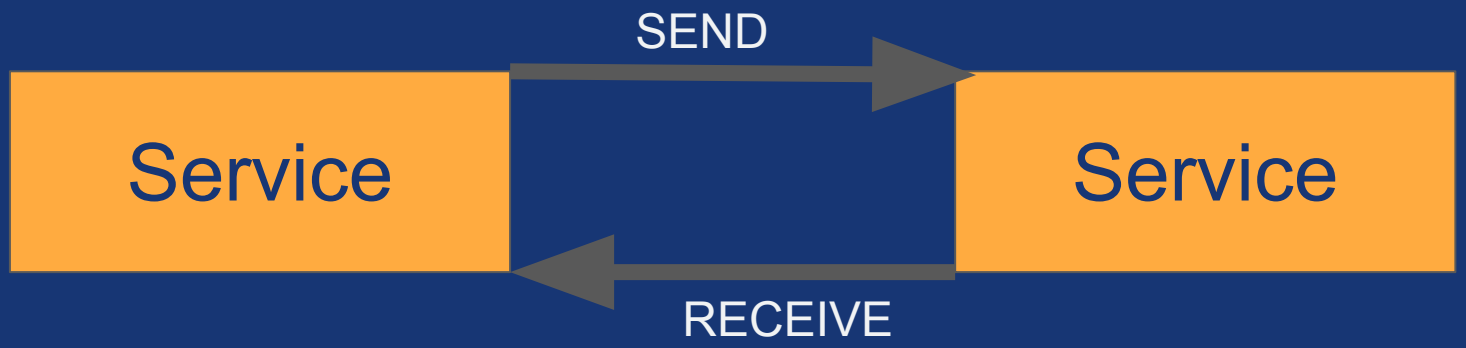
SCALEOUT

Configuration Management Tools -
Microservices

Max Andersson

Introduction to Microservices

- A Service is anything from a function to a module that handles some type of functionality.
- Domain-Driven design
- A Service can consume and produce messages



Background

- The Monolith
  - Large planning and design effort
  - Hard to get started
  - Hard to get a overview of production environment
  - Encourages software coupling
  - Complex build processes
  - Team challenges are increased
  - Hard to scale
  - Releases can take months

Background

- The Monolith (cont.)
  - Lack of
    - Innovation
    - Agility
  - Takes time to implement new features

Background

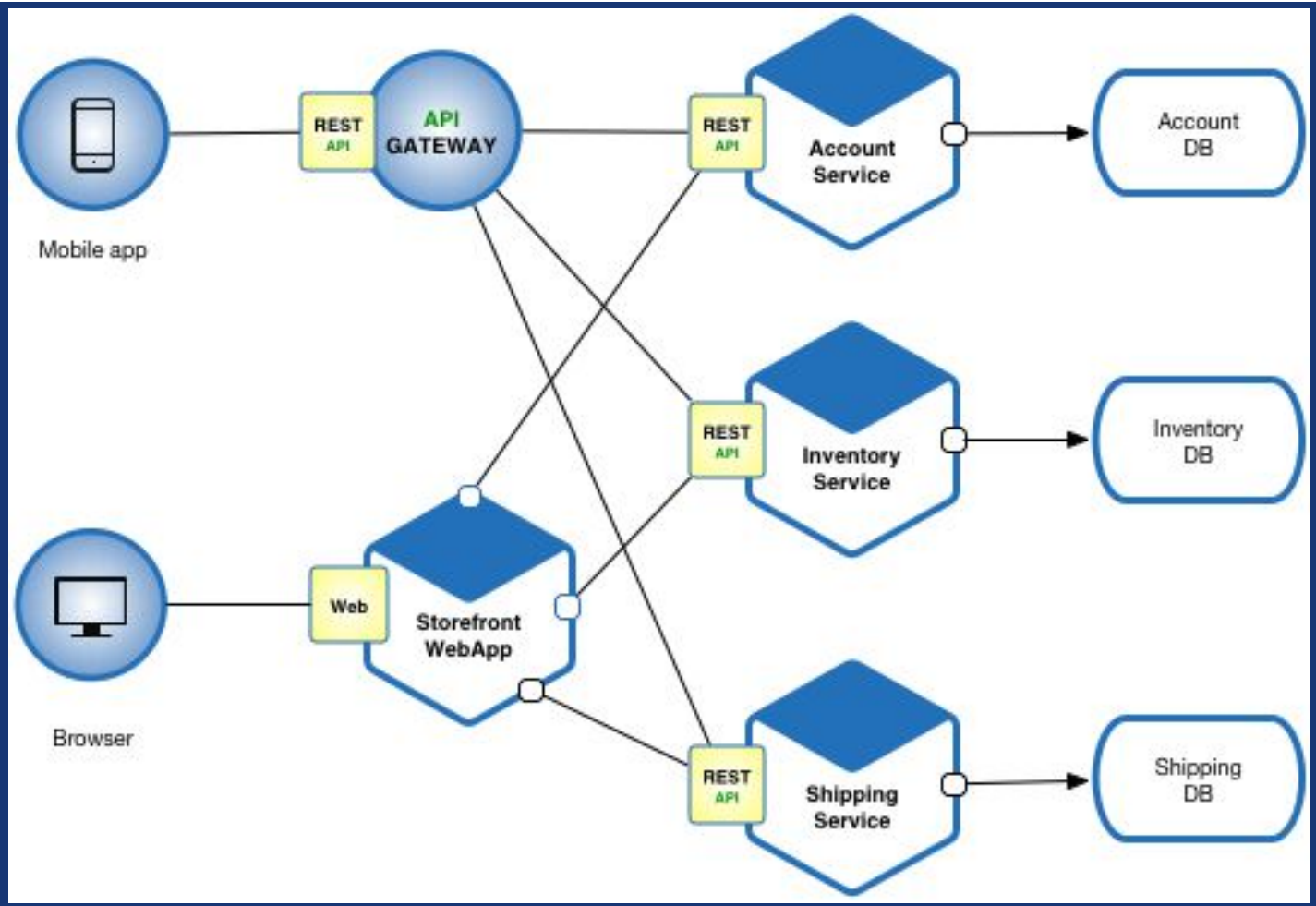
- Development teams
  - Law of diminishing returns kicks in around 7 people
  - Harder to get up to speed with a large codebase
  - Teams can take responsibility for a certain part of infrastructure.
  - Allows to use different programming languages and runtimes.

Background

- Development teams
  - Law of diminishing returns kicks in around 7 people
  - Harder to get up to speed with a large codebase
  - Teams can take responsibility for a certain part of infrastructure.
  - Allows to use different programming languages and runtimes.

Microservices

"Loosely coupled service oriented architecture with bounded context"


Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB

Service Oriented Architecture (SOA)

- Design Patterns makes it easier to:
  - Communicate
  - Implement

- Abstraction of a service
- Logic Cohesion
- Self-contained
- Loose coupling

Microservices

- Is a modern extension of SOA, but SOA != Microservices
- A more granular approach
- Can be implemented with different runtimes
- Enforces modular structure
- Works good with continuous delivery process
- Highly scalable
- Lightweight services

- Drawbacks
  - Needs more complex infrastructure
    - Load balancing
    - Service Discovery
    - Api Gateway
    - Security
    - Centralized logging/metrics/tracing
    - Fault-tolerance & resilience
    - Self-healing & autoscaling
    - Packaging, Deployment, Scheduling, Jobs

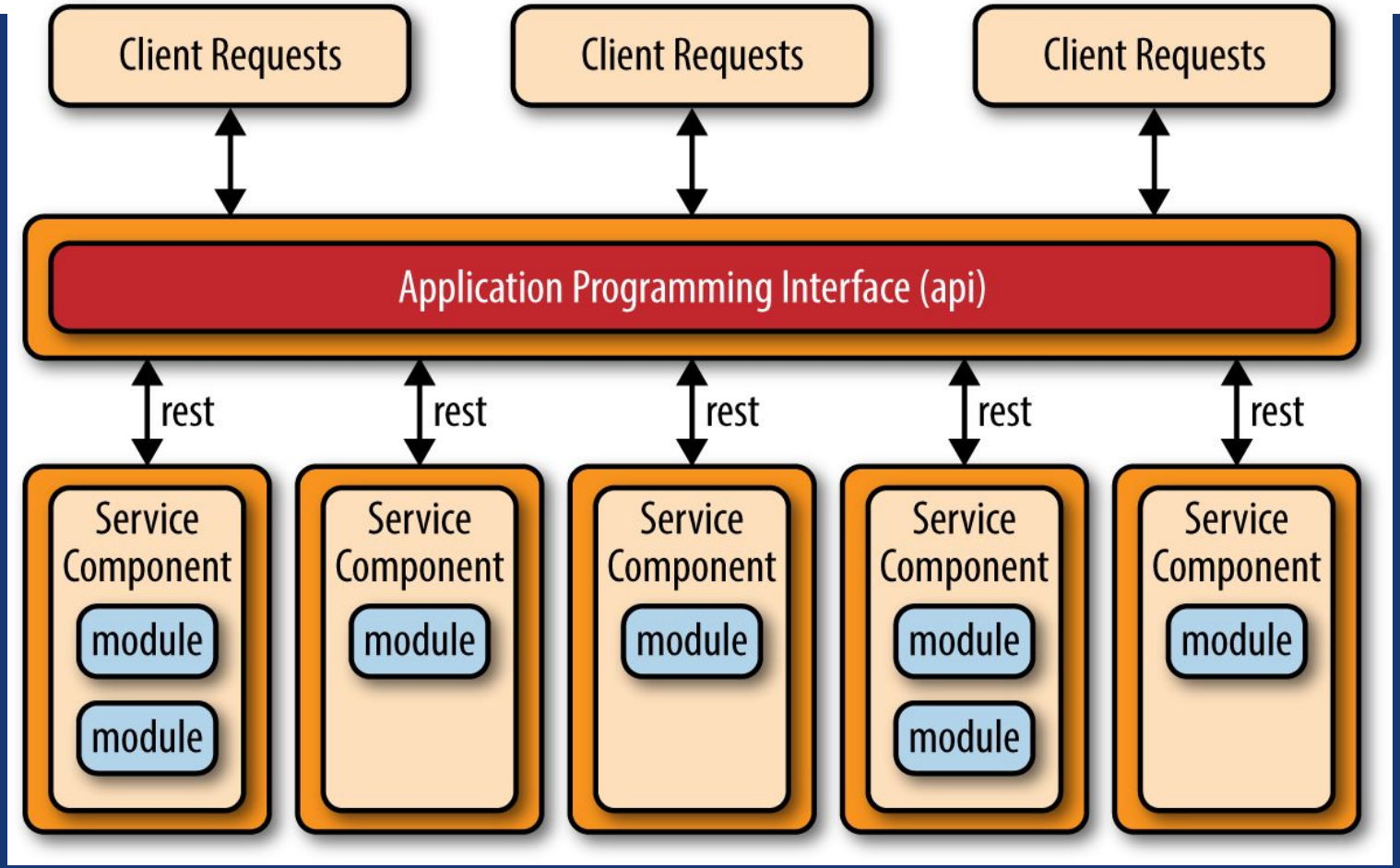
- Drawbacks (cont.)
  - Inter-Service communication can be costly (network, and message processing)
  - Testing is more complicated
  - Increases complexity
  - Migration can be costly

- Drawbacks (cont.)
  - Extra work
  - High Latency
  - Monitoring
  - Transactions
  - Testing at the application level

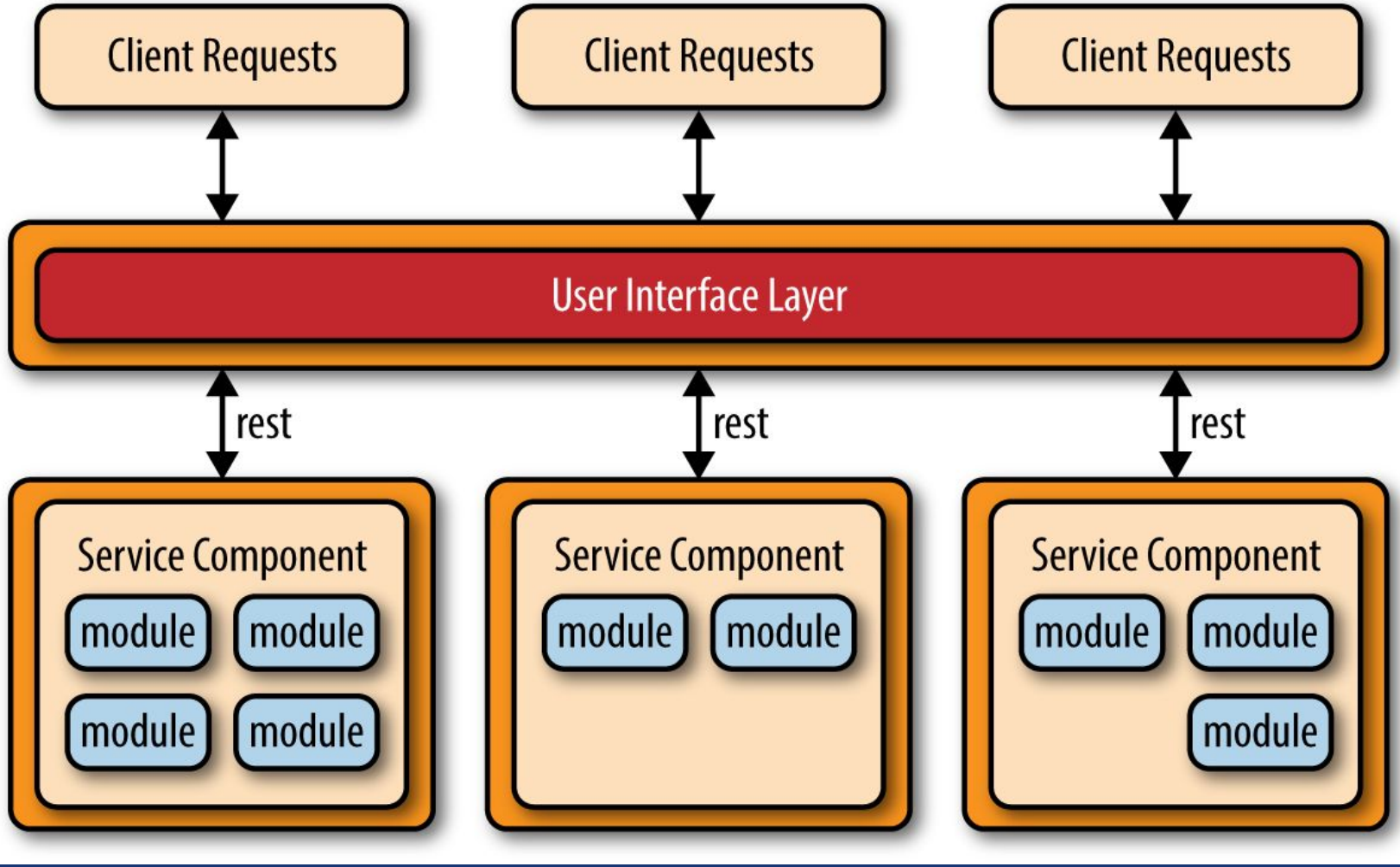
- Distributed Applications are hard
  - Needs decentralized Authentication/authorization
  - Needs to able to find services.
  - Needs to able to aggregate logs & metrics
  - Needs to be able to store configuration variables safely
  - Needs inter-service communication

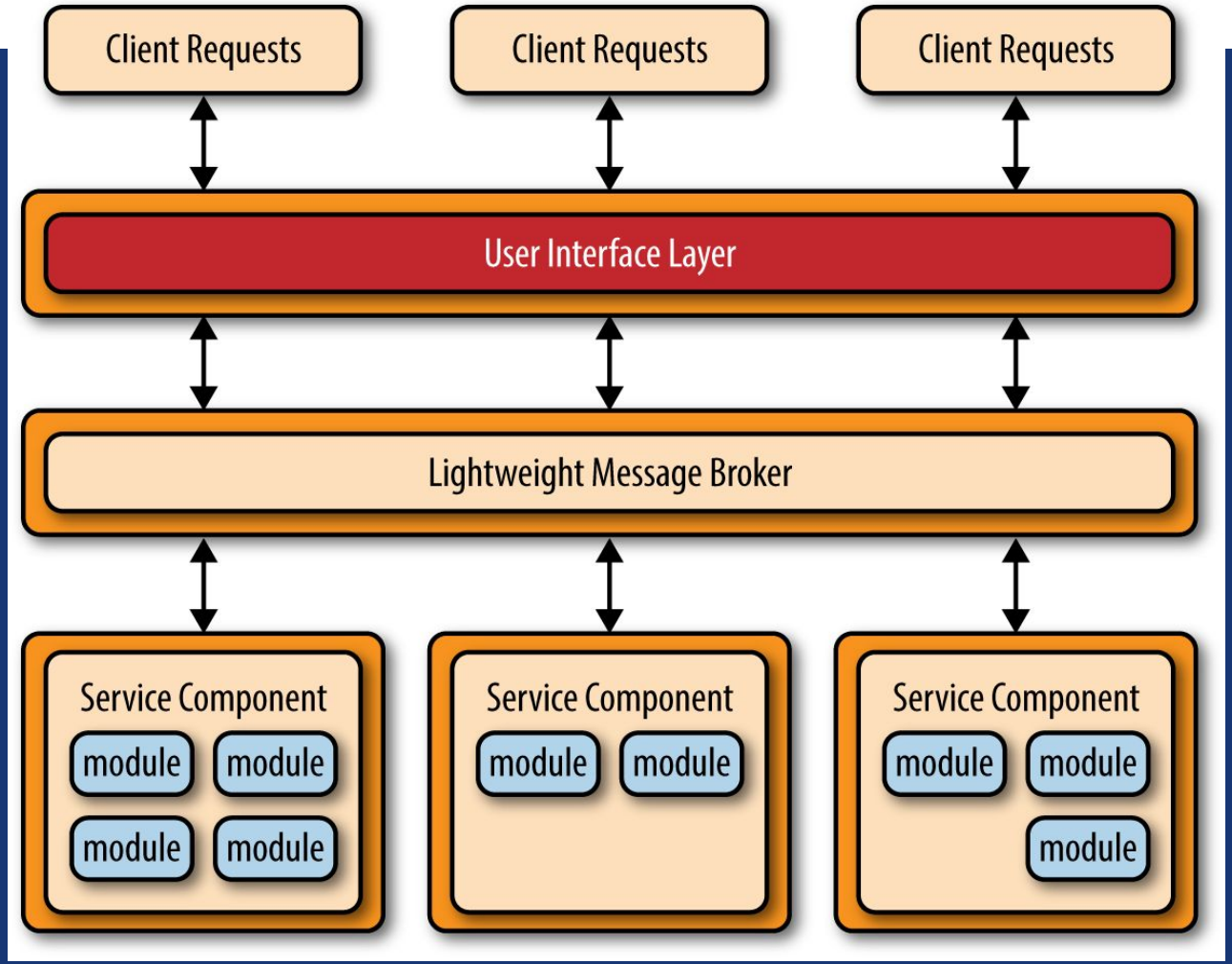
- Advantages
  - Scalability & Extensibility
  - Scaling at the service level
  - Future-proof
  - Ease of deployment and testing (at service level)
  - Reusability
  - Default architectural pattern in the cloud
  - Fault tolerance

- Topology
  - !Layered (break down layers to functional services)
  - Data stores
    - Small in implementation usually only has one persistence layer
    - Split into service data stores
- API Rest based topology
  - Small to medium applications that.
- Application REST topology
  - Larger deployments with ui layer that communicates with services.
- Centralized messaging topology
  - Uses a message broker and



https://www.oreilly.com/ideas/software-architecture-patterns/page/5/microservices-architecture-pattern


Client Requests
Client Requests
Client Requests
User Interface Layer
rest
rest
rest
Service Component
module
module
module
module
Service Component
module
module
Service Component
module
module
module


Client Requests
Client Requests
Client Requests
User Interface Layer
Lightweight Message Broker
Service Component
module
module
module
module
Service Component
module
module
Service Component
module
module
module

- Planning
  - Is it the right choice for you ? evaluate tradeoffs
  - Indentify the main functionallities of the system
  - Determine service components’ scopes; granularity
    - Size
    - Type
    - Complexity

- Design
  - Interfaces / Apis of service components
  - Api-layer for the whole system
  - Determine communication mechanism
  - Data persistence model (Central vs multiple data stores)

- Pitfalls
  - Data Migration
  - Handling service availability and responsiveness
  - Sharing functionalities, service templates
  - Login, monitoring, etc..
  - Cocktail of technologies
  - Static contract