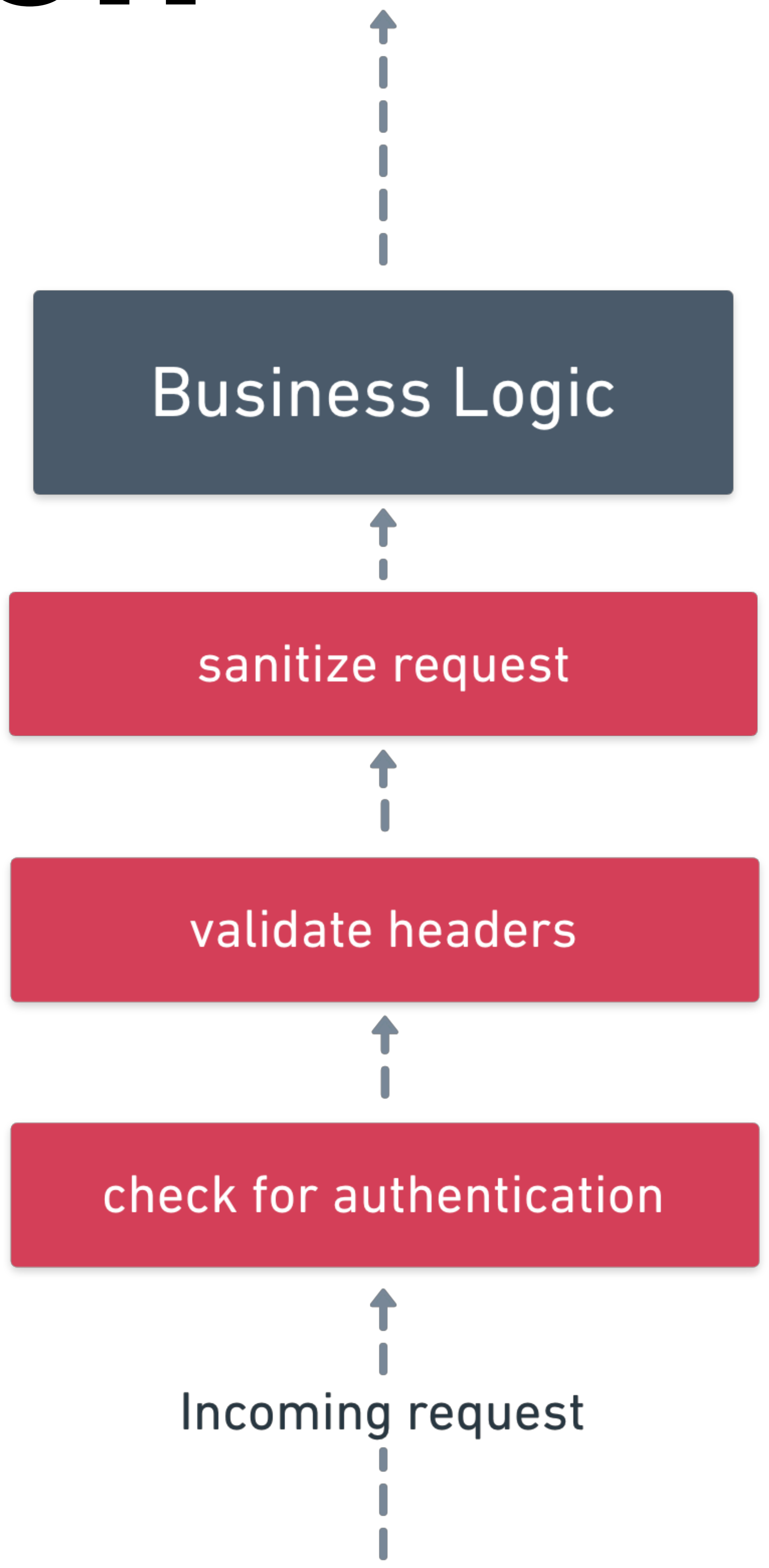
Service

Server side code which contains your business logic



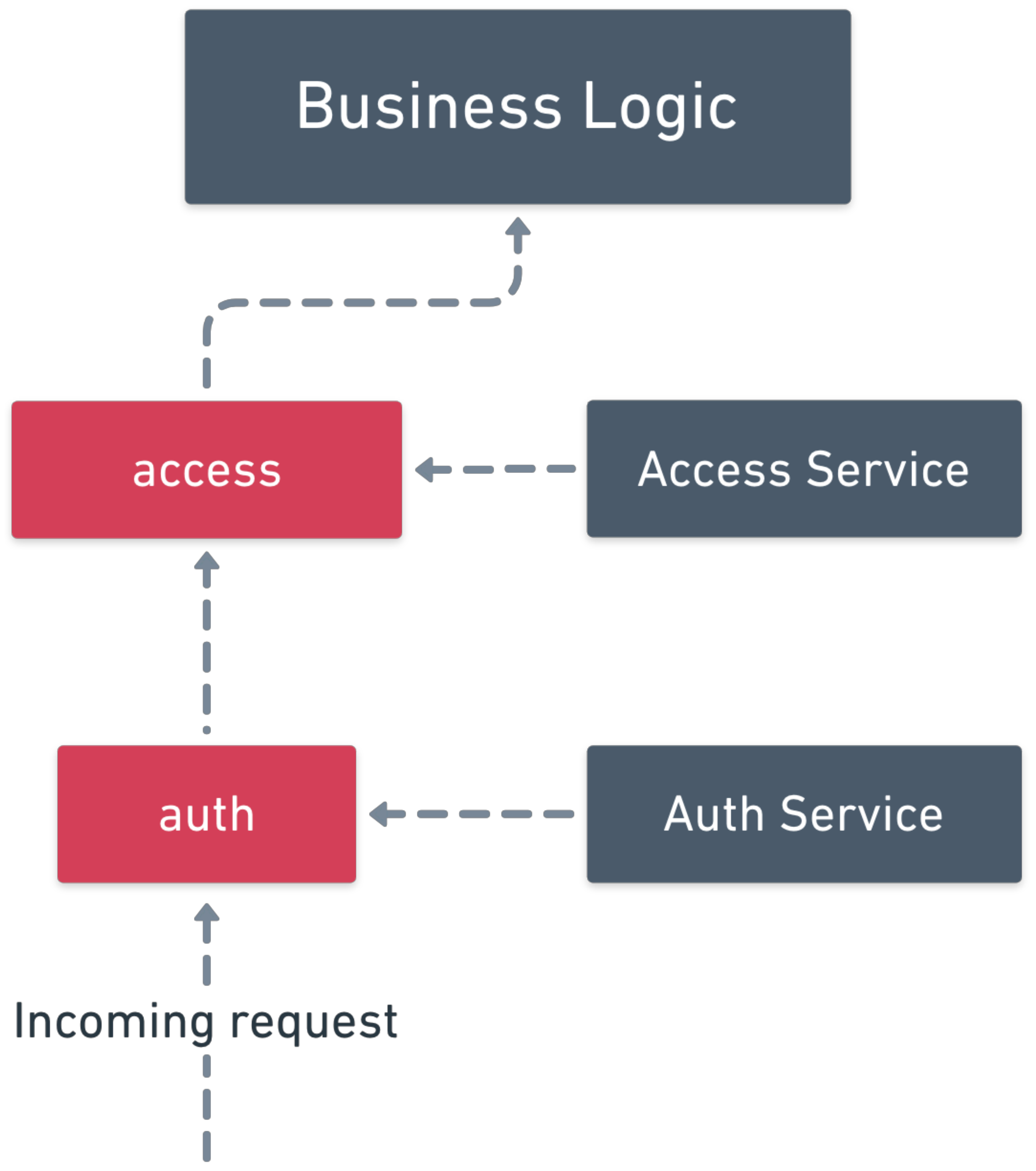
Validation

- BFF should not perform all validations.
- It should perform **ecosystem checks** - auth, validate header.
- Business logic specific checks are **deferred** to downstream services.



Critical Path

- Services called before request reaches the business logic.
- Critical **path length** is an indicator of the amount of validation done on BFF.
- Good to have a **short** critical path and **fallbacks**.



Principle of Least Privilege

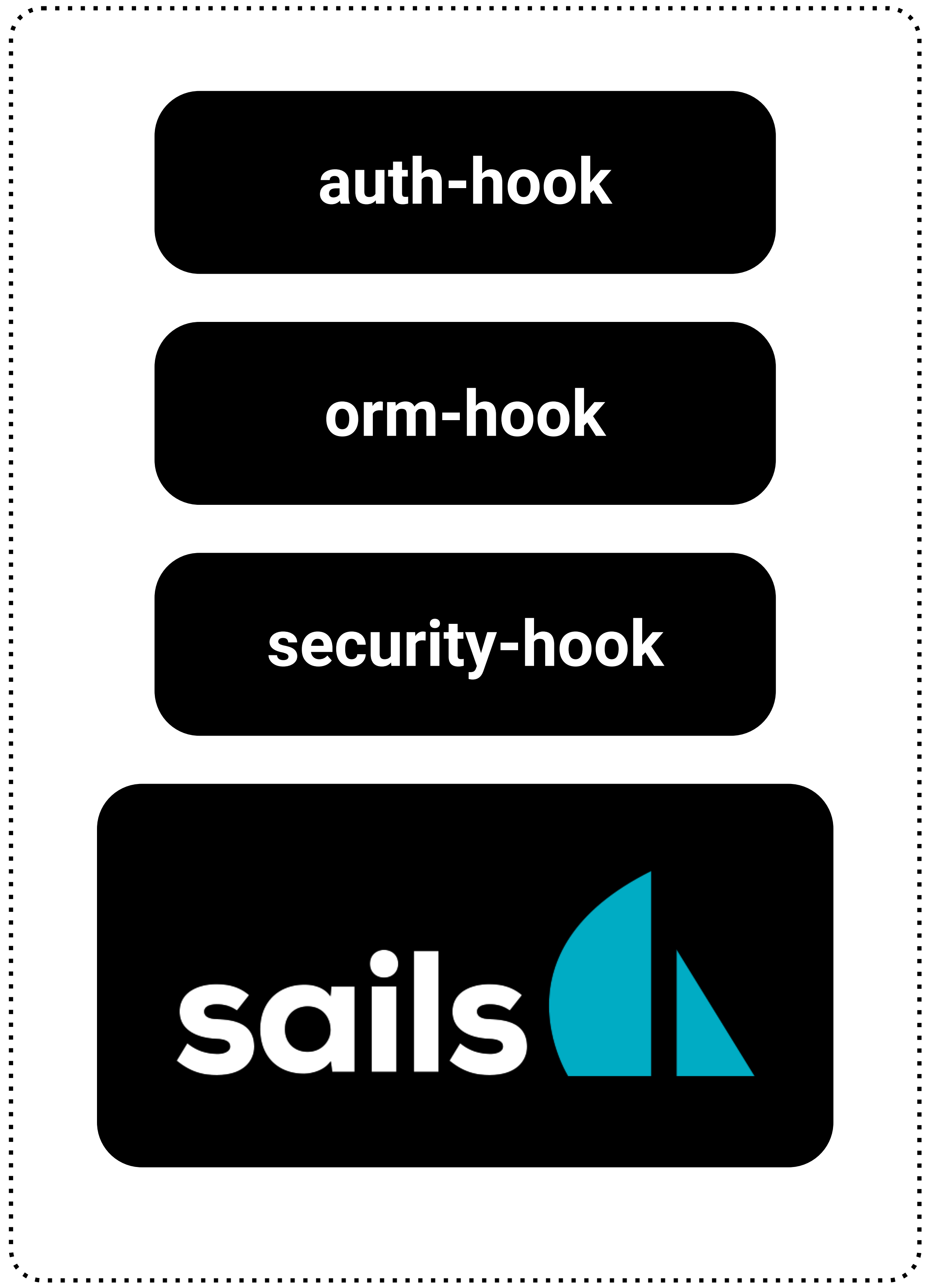
- User only has access to **minimum resources** that are necessary.
- Always assume user does not have access by default.
- Allow only for **specific conditions**.

```
function hasAccess (user, owner) {
  if (user.isAdmin) {
    return true;
  }

  if (user.id === owner.id) {
    return true;
  }

  return false;
}
```

Sample BFF

Architecture

- Make it harder to be insecure.
- Separate business logic from access control and validation.
- Stack installation with predefined security setup using **yeoman**.



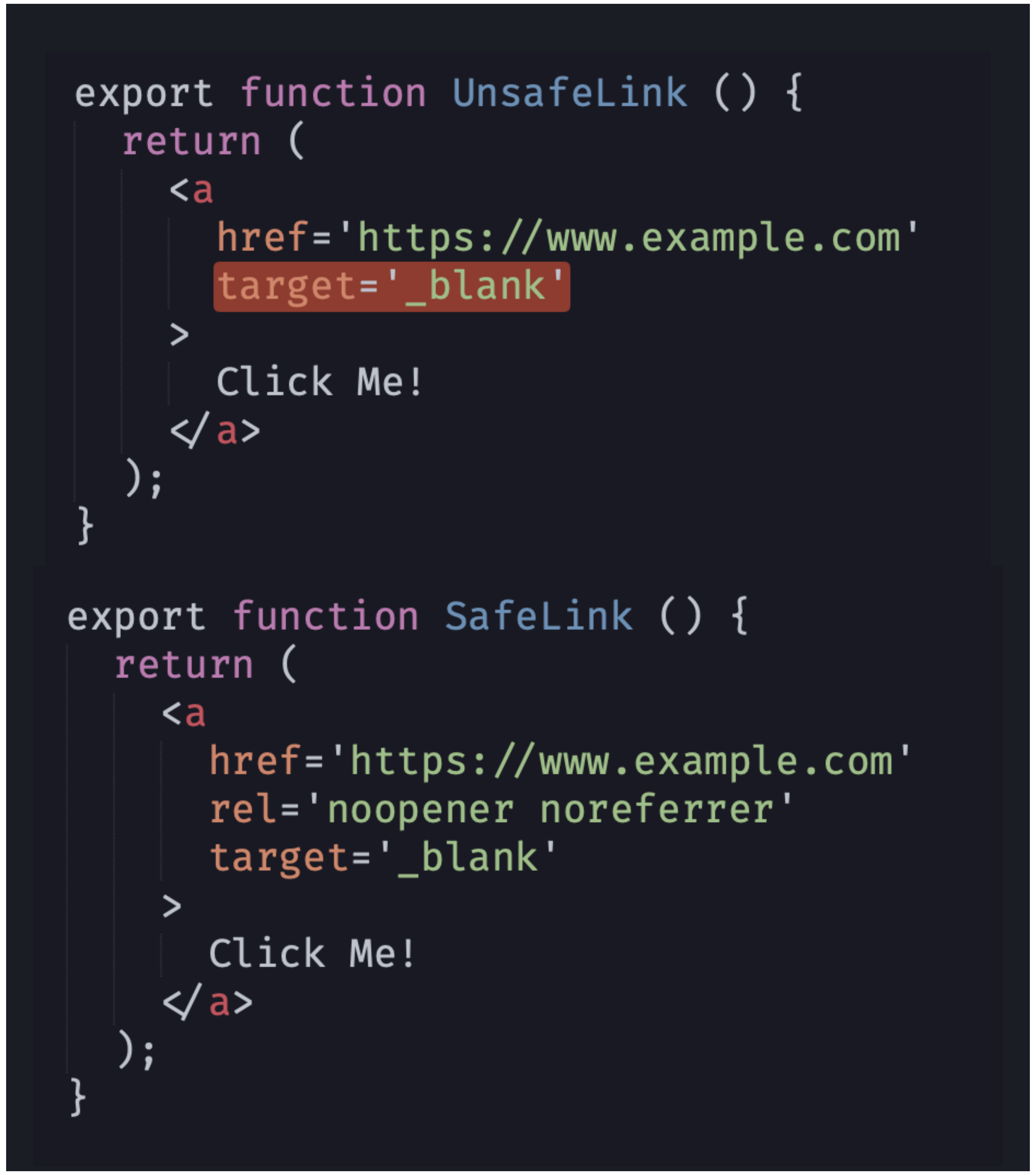
Vulnerable Dependencies

- Use **strict versions** for dependencies and **lockfiles.**
- Check vulnerable dependencies in CI pipeline.
- Tools - **`nsp`**, **`npm audit`**, **`snyk`**.

```
$ snyk test

✗ Medium severity vulnerability
  Description: ReDoS
  Introduced through: something@0.9.1
  Resolution: ...

✗ Medium severity vulnerability
  Description: TOCTOU
  Introduced through: package@1.2.0
  Resolution: ...
```

Enforcing Security

- Security **linting**.
- System tests to catch configuration issues.
- E2E tests using **Postman collections** integrated into the CI pipeline.

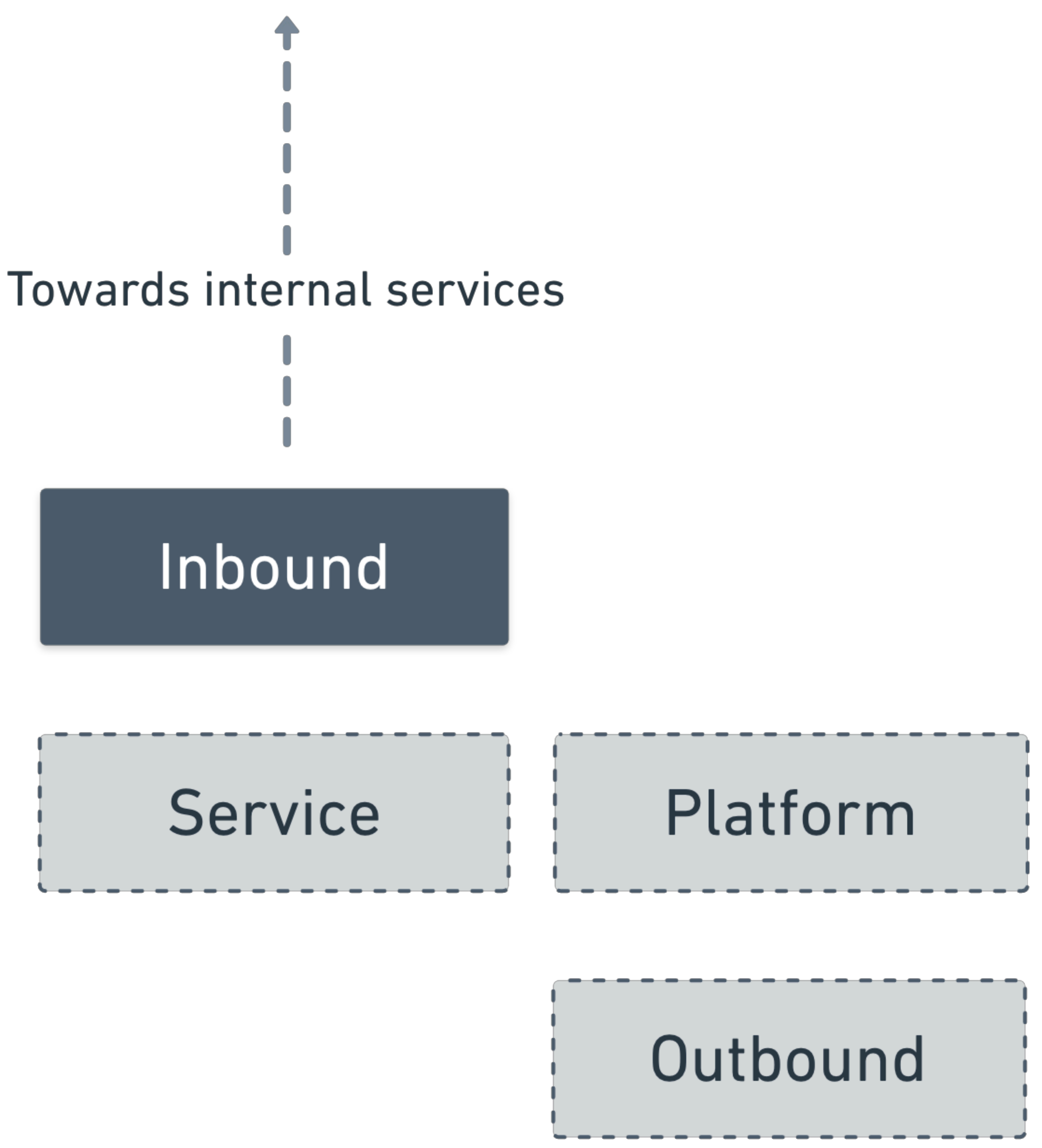


```
export function UnsafeLink () {
  return (
    <a
      href='https://www.example.com'
      target='_blank'
    >
      Click Me!
    </a>
  );
}

export function SafeLink () {
  return (
    <a
      href='https://www.example.com'
      rel='noopener noreferrer'
      target='_blank'
    >
      Click Me!
    </a>
  );
}
```

E2E tests in Newman

Inbound

Interaction with downstream services



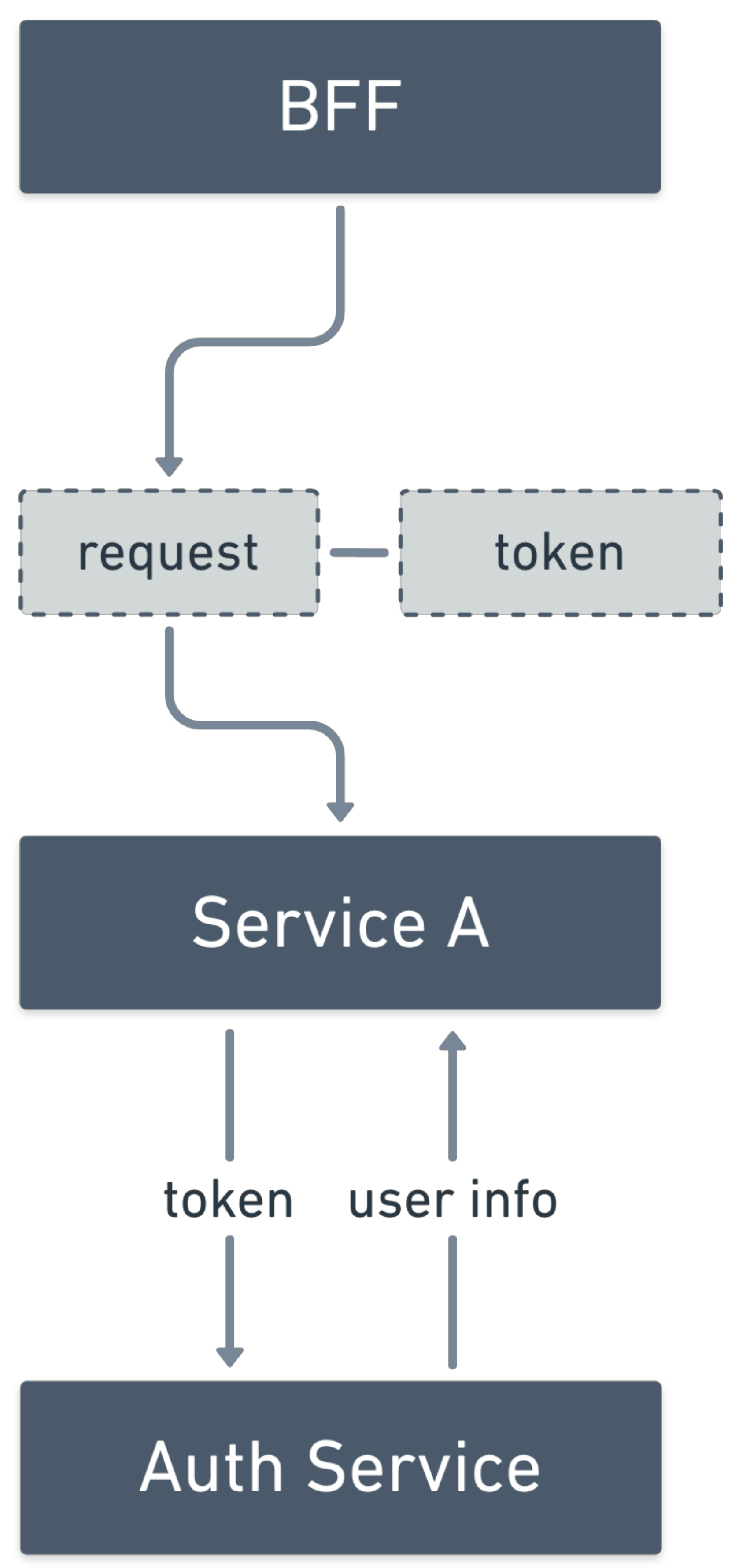
Handling Internal Auth

- Abstracting away internal server details from developer.
- Prevents server auth leak in response or logs.
- Allows for secret rotation without server side code changes.

```
user: async function (req, res) {

  let user = await internal({
    service: 'auth',
    path: '/users/current',
    query: { populate: true }
  });

  return user.toJSON();
}
```

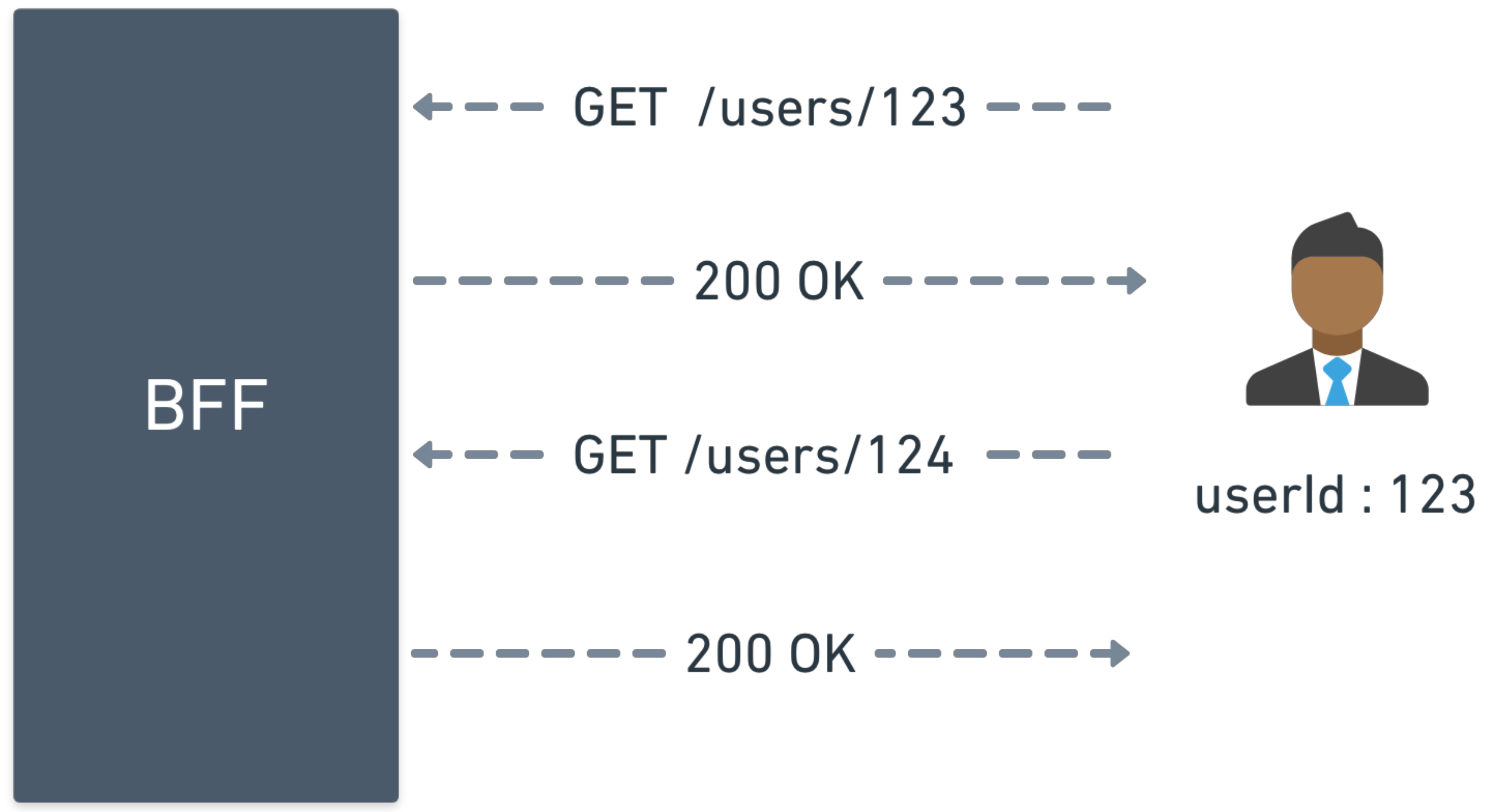
Request tagging

- Associate each incoming request with a user associated **token**.
- Each service can utilize this token to fetch user meta and **apply validations**.



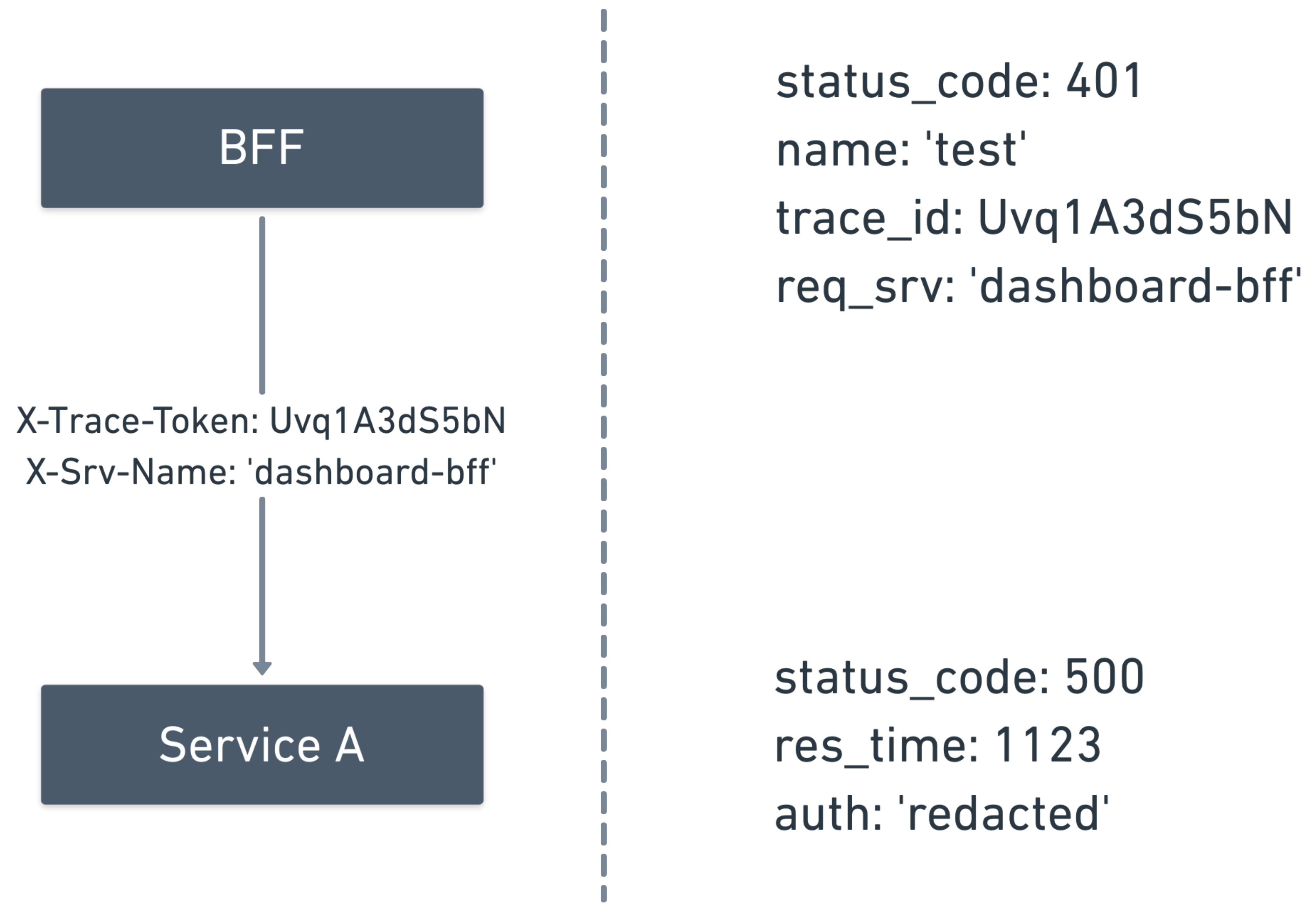
IDOR

- Exposing internal object references along with incorrect access control.
- All user initiated actions must have verifications based on user tokens.



Logging

- **Scrub** logs for sensitive information and user data.
- Use **heuristics** to prevent accidental logging.
- **Trace** logs originating from BFF to track potential **PII movement**.

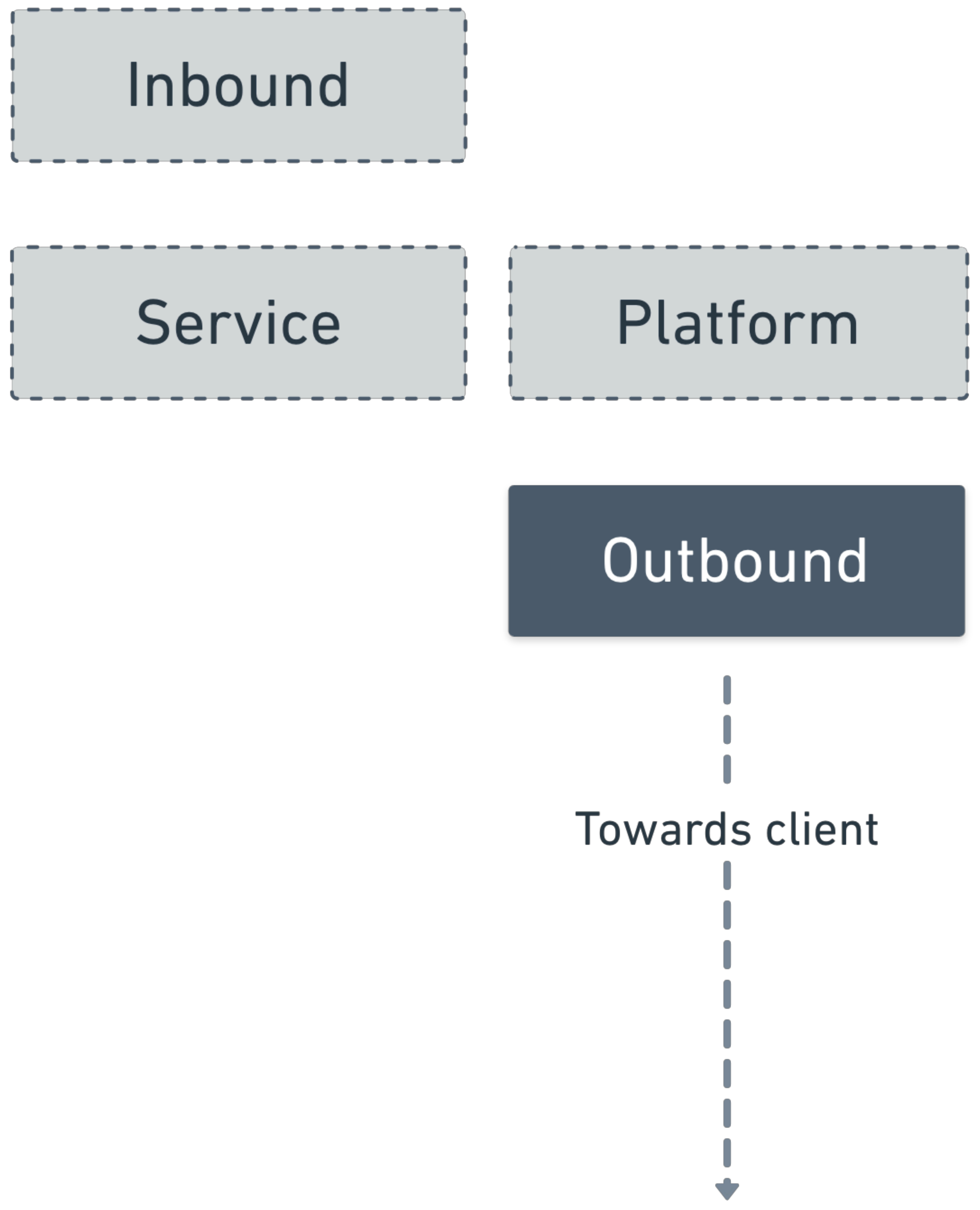


```
status_code: 401
name: 'test'
trace_id: Uvq1A3dS5bN
req_srv: 'dashboard-bff'
```

```
status_code: 500
res_time: 1123
auth: 'redacted'
```

Outbound

Content security while communicating with the client.



HTTPS / HSTS

- Choose the certificate based on your need and the level of user trust required - DV, OV, EV

- Ensure 3rd party calls and redirections are over HTTPS.

- Implement HSTS (+ preload) once you have verified everything is over HTTPS.

RFC 2818, 6797

CSP

- Reduces the harm caused by malicious code injection.
- Start by using **report-only** mode to prevent side effects.
- Not ideal to prevent data exfiltration - *hrefs not covered.*

```
Content-Security-Policy:
  connect-src: 'self'
  script-src: 'none'
  img-src: *
  default-src: 'none'
  report-uri: 'https://...'
```

RFC 7762

Other Headers

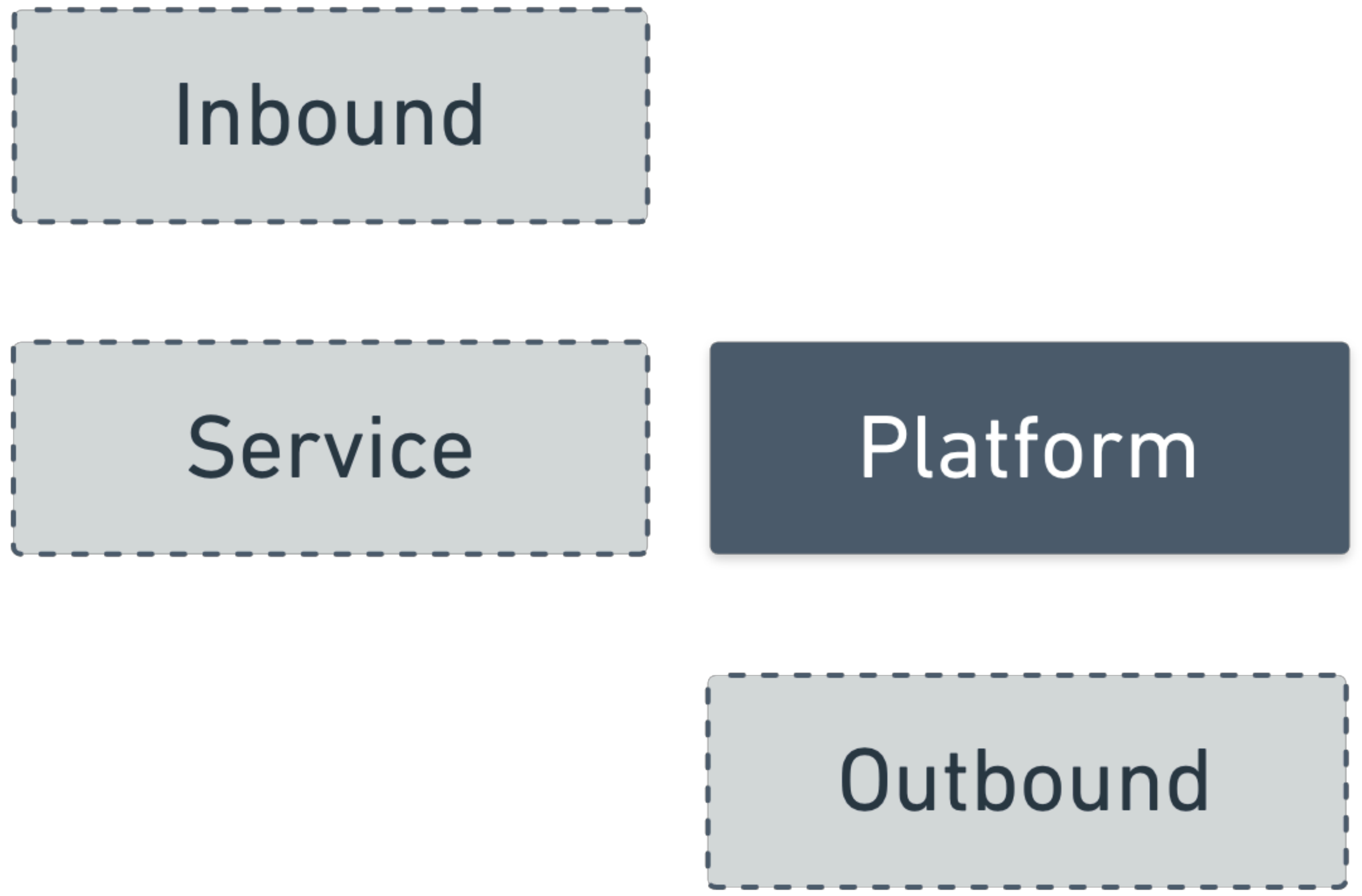
- **CORS**: Who can access your resource.
- **X-XSS**: Detect and prevent XSS in some browsers.
- **X-Frame-Options**: Permit or deny displaying the website within an iframe.
- **HPKP**: Allows HTTPS websites to resist impersonation.
- **SRI**: Verify 3rd party assets
- Refer OWASP Security Headers Project for more.

Caveats

- The support for all the headers is dependent on client browser.
- Cannot be solely relied on for securing your BFF.
- **Not** a replacement for deliberate input validation and output formatting.

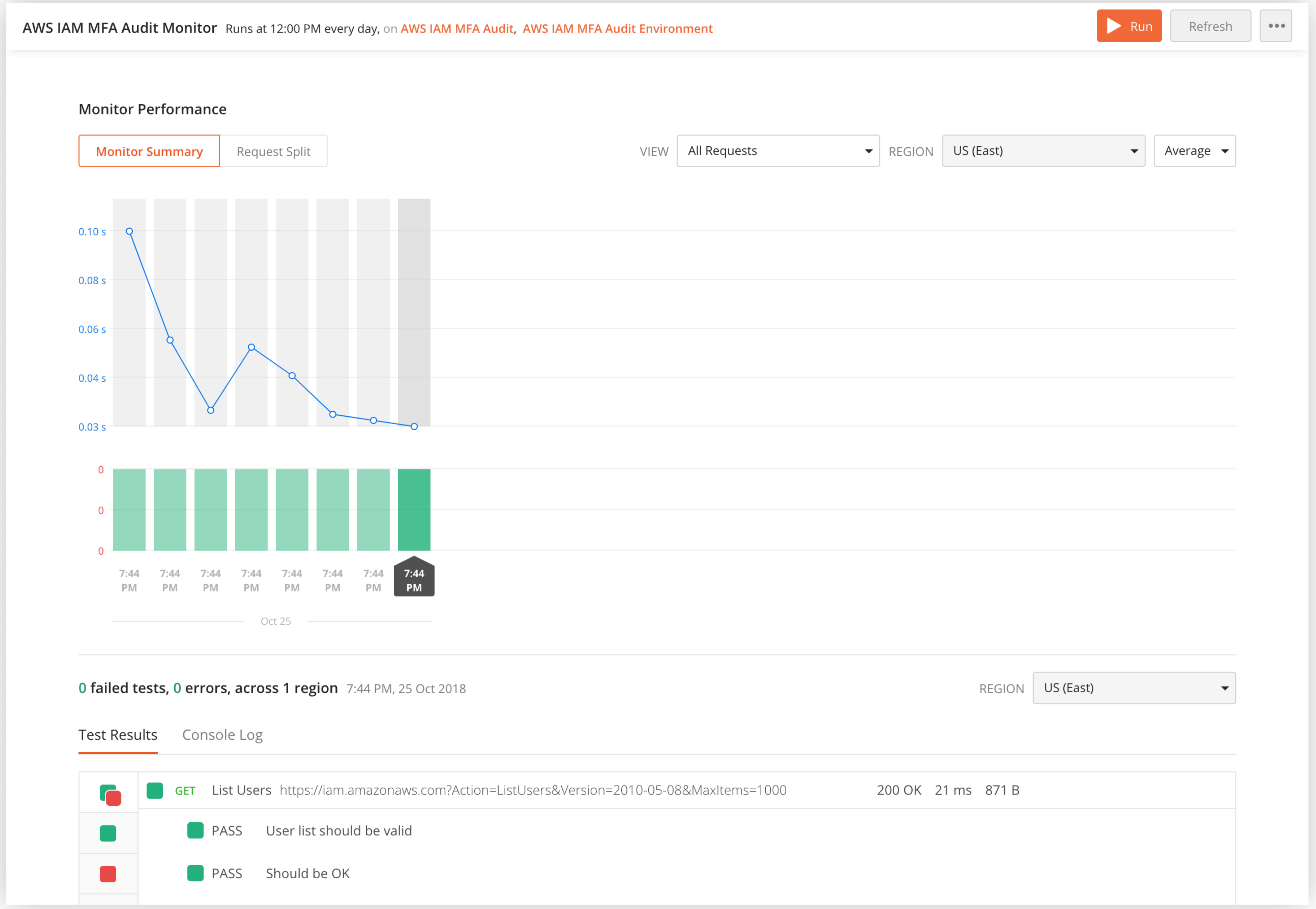
Platform

Security considerations and processes for infrastructure.



Audits & Automation

- What to audit?
  - Developer access
  - Setup configuration
  - Creation of new resources
- We use collection runs to create new resources reliably.
- Postman Monitors to perform periodic audits of our services.



AWS IAM MFA Audit Monitor Runs at 12:00 PM every day, on AWS IAM MFA Audit, AWS IAM MFA Audit Environment

Monitor Performance

VIEW All Requests REGION US (East) Average

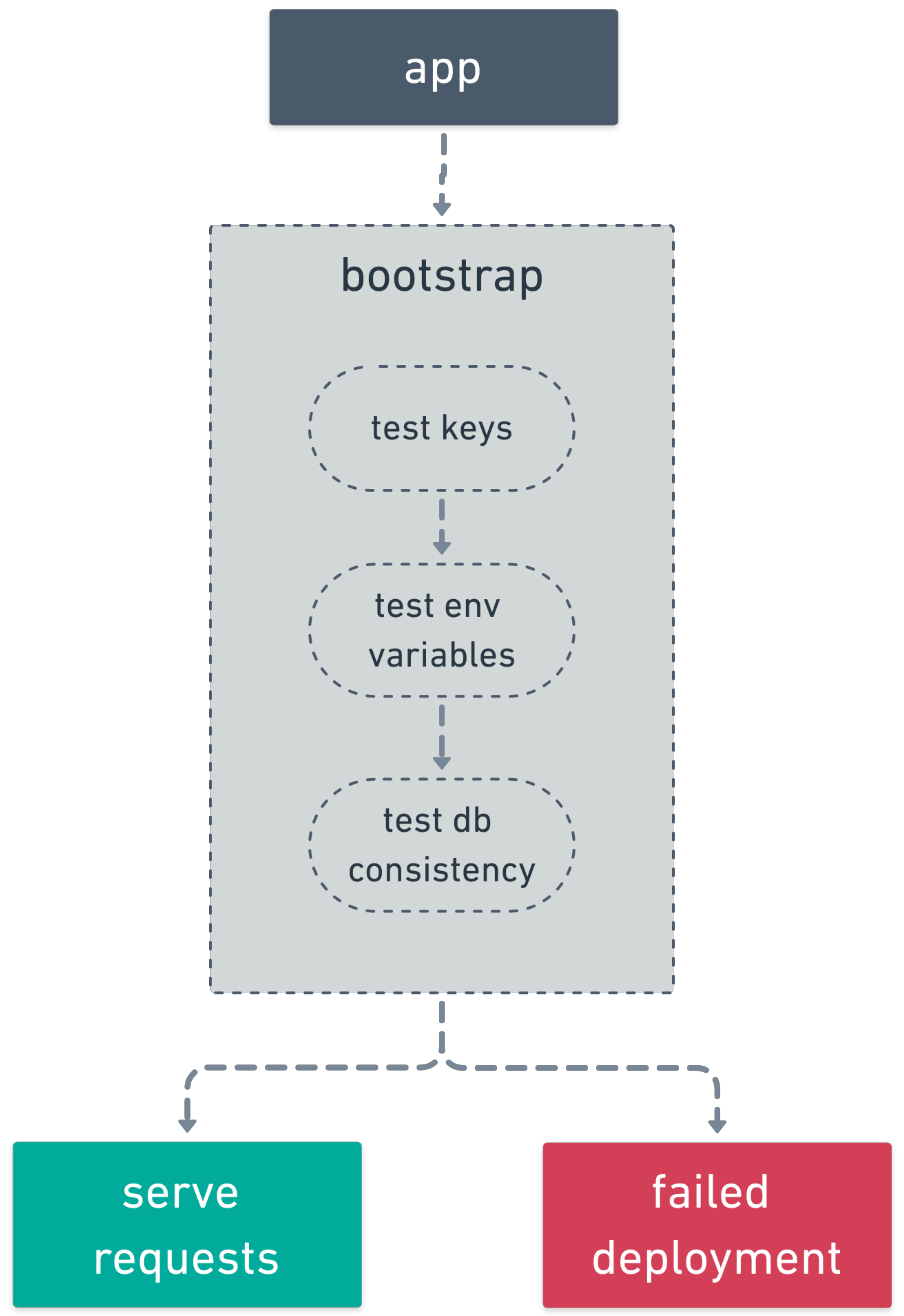
0 failed tests, 0 errors, across 1 region 7:44 PM, 25 Oct 2018 REGION US (East)

```
GET  List Users  https://iam.amazonaws.com?Action=ListUsers&Version=2010-05-08&MaxItems=1000   200 OK  21 ms  871 B
     PASS  User list should be valid
     PASS  Should be OK
```

Audit with Collection

Health Check

- Verify critical config based on environments.
- **Prevent deployment** if there is something obviously wrong. Ex. *leaking private keys*.
- This is a **safety net** and not a testing mechanism.

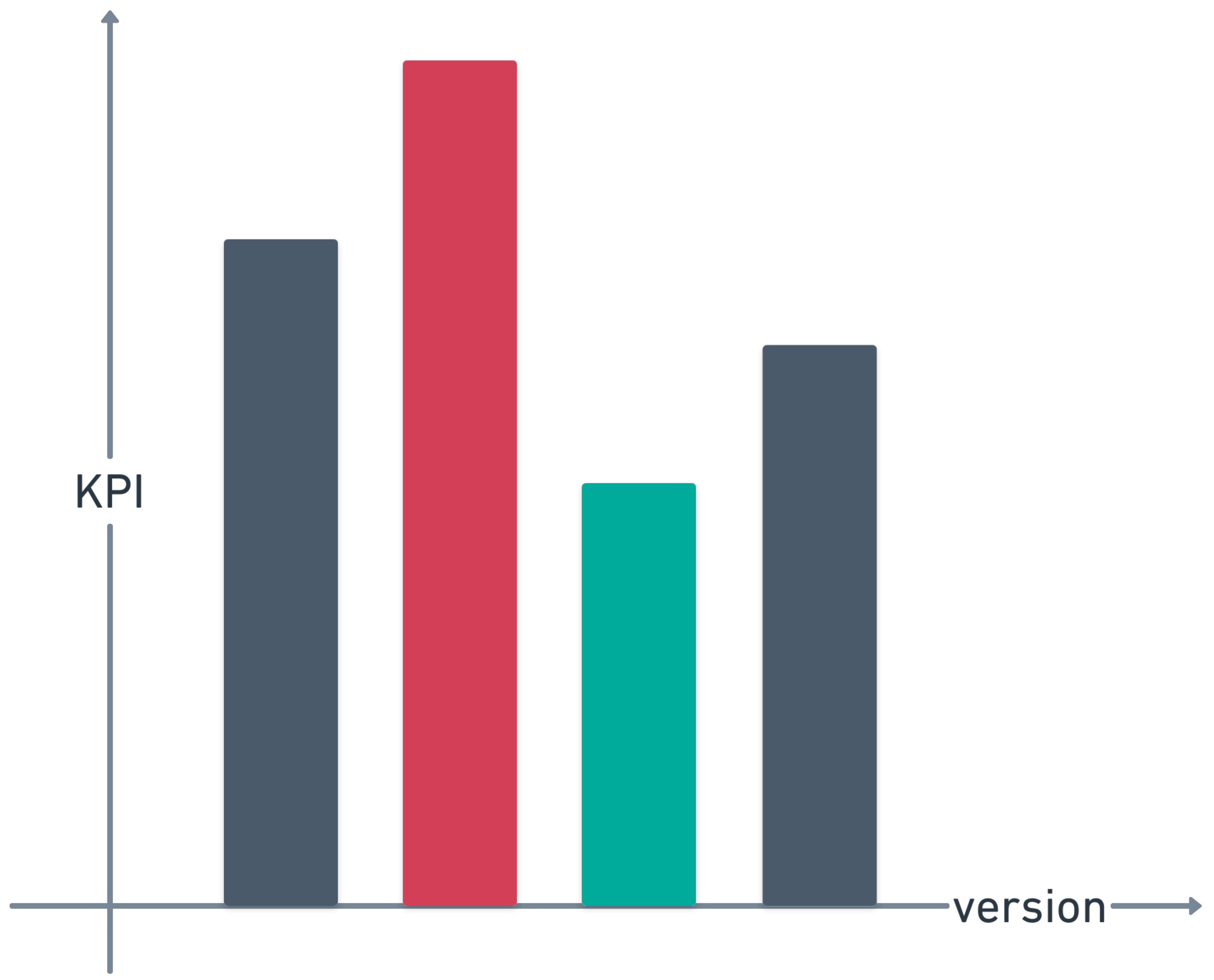


SDLC

Processes involved to ensure security

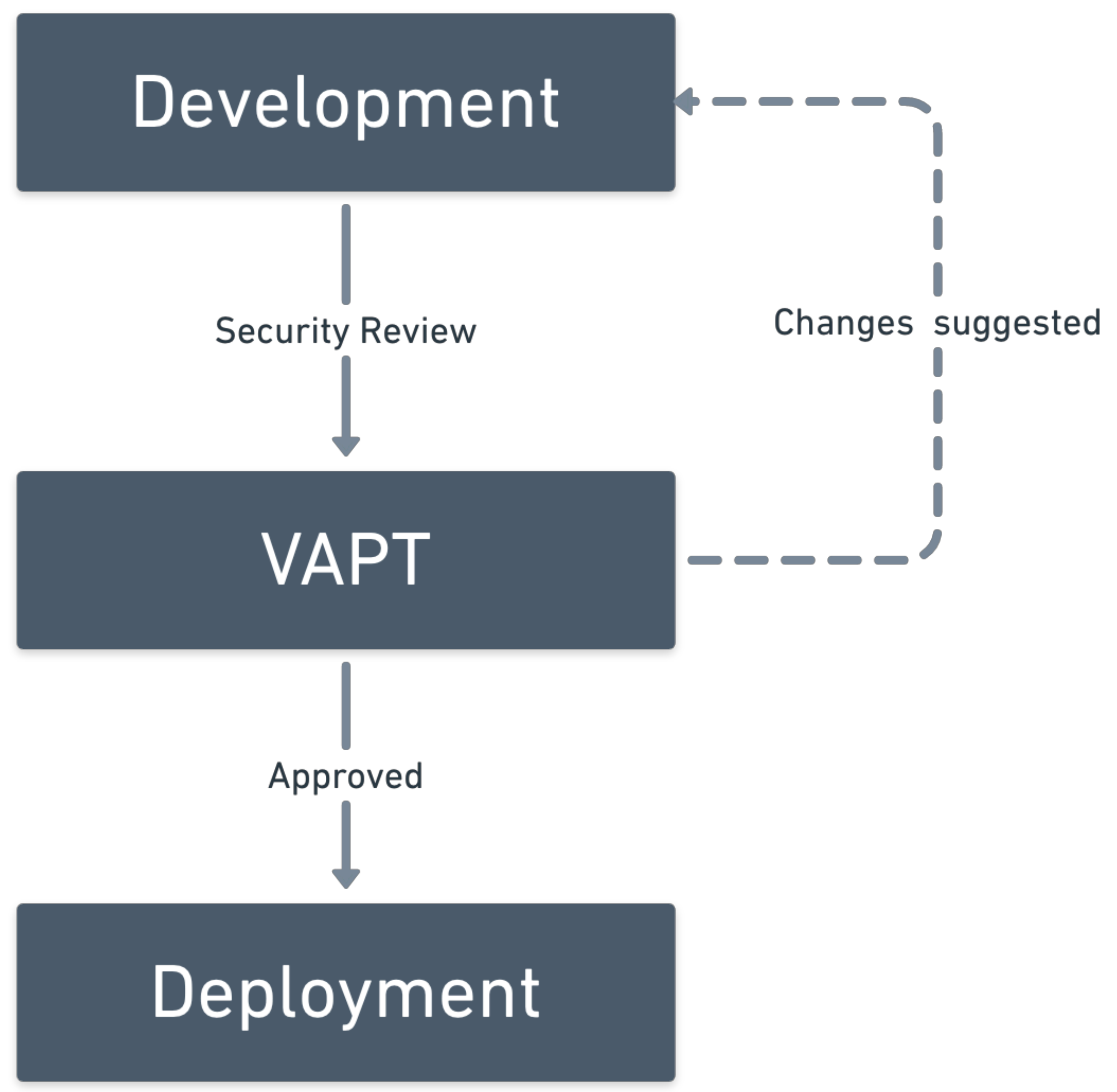
Security KPIs

- Vulnerability categorization by **CVSS** scores.
- Vulnerability **regression**.
- Time to resolve - **SLA**.
- External security reports - user identified, Hacker One, etc.



VAPT

- Post-development step to assess the security of a software release.
- Black box and white box testing of services.
- Automation of security processes.



Outro

Revisiting Security Parameters

Confidentiality	Integrity	Availability
• Validation • PoLP • Log scrubbing	• Request tagging • Access control (IDOR) • Content security (HTTPS, SRI, CSP, etc.)	• Short critical path • Platform audits • Healthcheck

Key Takeaways

- Security considerations while building a BFF / public API.
- Building a secure API is a gradual process.
- Security is a part of development process.

Thank you

Assets

https://github.com/ankit-m/talks/tree/master/jsfoo-2018