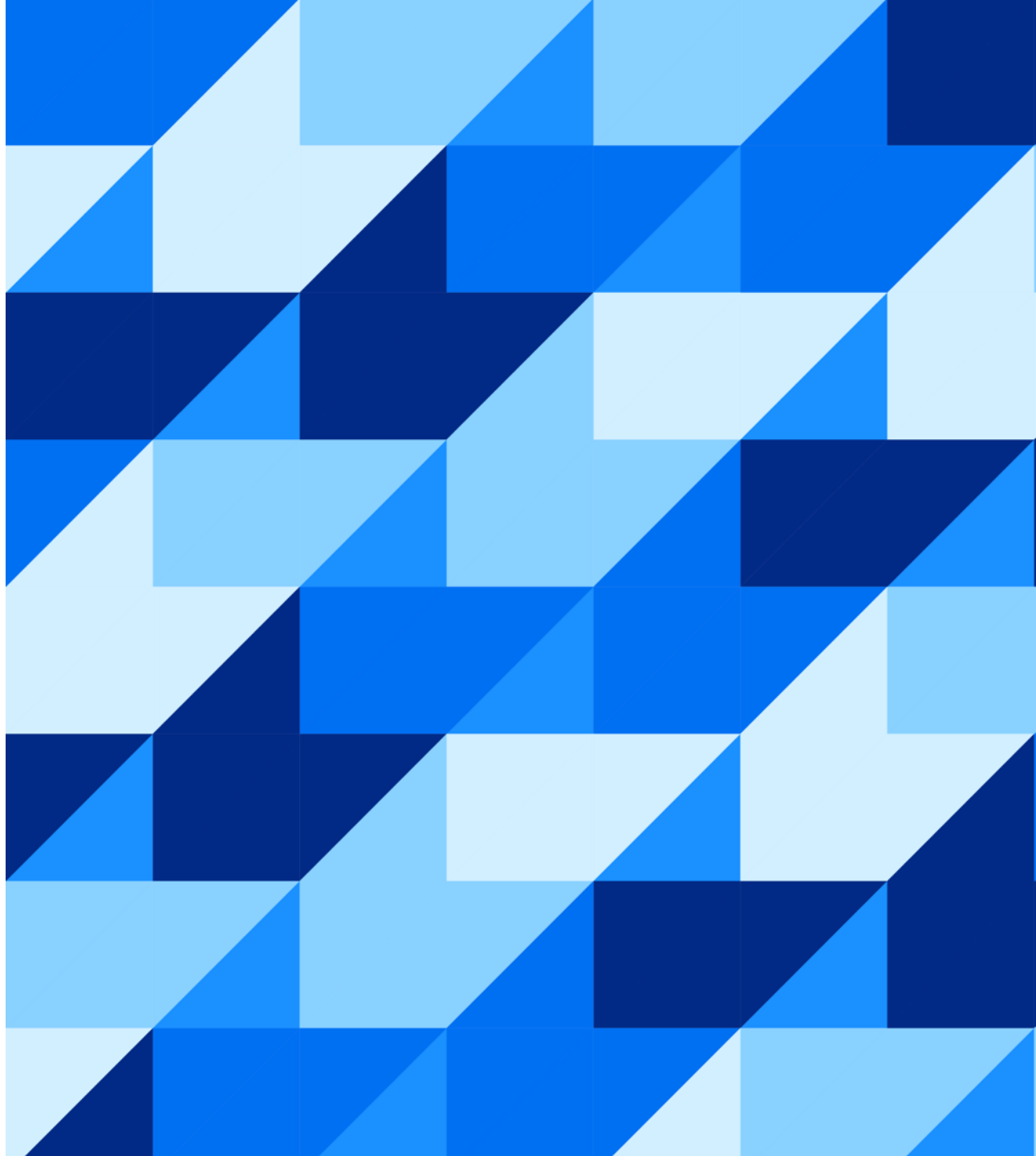
SAP Signavio

To Microservices and Beyond

Richard Všianský; Nodar Pylypyshak, SAP
November 07, 2024

Public

MUNI



Agenda

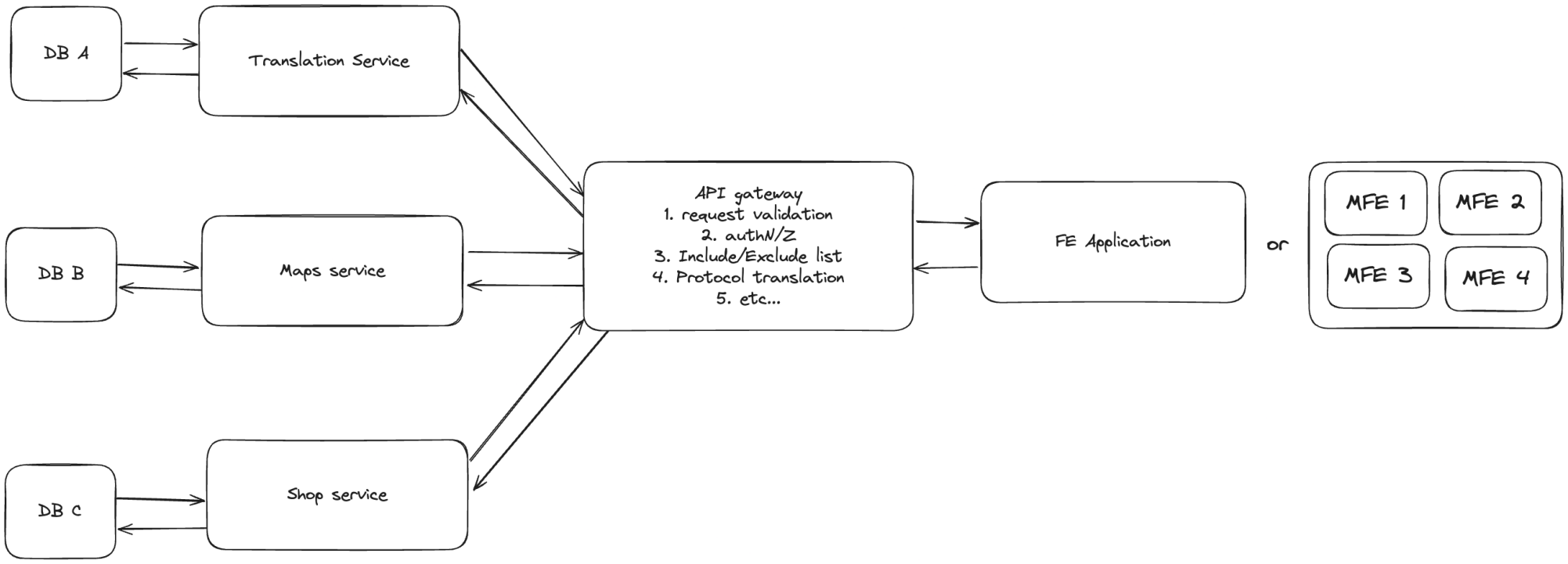
1. Microservices
2. Microservices vs Monolith
3. Microservices disadvantages
4. Microservices communication patterns
5. Logging
6. Testing
7. Scalable deployment

Public

Microservices introduction

Microservice architecture allows developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP API.

It allows to distribute responsibilities of different parts of a product across multiple services



Public

Microservices introduction

Note: Monolith implementation might require communication with 3rd party services (0Auth for example), but it does not make it microservice architecture

Public

Little bit of history

1960s - Large-Scale Software Challenges: Issues with large-scale software development began surfacing, leading to a need for new design methods and structures.

1970s - Rise of Software Design Research

1980s - Early mentions of software architecture began

1990s - Foundation of Component-Based Software Engineering (CBSE)

Early 2000s - Service-Oriented Computing (SOC): aimed at managing the complexity of distributed systems and using message-passing for service communication.

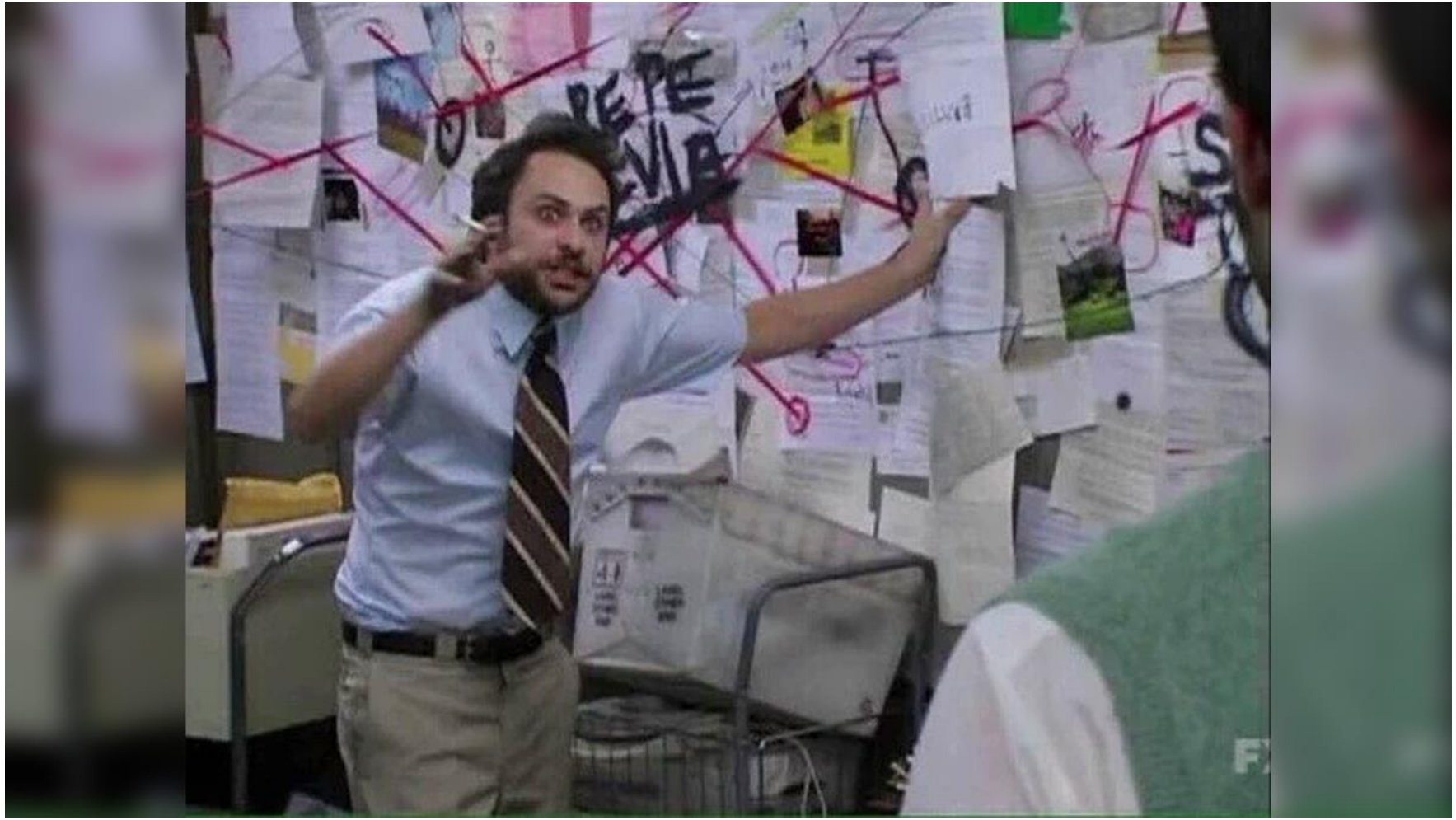
Today - Microservices Emergence: As a streamlined successor to SOA, microservices focused on creating simple, single-function services to reduce complexity.

Public

Comparison of microservice and monolith architectures

1. Smaller codebase
2. Microservices bring team the autonomy and its own lifecycle.
    1. you need to deploy multiple times a day? Good, you can do it
3. Microservice approach preferring that a team should own a product over its full lifetime.
    1. “you build, you run it”, development team takes full responsibility for the software in production.
4. Microservices bring clear domain boundaries (but of course, you can mess everything)
5. With microservices its easier to introduce new library or any other dependency (just make sure that library is compliant with company policies)

Microservices disadvantages



Meme: https://knowyourmeme.com/memes/pepe-silvia

Public

Microservices disadvantages

1. More complex maintenance
2. Increased security complexity
3. Data consistency and transaction
4. Local setup
5. Issue tracing becomes much more difficult with big amount of services

Public

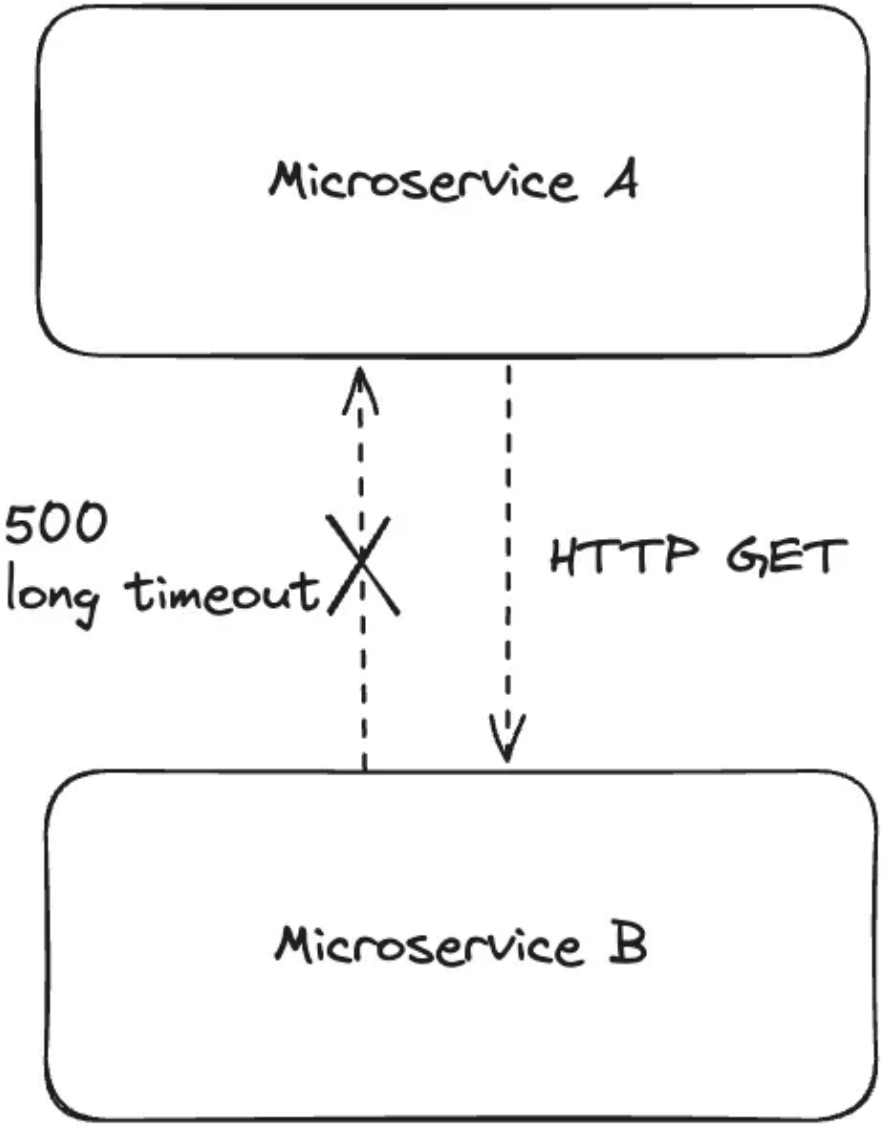
Things to consider

1. Design to handle failures
2. Monitoring and alerting
3. Communication

Public

Design to handle failures

A consequence of using microservices is that applications need to be designed so that they can tolerate the failure of other services. Any service call could fail due to unavailability, the client has to respond to this as gracefully as possible.



Public

Monitoring and alerting

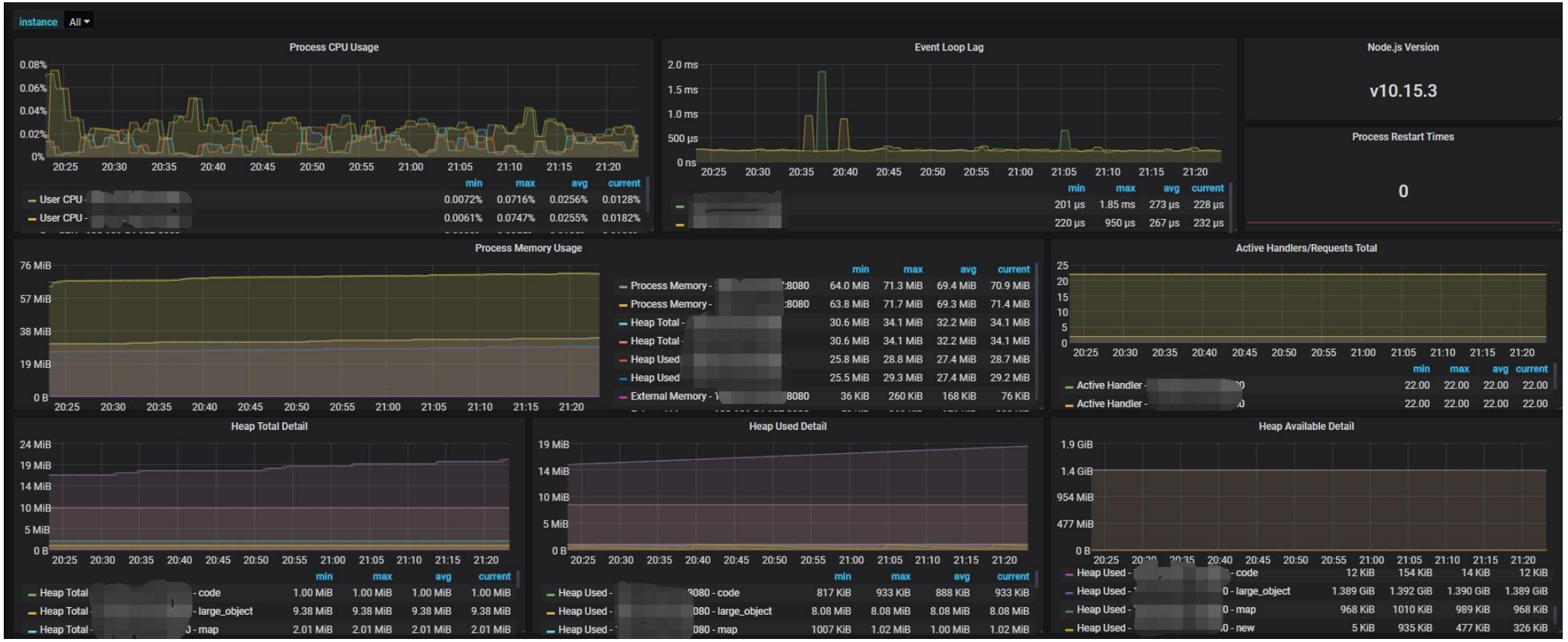
Since services can fail at any time, it's important to be able to detect the failures quickly by using:

1. Metrics: **Prometheus and Grafana**
2. Centralized logging: **Elasticsearch, Logstash and Kibana**
3. Error tracking: **Sentry**
4. Tracing: **Jaeger and Zipkin**

Grafana logo https://grafana.com/docs/grafana/latest/
Kibana logo https://www.elastic.co/kibana
Jaeger logo https://www.jaegertracing.io/
Sentry logo https://sentry.io/branding/

Public

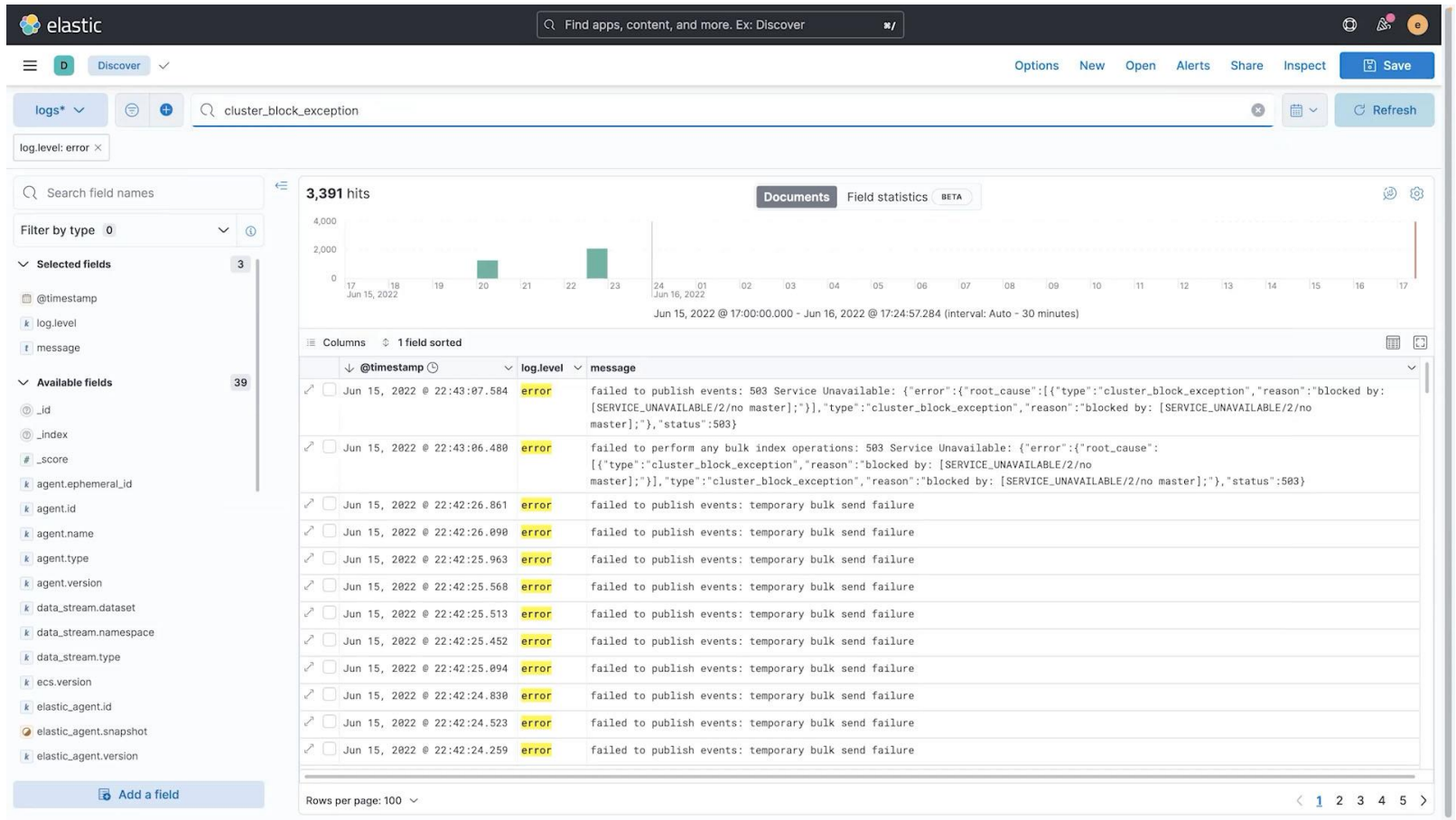
Grafana



Grafana example: https://grafana.com/grafana/dashboards/11159-nodejs-application-dashboard/

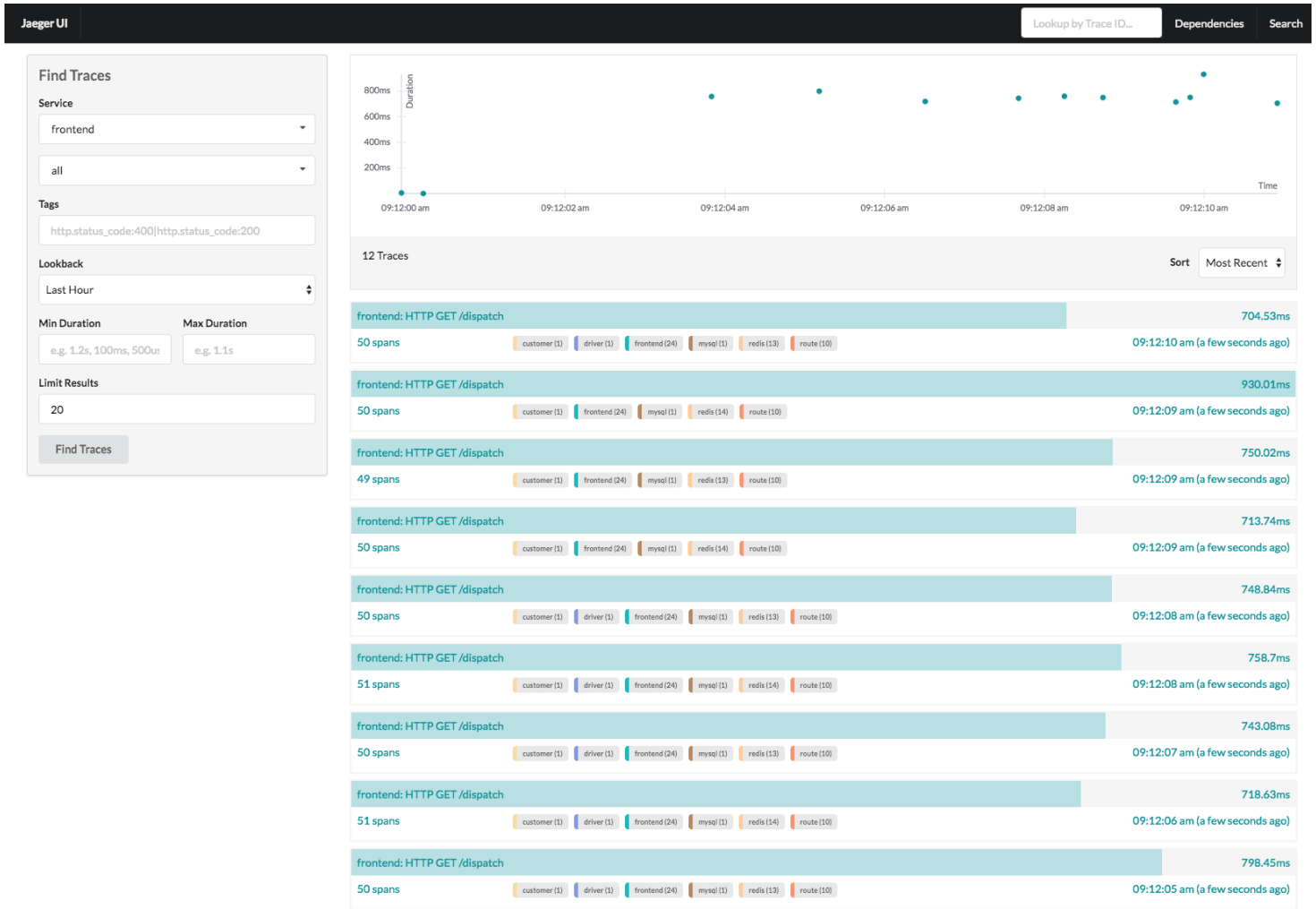
Public

Kibana



Kibana example https://www.elastic.co/kibana

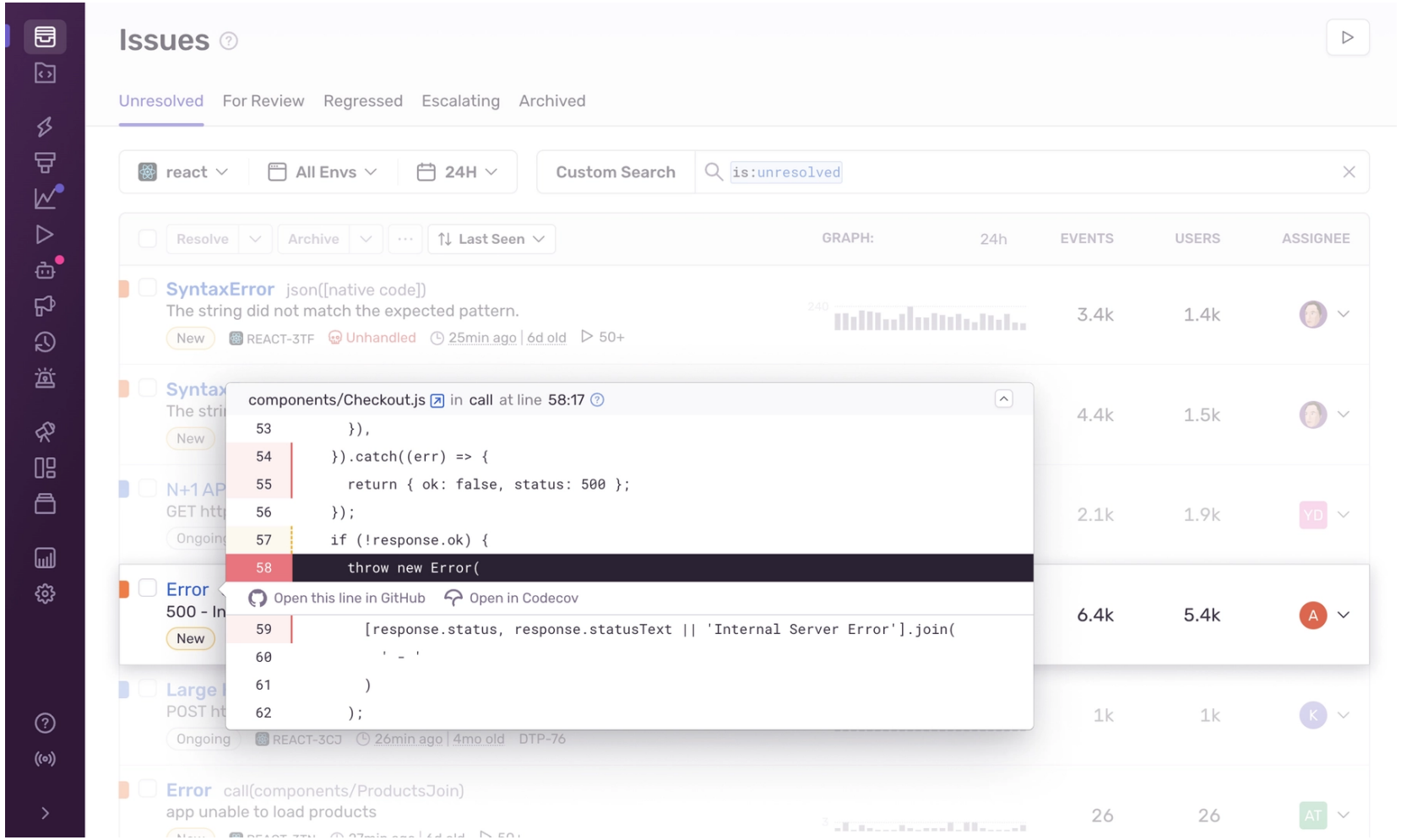
Jaeger tracing



Jaeger example https://www.jaegertracing.io/docs/1.62/

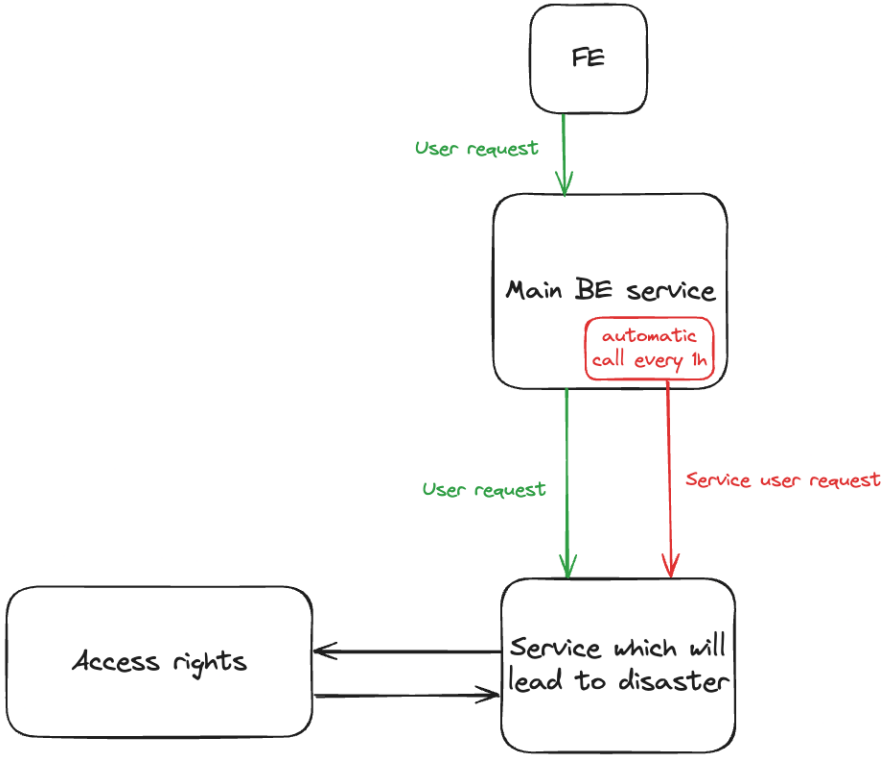
Public

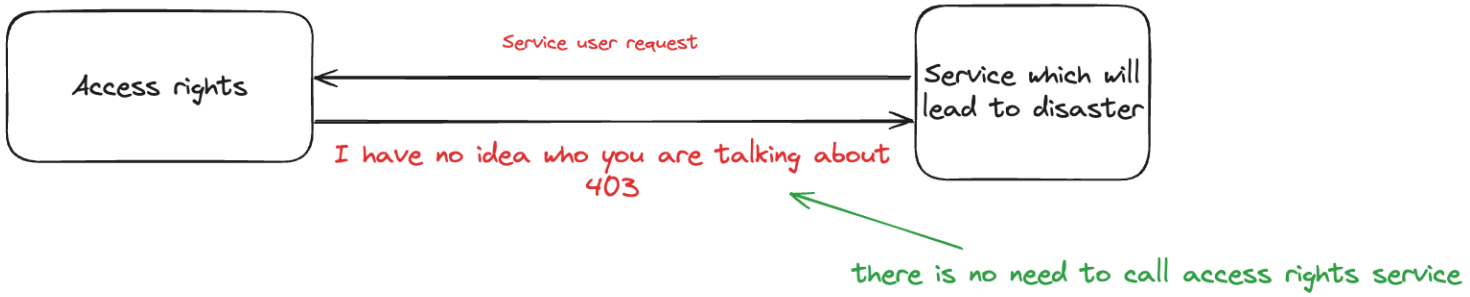
Sentry

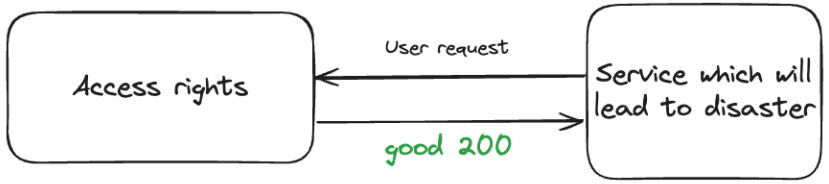


Sentry example https://sentry.io/welcome/

Do you really need all of this? Let us tell you a short story







Public

Communication types

1. Synchronous Communication – direct communication, client waits for server to respond
2. Asynchronous Communication – client doesn’t wait for the response

Public

How microservices communicate between each other?

1. HTTP/HTTPS
2. Messaging queues (RabbitMQ, Apache Kafka)
    1. Send message to a queue, specific service will consume it
    2. Example: **Order Service sends a message to a queue and Shipping Service receives it**
3. RPC (gRPC) – remove procedure calls
    1. Allows to call another service endpoint as it is local method
4. Event streaming
    1. Populate message to all services
    2. Example: **order was made so notification service, shipping service and other services got a message**

Communication and central logging demo

Public

Do you really need microservices?

Well, lets start from one important thing, what microservices bring in general?

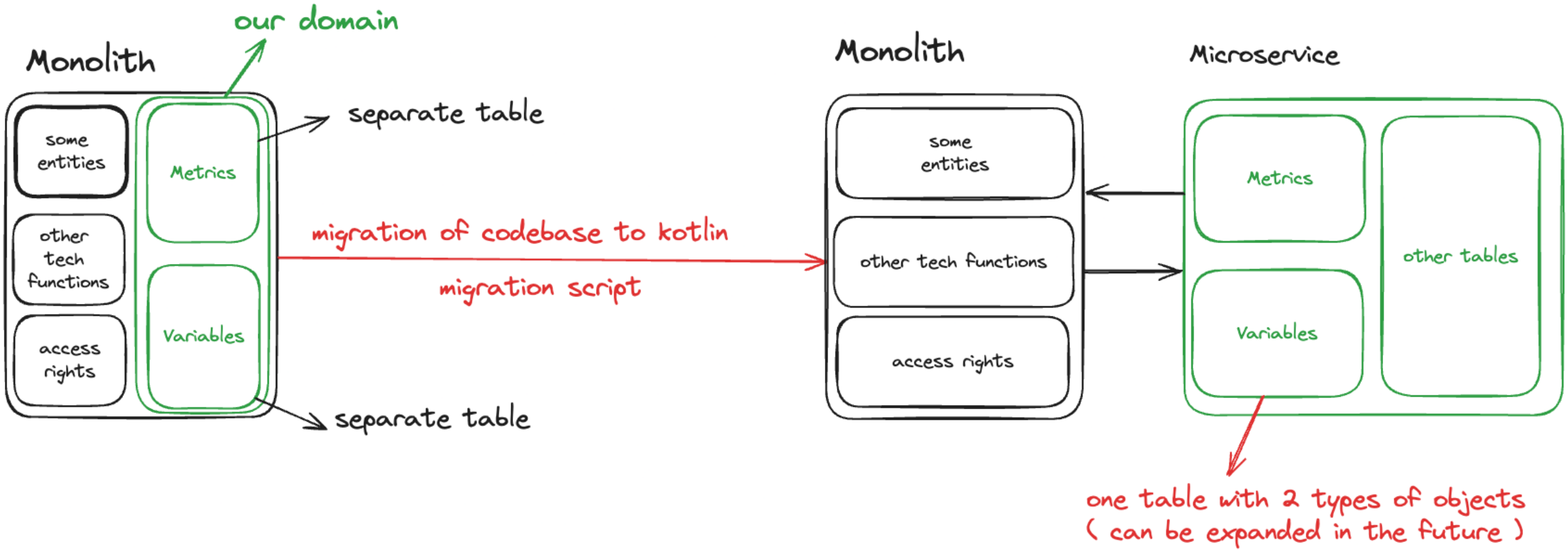
“Microservices are more about the organisational structure than about organisation of your code” – DHH, creator of Ruby on Rails

1. “Don’t have more microservices than users”.
   - Sometimes you need to start small just one repo and nothing else
2. How much time it will take to split a monolith into smaller services?
3. What benefit it will bring you in the end?

Public

Our experience

Our part of product started to grow and we decided to split

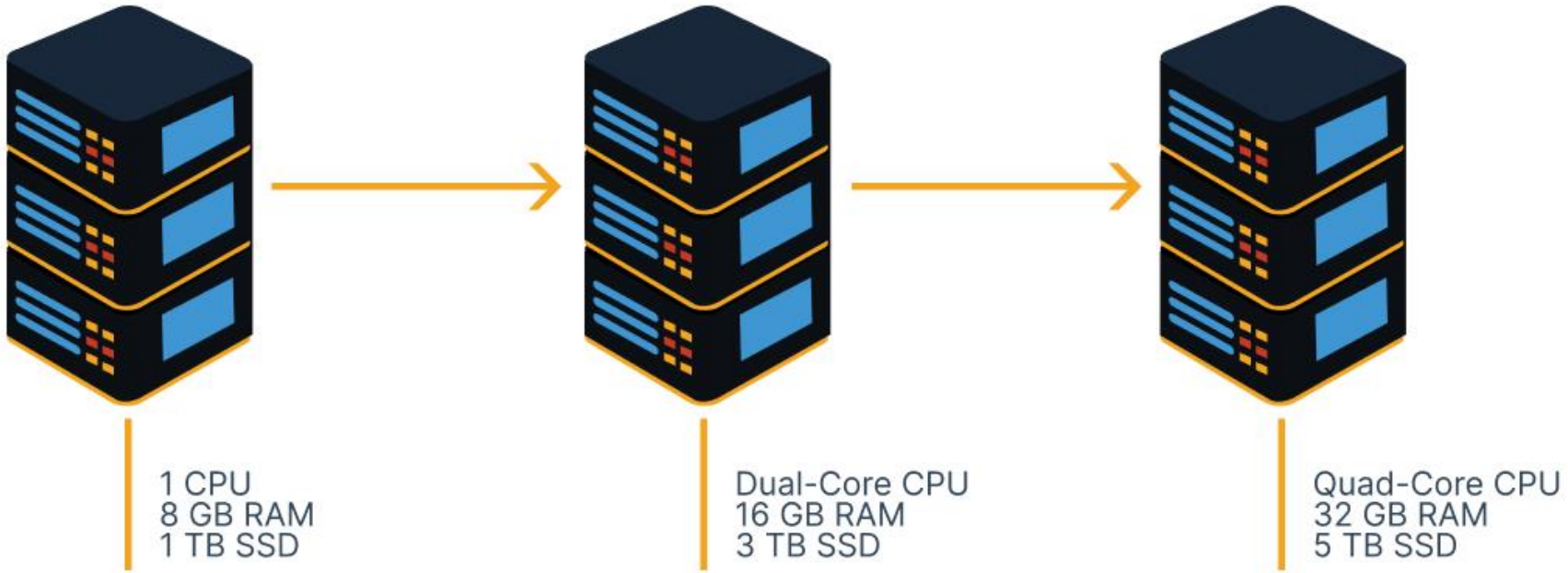


Things to consider before migration to microservices

1. Do you need to transform existing data before migration?
2. How you will migrate the data to a new database
   - You need to create a migration script to migrate all data
   - Make sure that you have **proper logging and metrics** during the whole process to not loose any data
   - You should be able to rollback the whole migration in case or error
3. You might need to run old implementation of monolith together with a new microservice to have a synced data (still, it’s a very tricky part)

Vertical scaling

Vertical scaling – upgrading hardware (CPU, RAM, SSD, etc..) for existing services/instances

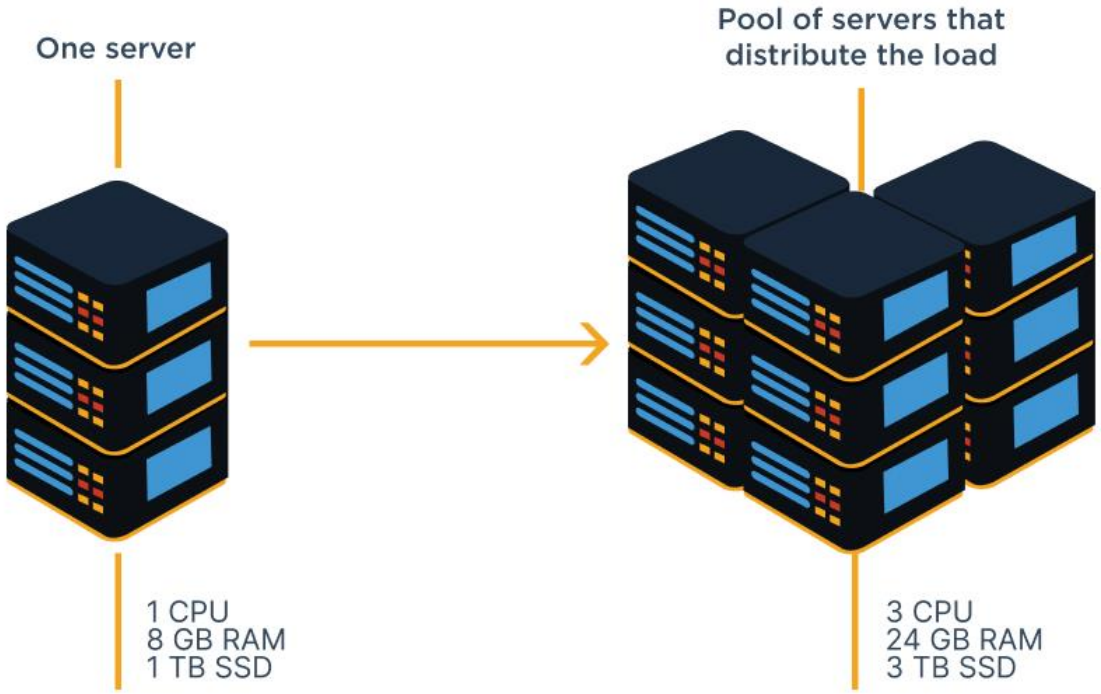


Picture from https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/

Public

Horizontal scaling

Horizontal scaling – increase the number of instances (buying more servers or increasing number of virtual machines)



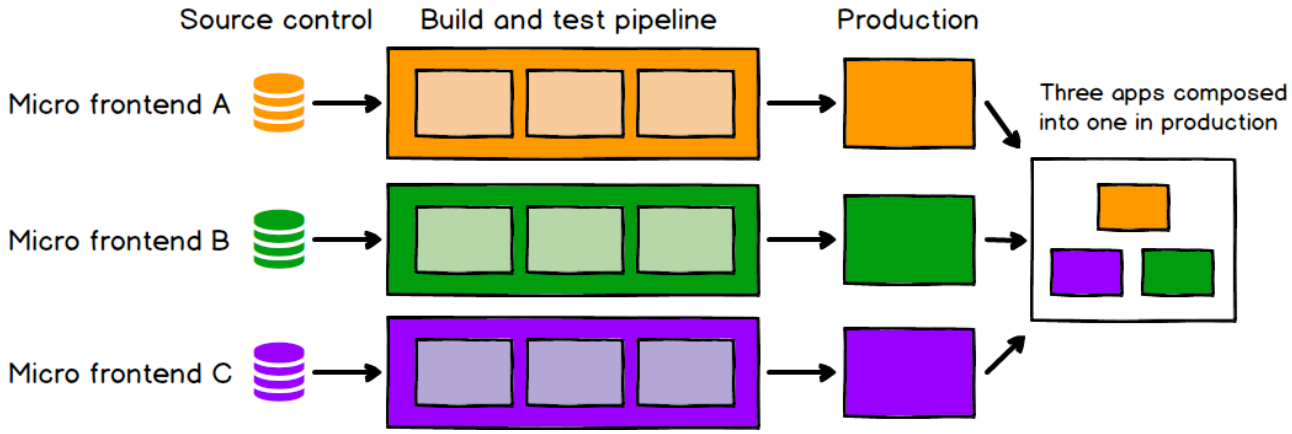
Picture from https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/

Public

Key points of vertical and horizontal scaling

1. Limitations
    1. Vertical scaling is limited
    2. For horizontal scaling you can add as many machines or virtual machines as you want
2. Costs
    1. Vertical scaling is cheaper as you need to upgrade just one component
    2. Horizontal scaling has high costs initially
3. Complexity of maintenance
    1. Vertical scaling has low complexity
    2. Horizontal scaling is more difficult to maintain
4. Failure resilience
    1. Horizontal scaling has high resilience, because other servers will provide backup
    2. Vertical has low resilience because there is only one point for access

Microfrontends



C. Jackson, https://martinfowler.com/articles/micro-frontends.html

- An approach to bring Microservices architecture to Frontend side
- Microservice team usually develop also a microfrontend application
- Teams can use different frameworks, not blocked by others, code is separated
- Enable **independent releases**
- One client (shell) consists of several UI microfrontend applications
- Downsides
  - Using different frameworks is not scalable (sharing components, knowledge, programmers)
  - Frontend parts usually interact a lot with each other
  - Serving different applications can increase application size

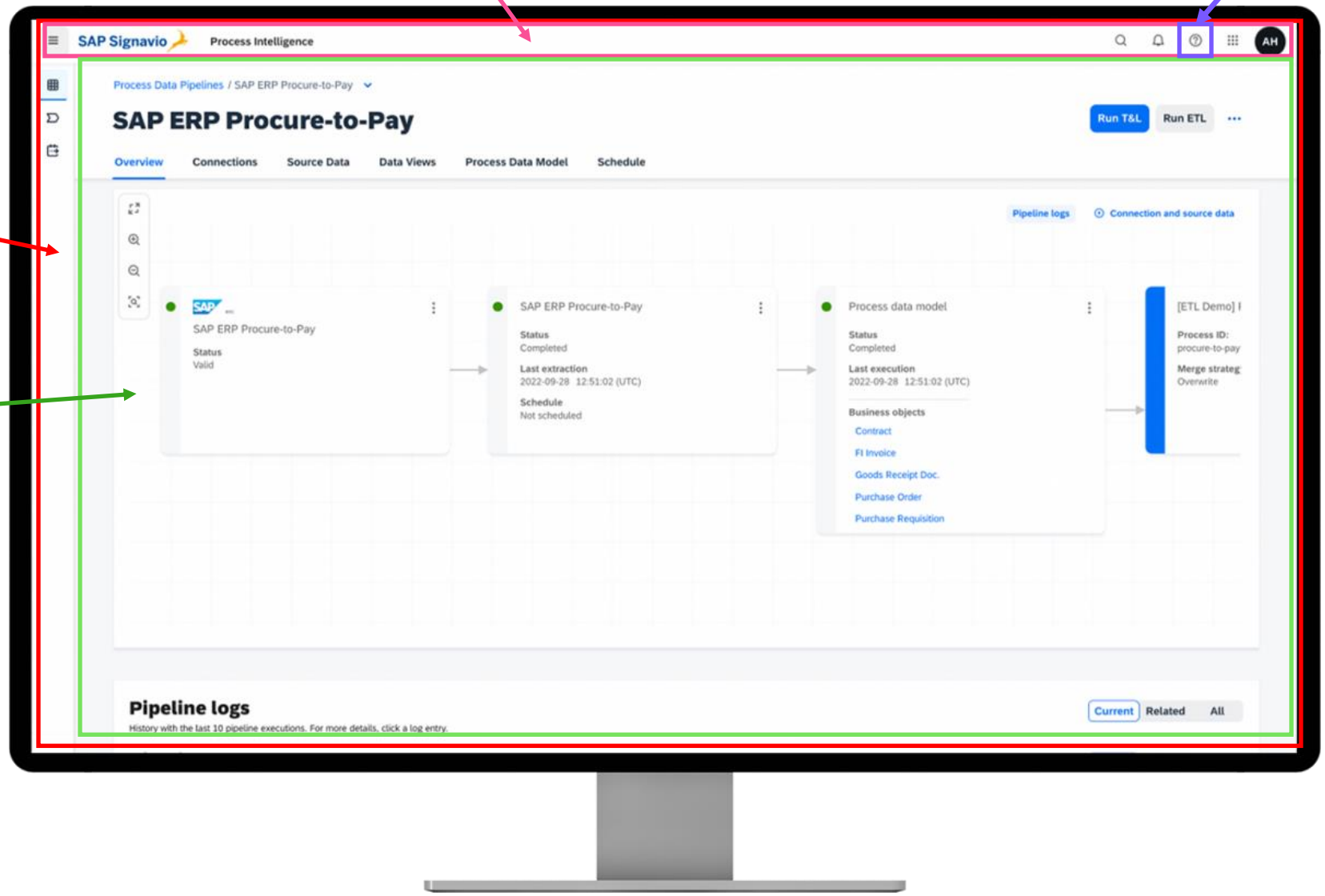
Public

Header

Notifications

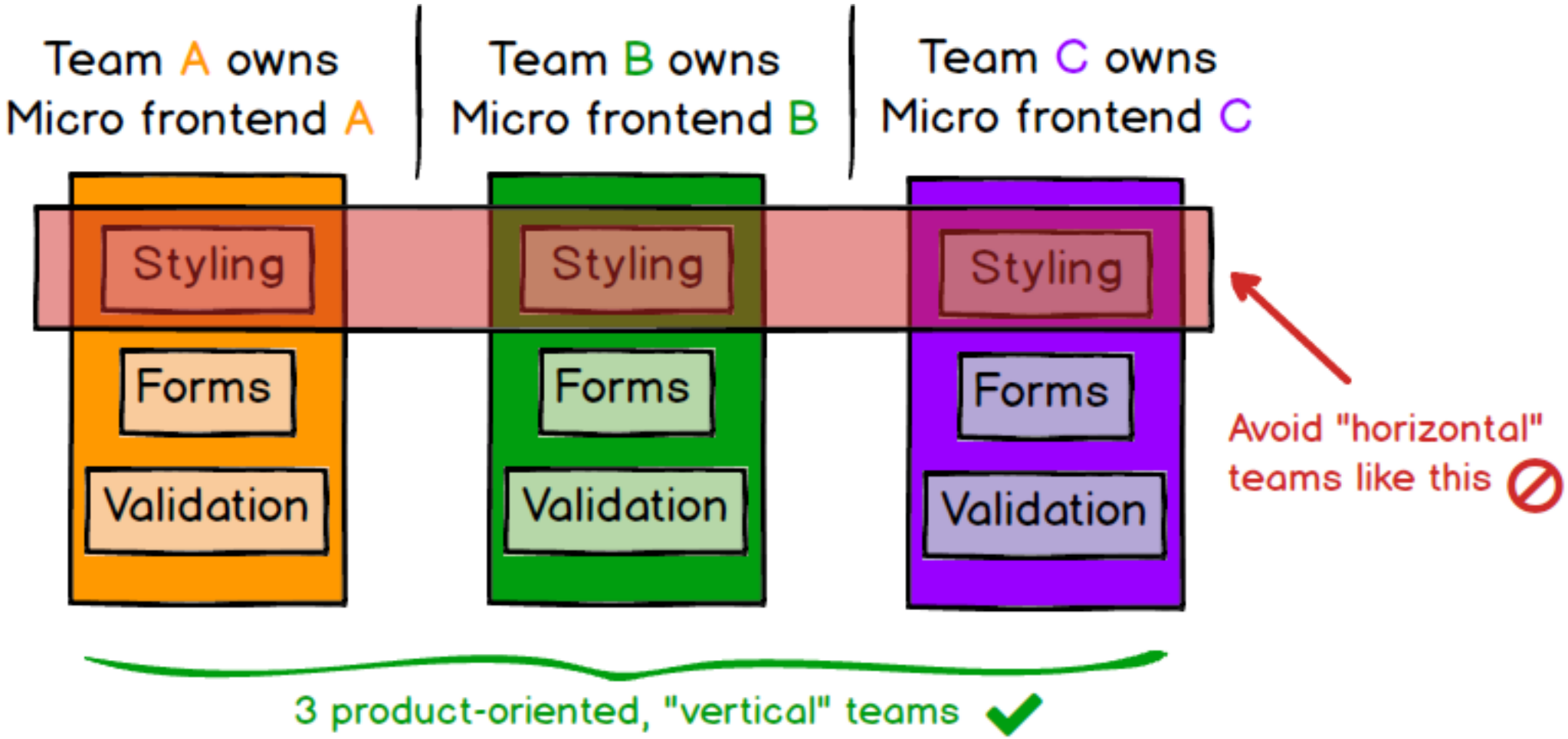
Client
(host application)

Dashboard



* https://www.signavio.com/products/process-intelligence/, Illustrative image

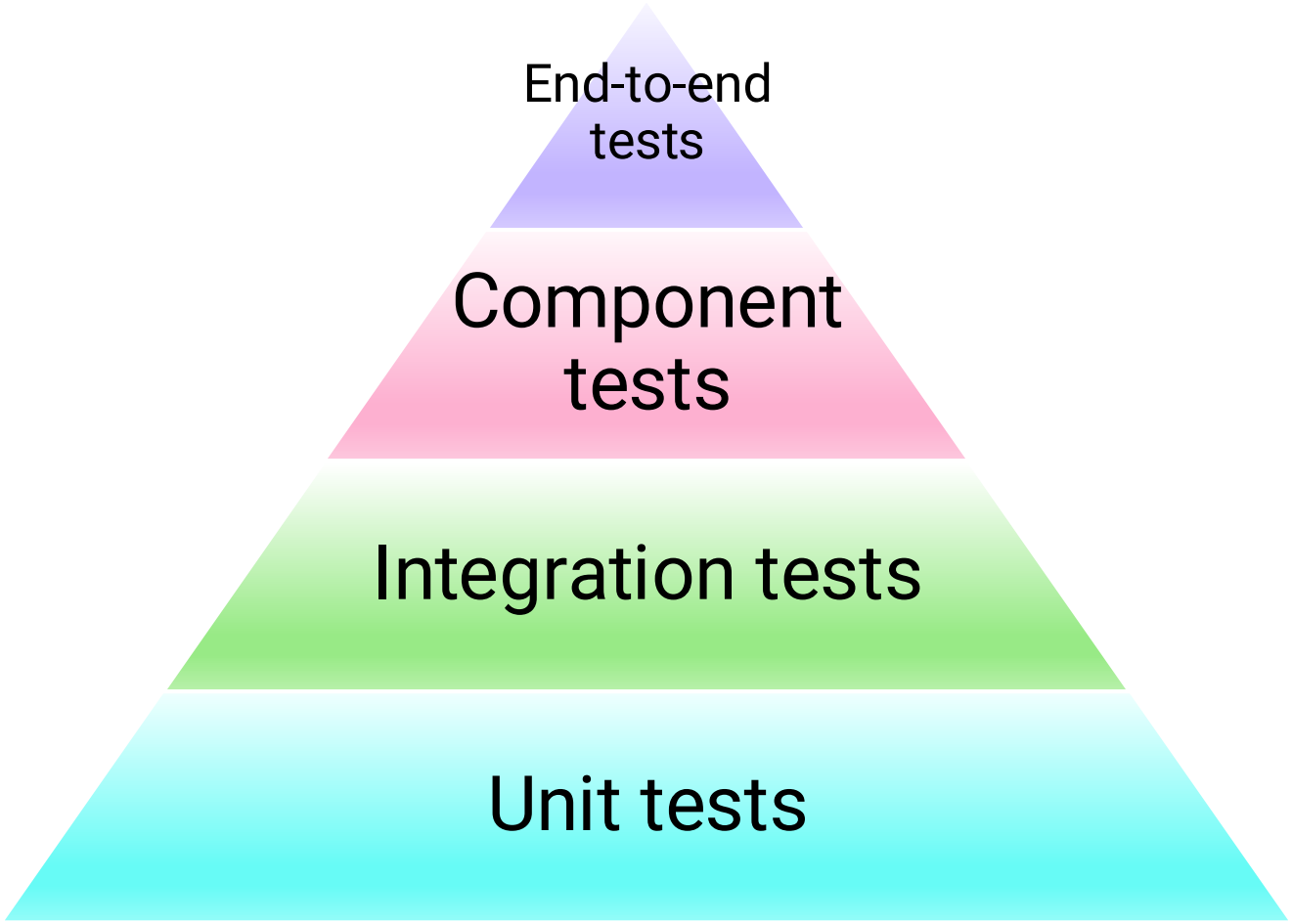
Public



https://martinfowler.com/articles/micro-frontends.html

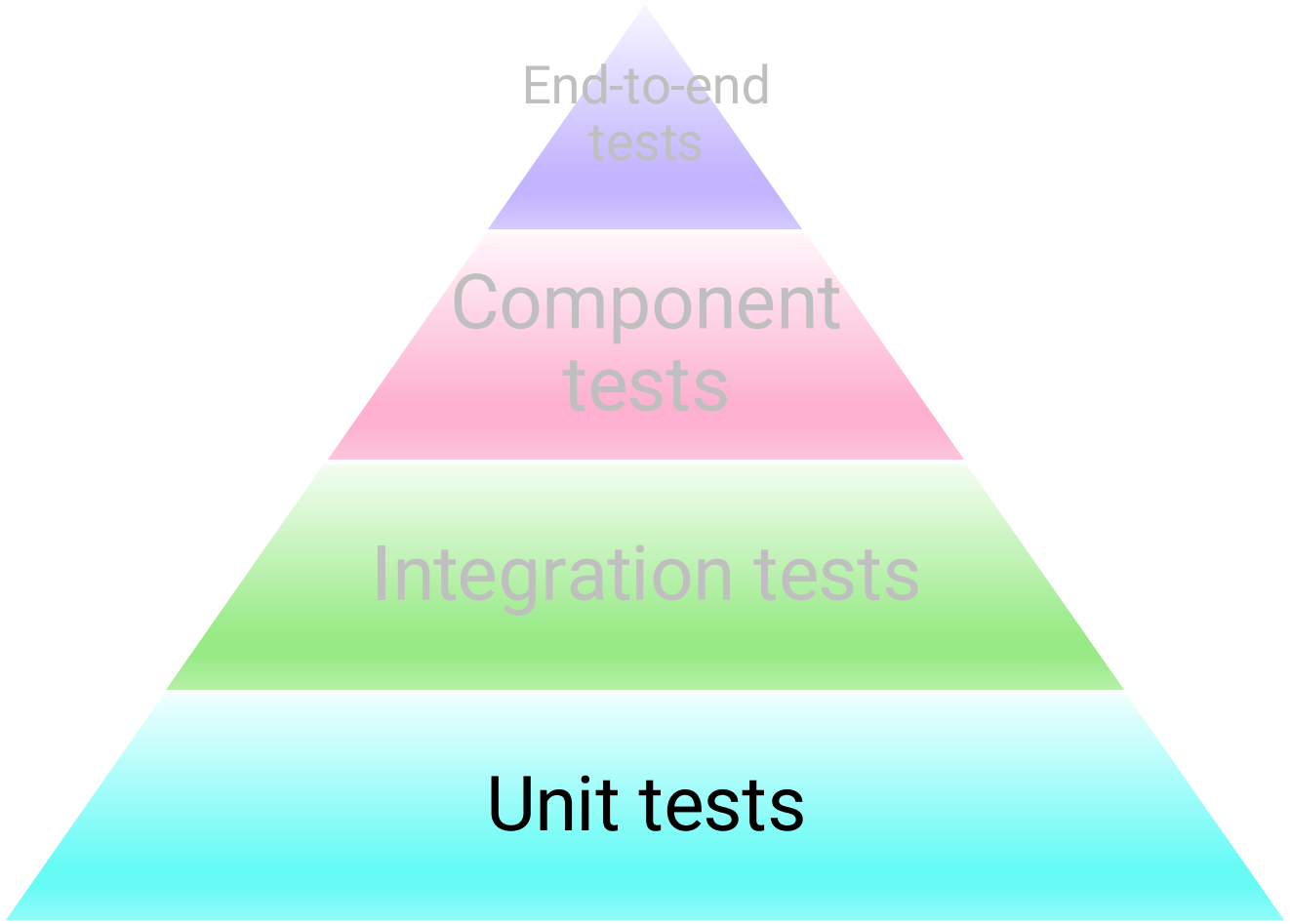
Public

Testing



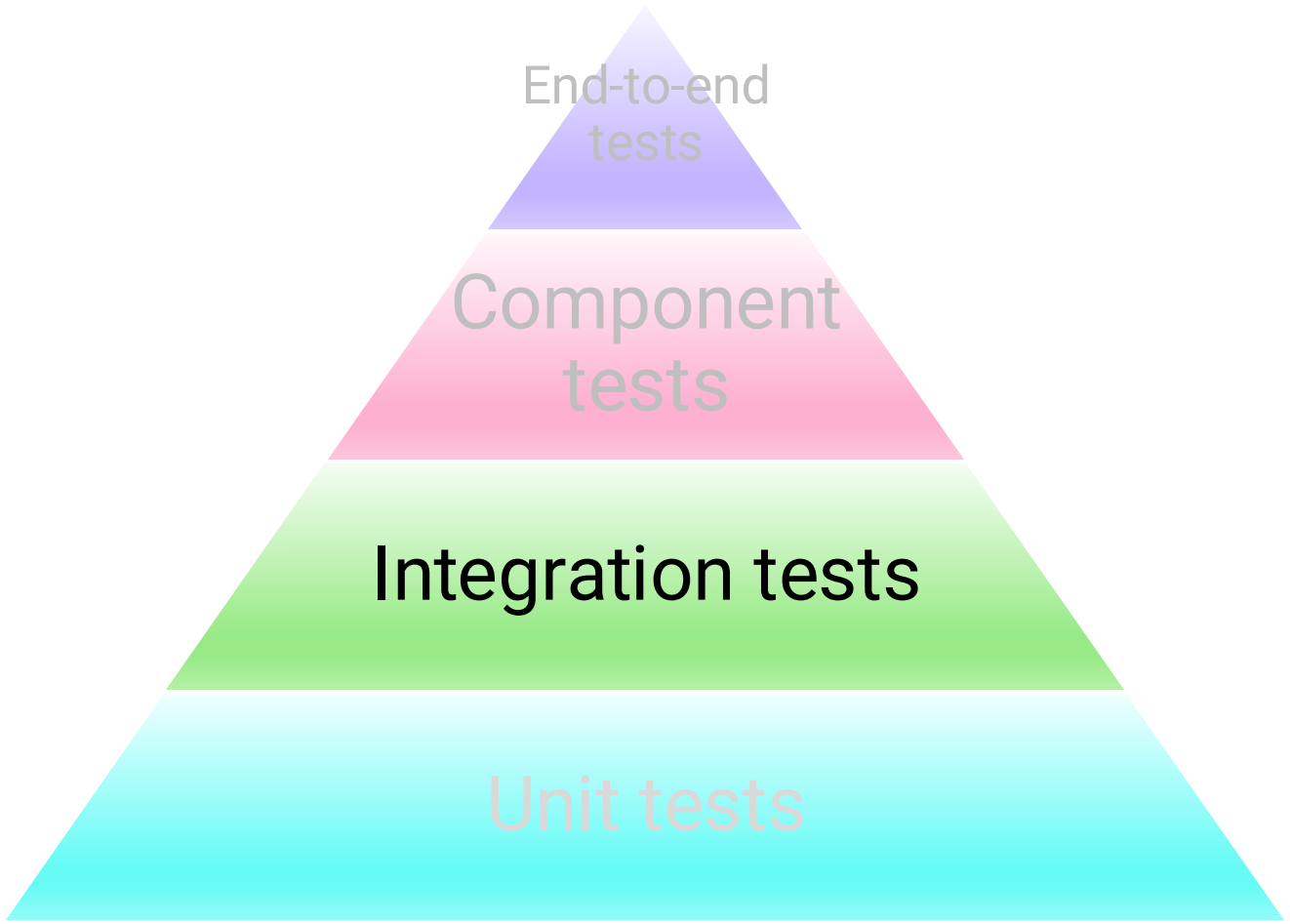
Public

Testing



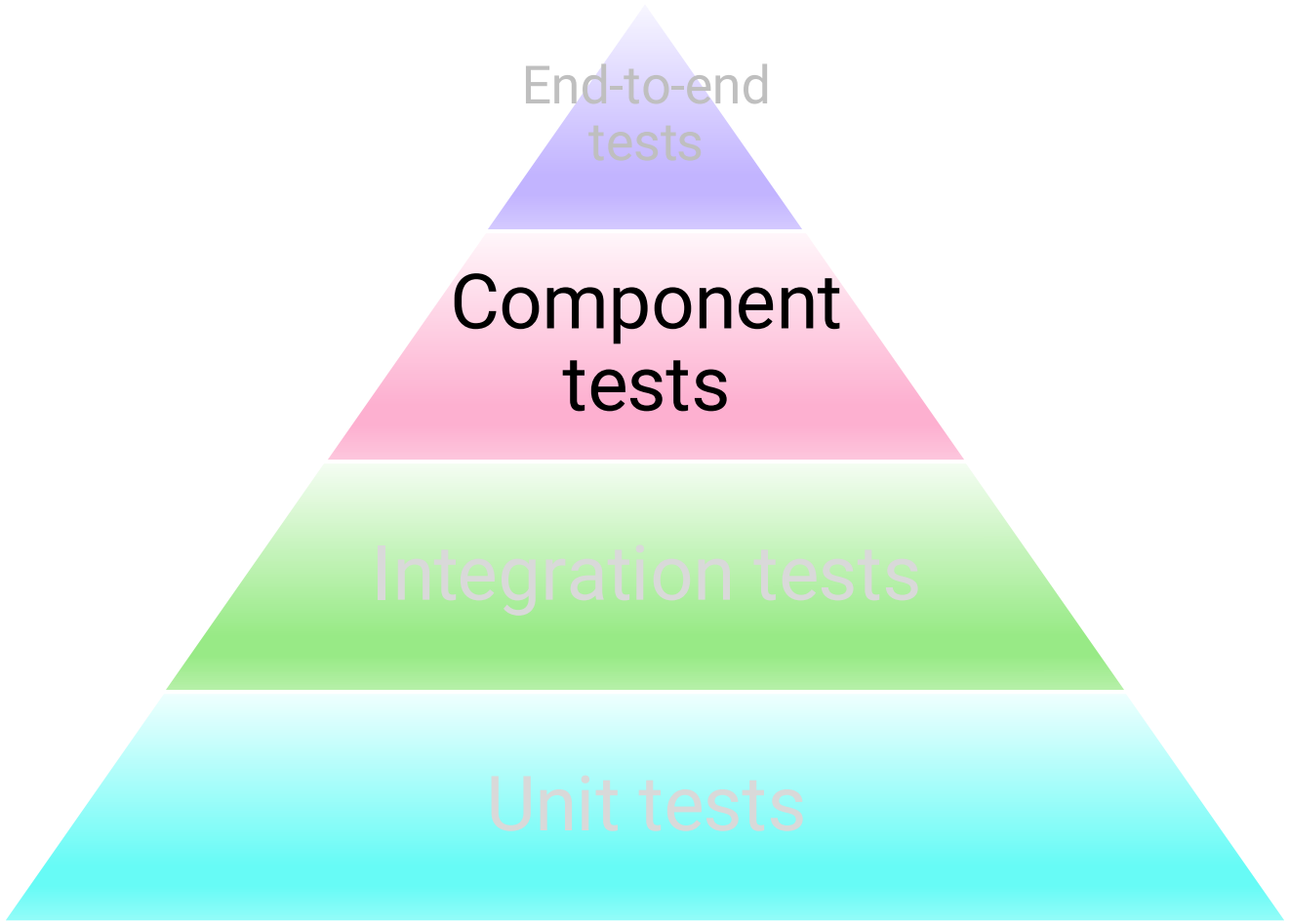
- Usually on level of class or a group of related classes
- Tests expected behaviour
- Can serve as a documentation, explain what a class/function does
- Consider costs of maintenance for having a larger number of tests

Testing



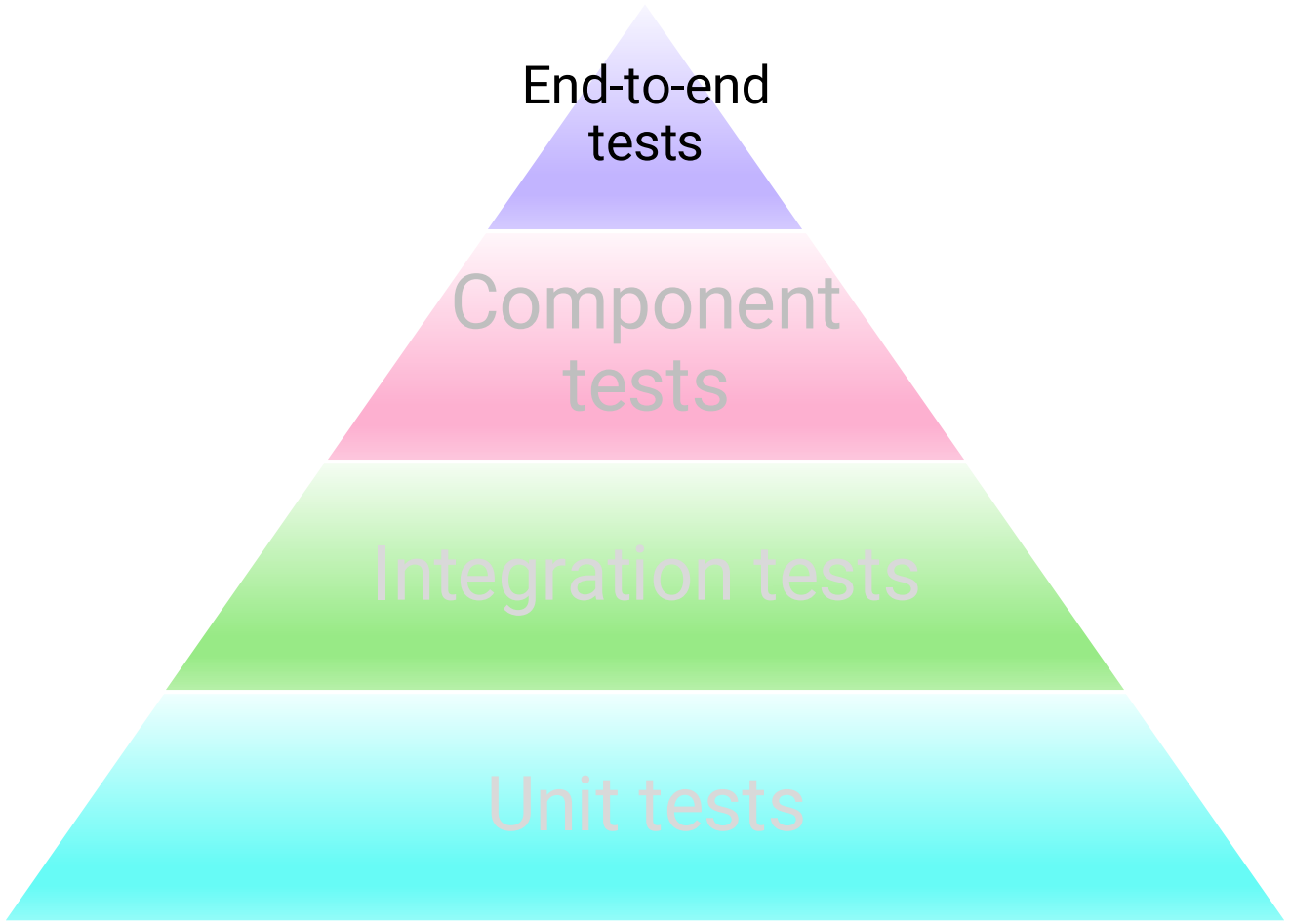
- To verify the **communication** paths and interactions between components
- On level of service layers
- Example - test between a service, a data store and a cache
- Use unit testing and contract testing to validate both sides of the communication

Testing



- A component is well-encapsulated, coherent and independently replaceable part of a system = a service itself
- Acceptance tests
- Provides controlled testing behaviour
- Usage of network interaction / in-memory doubles
    - Network interaction – more reliable
    - In-memory doubles – faster
- A shim = API/network interceptor

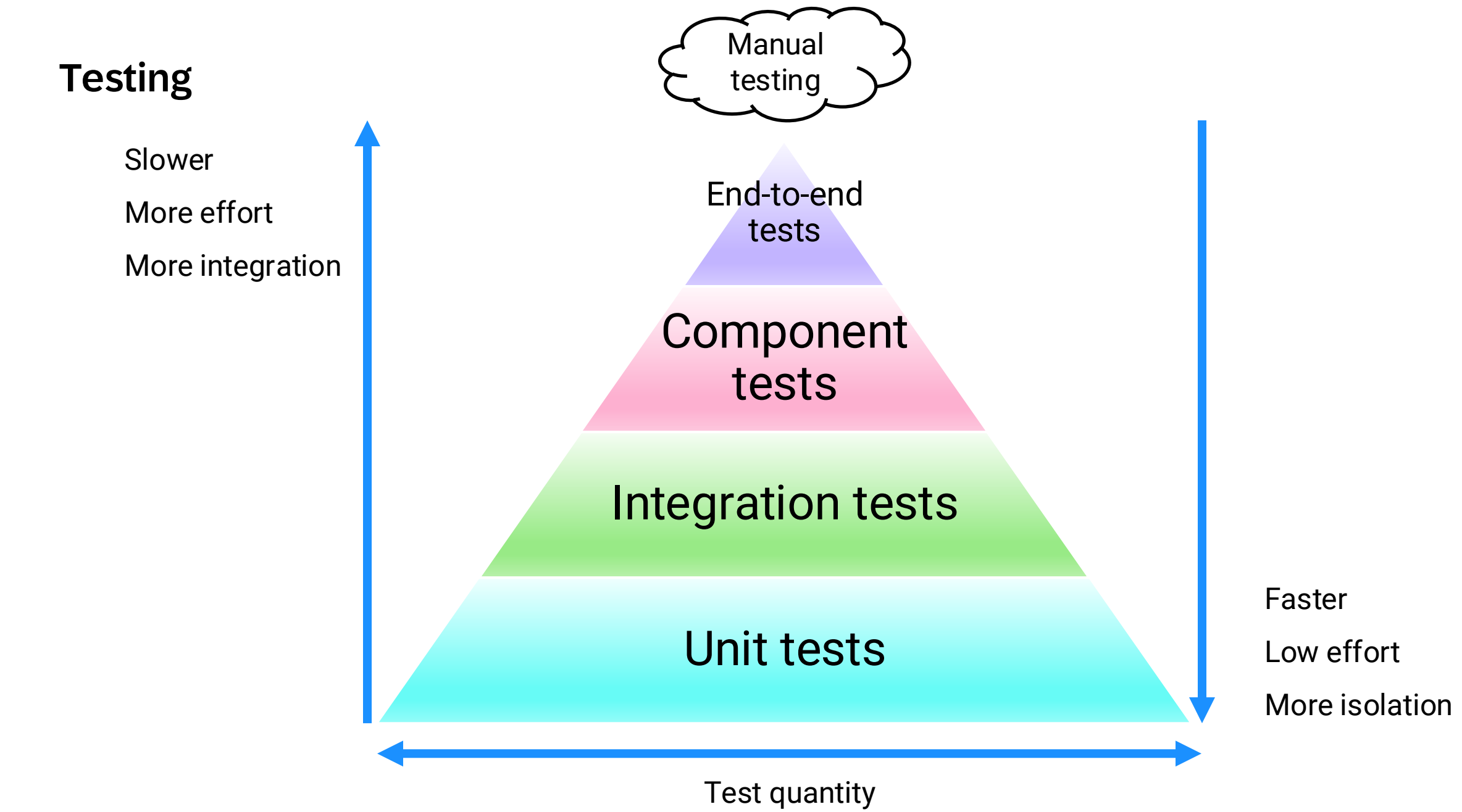
Testing



- Verifies that a system meets external requirements and achieves its goals
- Check correctness of messages between services and also checks network infrastructure
- Tested using exposed GUIs/APIs
- Reliability problems outside of team's control and asynchronous processes lead to flaky tests

Public

Testing



Public

E2E test example

Live / video

Public

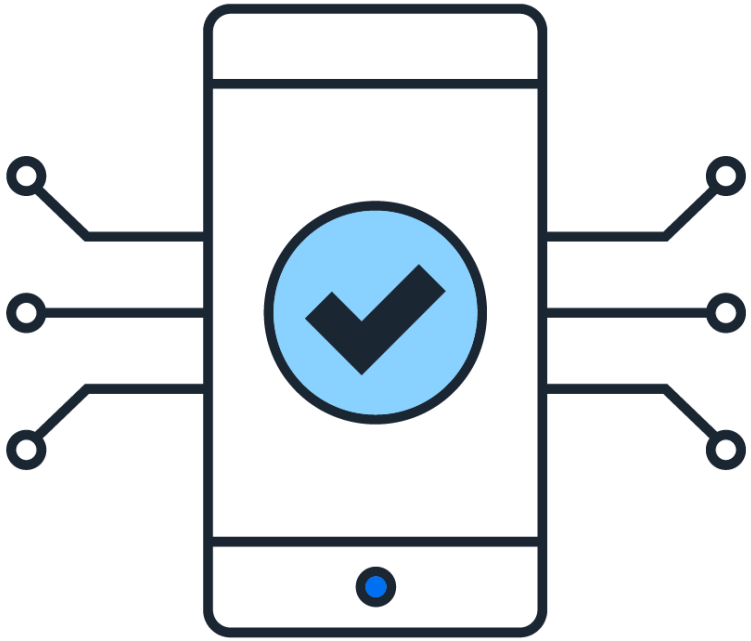
Contract testing

- An integration contract test verifies that the services meets the contract expected by a consuming service
- Checks only inputs and outputs
- Allows to make safe changes
- Written by consumers and ran by producers

Public

Another types of testing

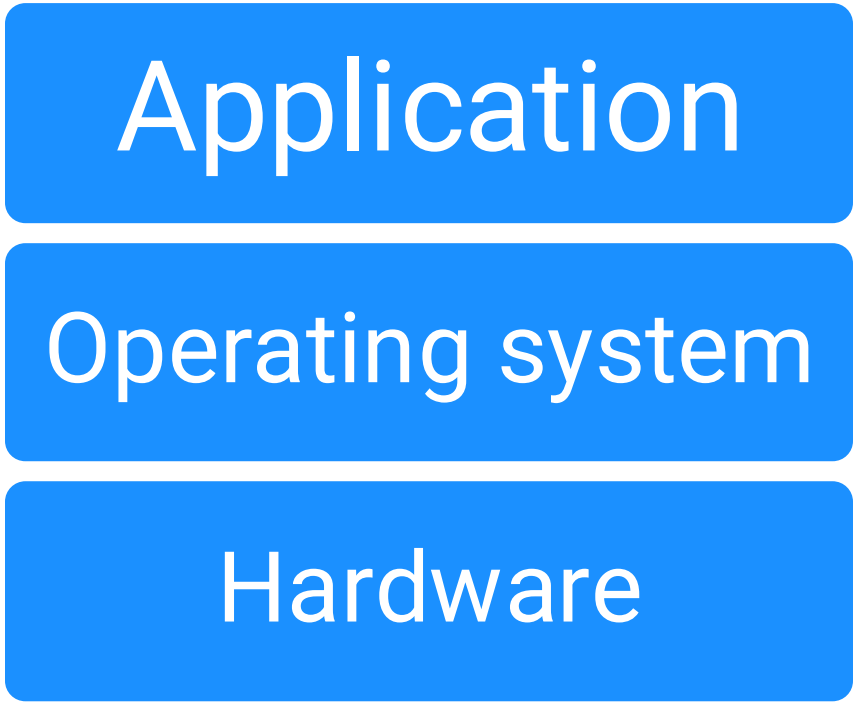
- **Load testing**
  - Testing high volume traffic
- **Resiliency testing**
  - What happens if one service is down?
- **Smoke testing**
  - Is application running?
  - health_check endpoints are usually integrated in all services
- **Security**
  - To discover exploit areas
  - Static code analysis can be used (SonarQube, CodeQL, …)
- **Performance**
  - Measuring performance metrics



Public

Deployment

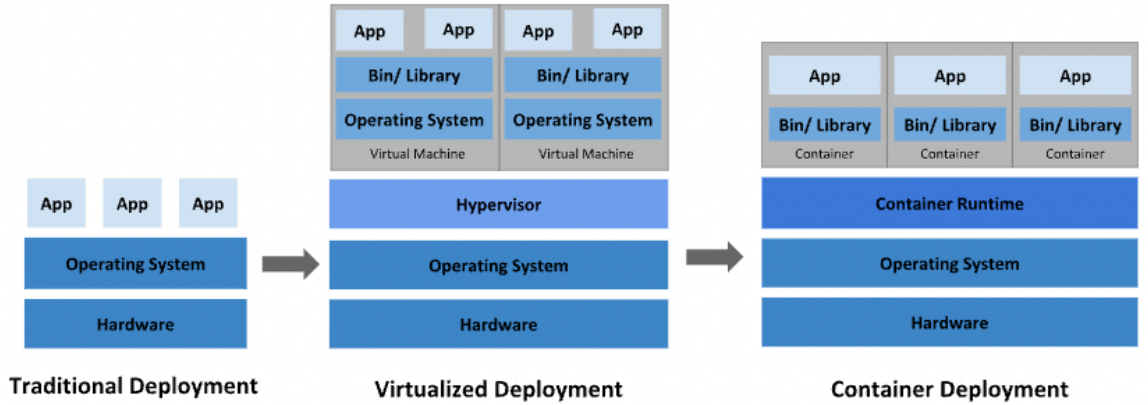
- **Traditional deployment**
  - Application code is built and deployed on a server/hardware
  - Companies need to maintain the infrastructure
  - Developer and server environments has to be compatible



Public

Container Deployment

- A container includes all the code, runtime, libraries that an application needs to run
- Benefits
  - Speed
  - Agility and flexibility
  - Resource utilization and optimization
  - **Run anywhere**



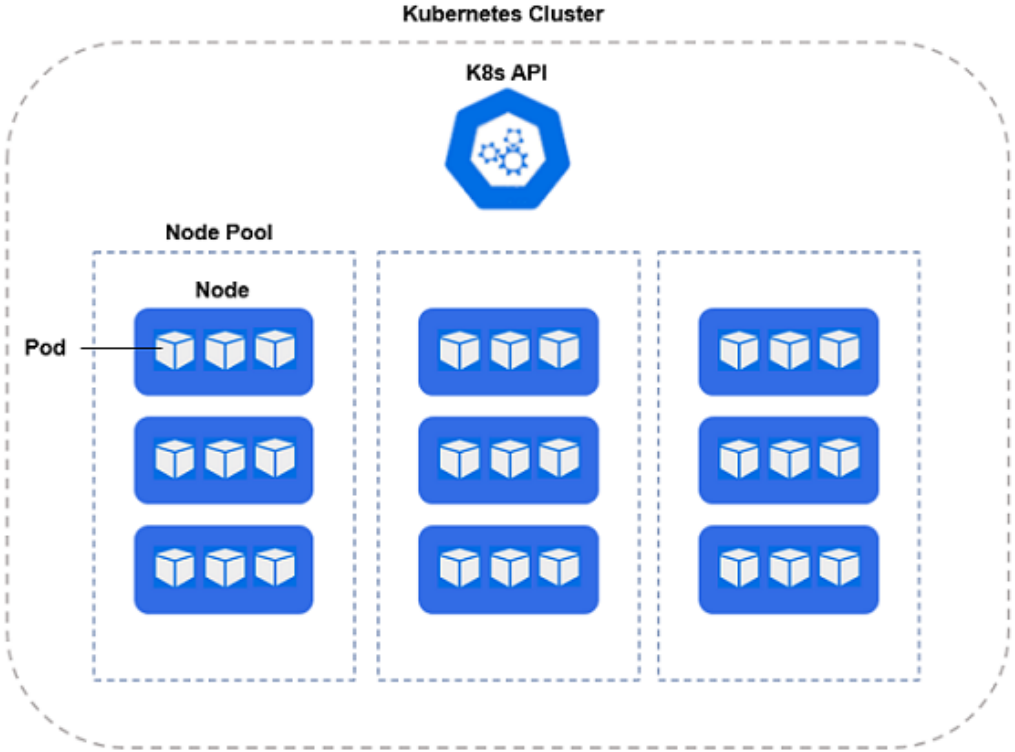
https://medium.com/@goyalarchana17/why-containerised-deployment-6b0a87c68f1e

- Docker, Podman
  - Build image of your application (containing all you need to run the application)
  - Run container of the application (one instance of your application)
- Docker Compose
  - Allows to combine multiple Docker images to one (Application + Database example)

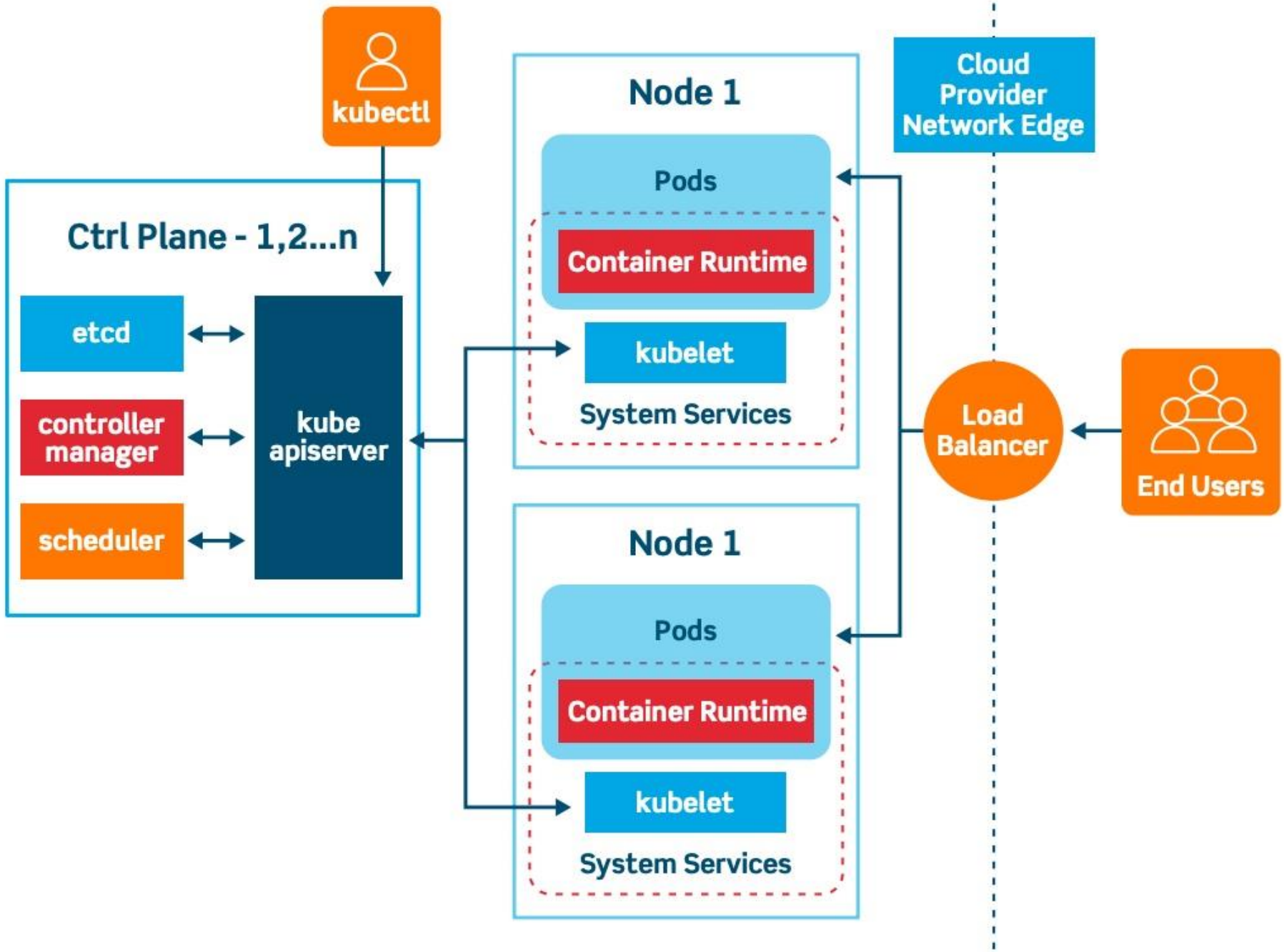
```
# DockerFile of a simple Node Application

# image to build, in this case latest Node.js LTS version
FROM node:lts-alpine

# Change to the application directory
WORKDIR /app

# Copy whole projecty
COPY . .

# Install dependencies
RUN yarn install --production

# Run the application
CMD ["node", "src/index.js"]

# Expose the port the app runs on
EXPOSE 3000
```

Kubernetes (K8s)

- Open-source container-orchestration system for automating application deployment, scaling and management
- Containers synchronization
- Handles services crashes and errors
- Pod – smallest deployable units of computing
  - – runs container
- Node – virtual or physical machine – multiple pods
- Cluster – a group of nodes or machines
- Deployments
  - Specify your desired state



https://docs.cambridgesemantics.com/anzo/v5.4/userdoc/k8s-concepts.htm



https://docs.docker.com/get-started/workshop/02_our_app/

```yaml
apiVersion: apps/v1 #  for k8s versions before 1.9.0 use apps/v1beta2  and before 1.8.0 use extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below:
          # value: env
        ports:
        - containerPort: 80
```

https://github.com/kubernetes/examples/blob/master/guestbook/frontend-deployment.yaml

Public

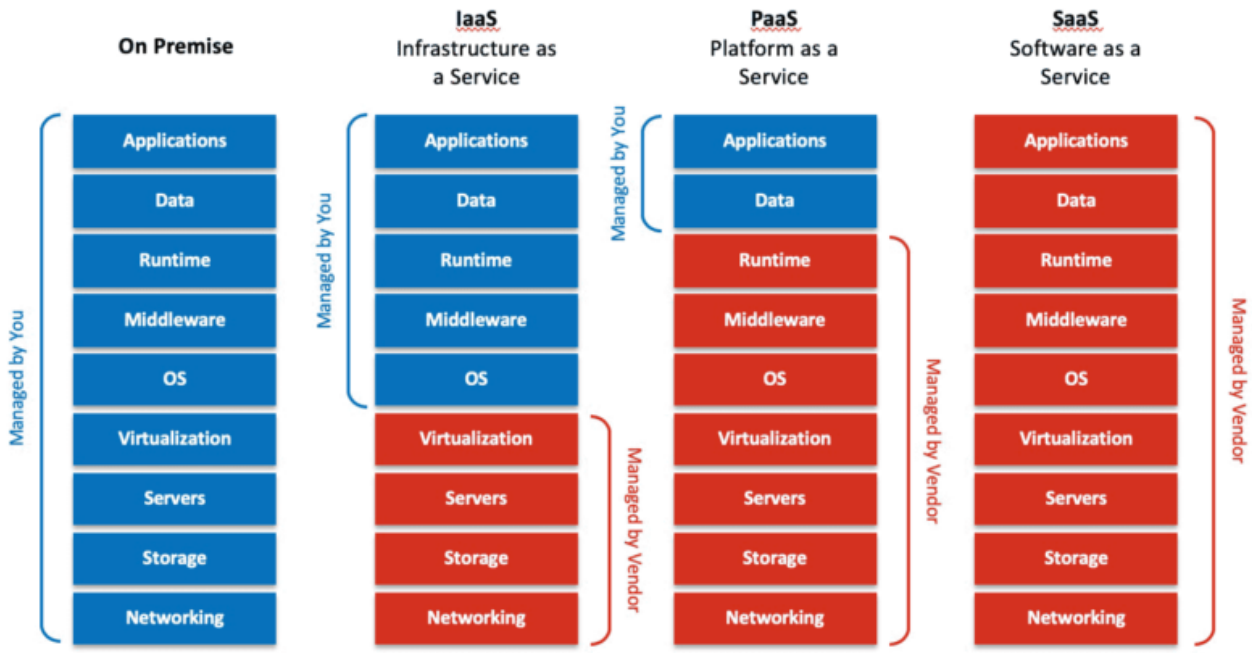
Infrastructure as a Service

- Cloud Computing Platforms allows to develop, run and manage applications without building and maintaining the infrastructure
- Vendor provides computing resources and backend infrastructure
  - Storage, servers, network, virtualization,
  - Security is shared responsibility between vendor and customer

Public

Platform as a Service (PaaS)

- Vendor provides and manages backend infrastructure, but also provides software and tools needed for application development
- Configuration as Code approach



https://hazelcast.com/glossary/platform-as-a-service-paas/

Summary

- Microservices enables independent development and deployment
- Both Backend and Frontend applications can follow the pattern
- They are not suitable for all applications and situations
- Can be easily monitored and deployed
- Testing across multiple levels to ensure quality
- Ecosystems exist to facilitate microservice development

Thank you.

Contact information:

Richard Všianský; Nodar Pylypyshak
richard.vsiansky@sap.com; nodar.pylypyshak@sap.com

Resources

- Fowler M., Microservice Testing, https://martinfowler.com/articles/microservice-testing/
- History of microservices, https://www.researchgate.net/publication/315664446_Microservices_yesterday_today_and_tomorrow
- More about microservices, https://blogs.newardassociates.com/blog/2023/you-want-modules-not-microservices.html
- VMWare, Container Deployment, https://www.vmware.com/topics/container-deployment
- Vertical/Horizontal scaling, https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/
- Microservices according to DHH, https://www.youtube.com/watch?v=rkXGSLf-rVQ

Public