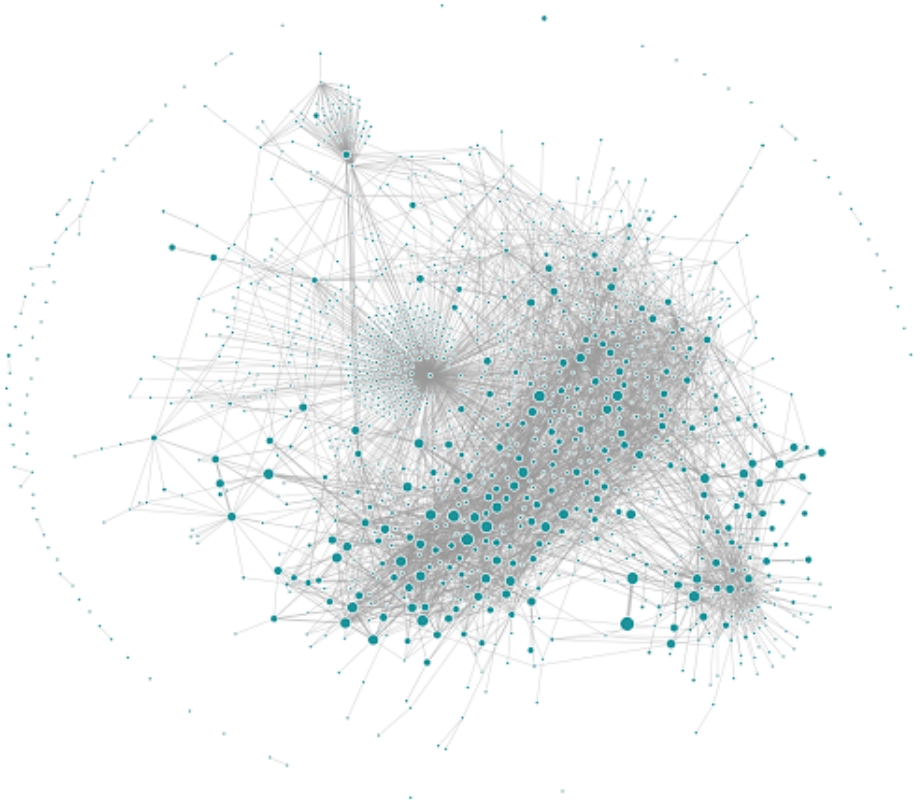
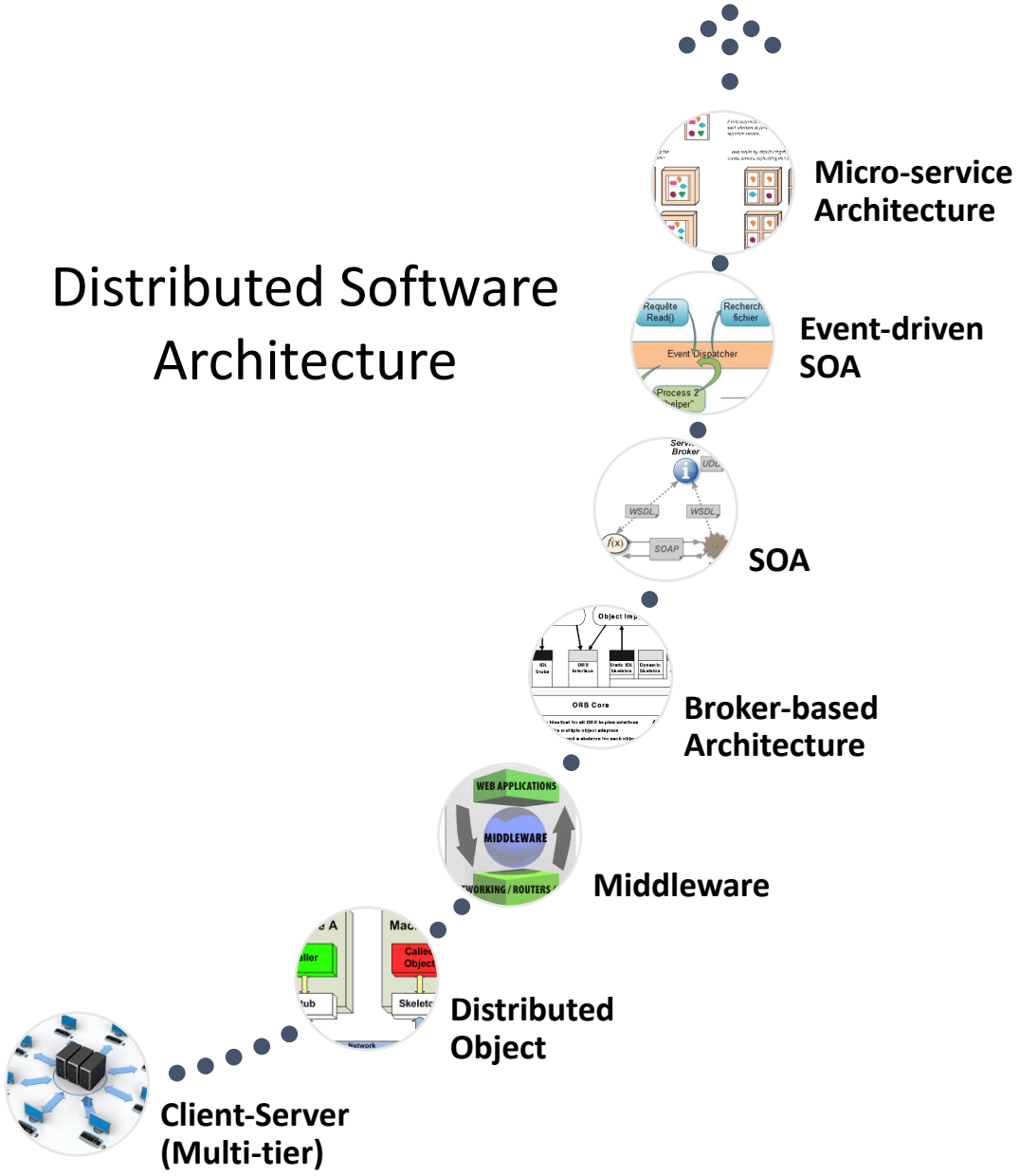
Microservice Tutorial



Slides are adopted from the Internet

Distributed Software Architecture



Component + **Connector**

- **Component patterns**
  - Distributed process
  - Distributed object
  - Service
  - **Microservice**
- **Connector patterns**
  - Remote Procedure Call （RPC)
  - REST
  - Stub/Skeleton of Distributed object
  - Middleware
  - Broker-based
  - Messaging
  - Event-driven

The ultimate goal: to deliver better software faster.

Micro-Service Architecture

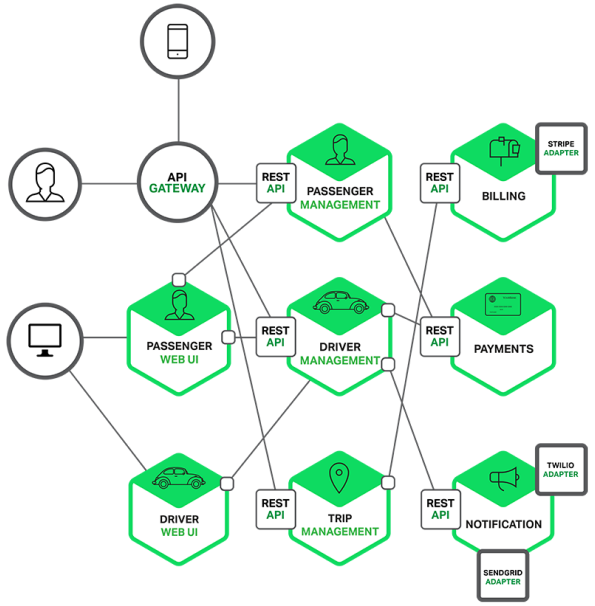
- The Micro-service architectural style is an approach to developing a single application as **a suite of small services**, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

- These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery.

Industrial Microservice System:

WeChat system: 3,000 services, over 20,000 nodes

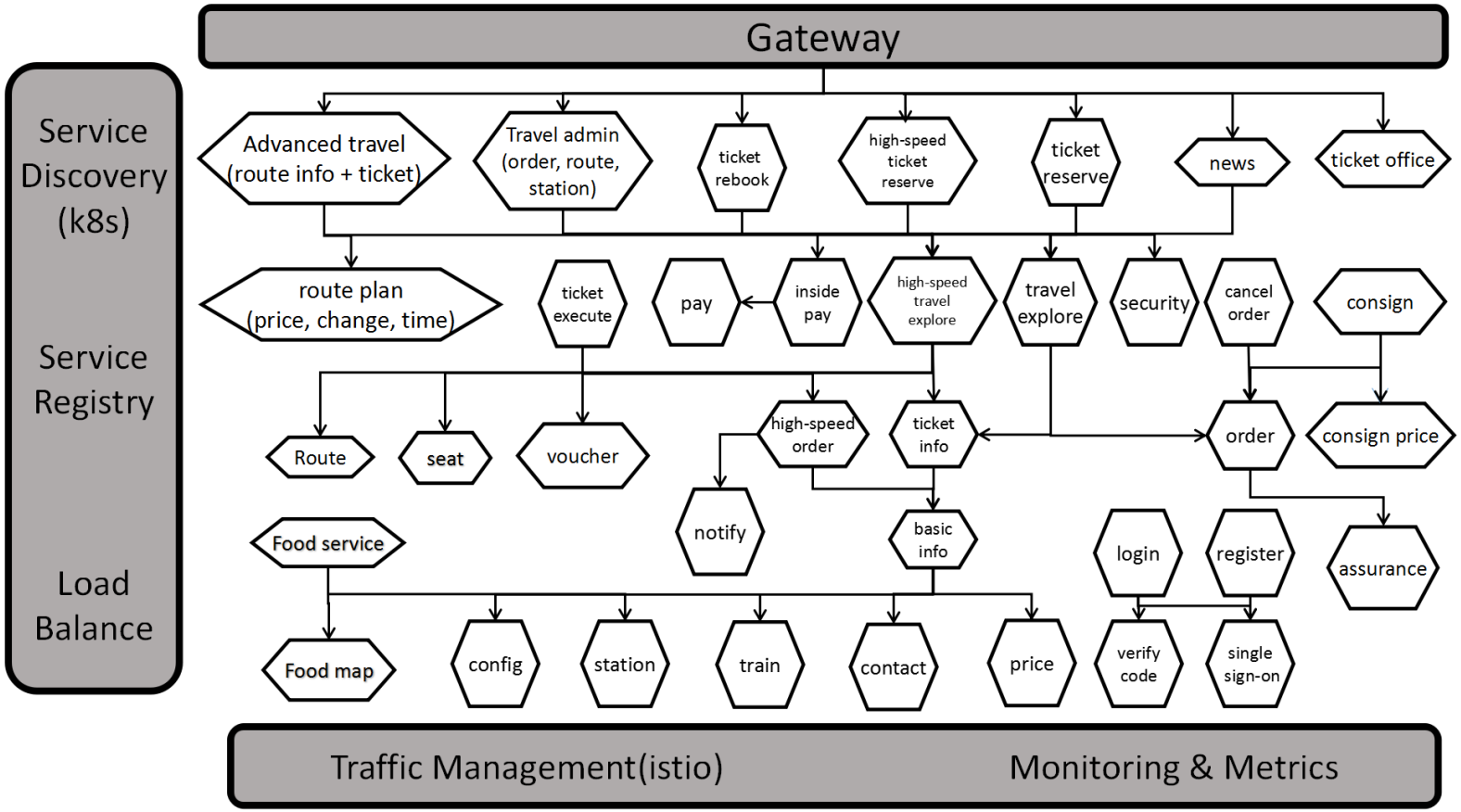
Netflix system: 500+ microservices, about two billion API requests every day



https://www.martinfowler.com/articles/microservices.html

http://2012.33degree.org/pdf/JamesLewisMicroServices.pdf

Microservice for Train Ticket Purchasing System



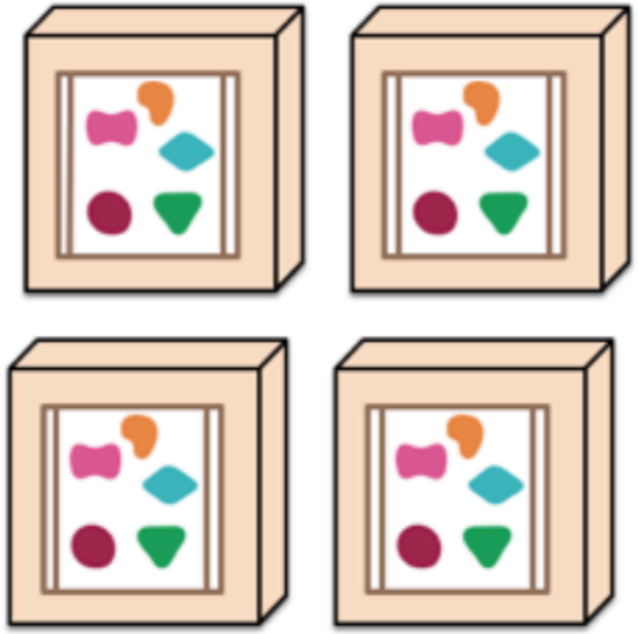
Xiang Zhou, et. al. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Transactions on Software Engineering*, DOI: 10.1109/TSE.2018.2887384.

https://github.com/FudanSELab/train-ticket

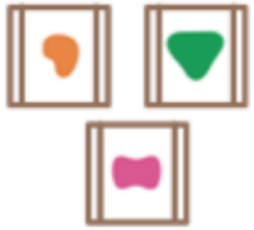
From Monolithic Application to Microservices

A monolithic application puts all its functionality into a single process...

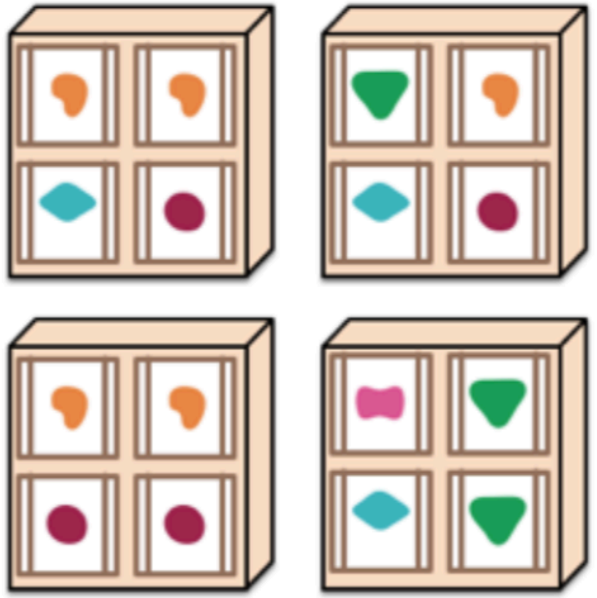
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...

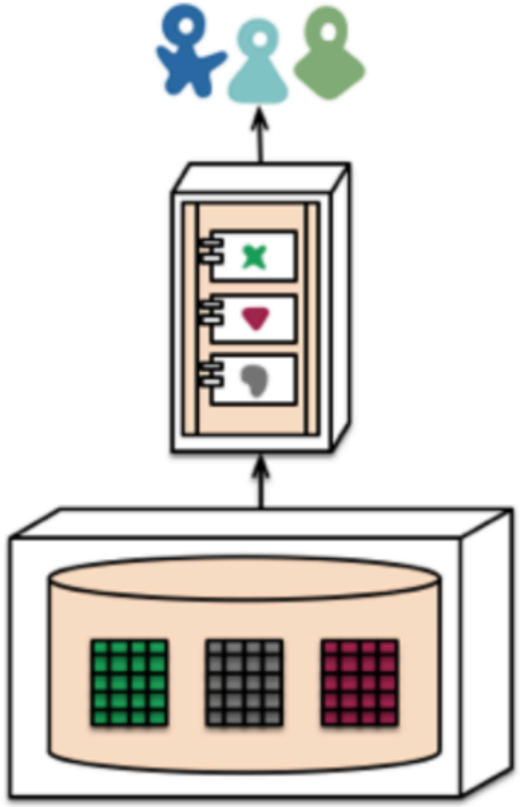


... and scales by distributing these services across servers, replicating as needed.

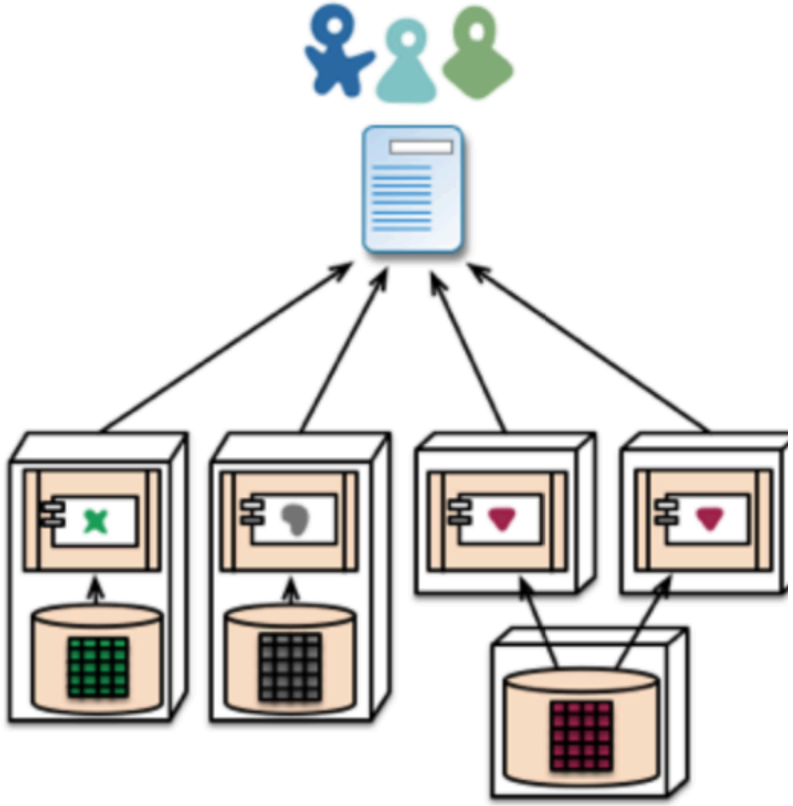


Credit: James Lewis and Martin Fowler, Microservices

Database Deployment



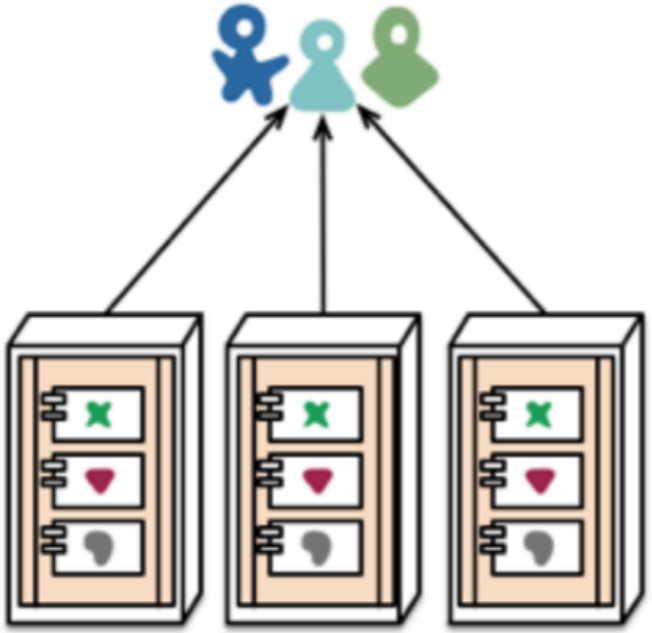
monolith - single database



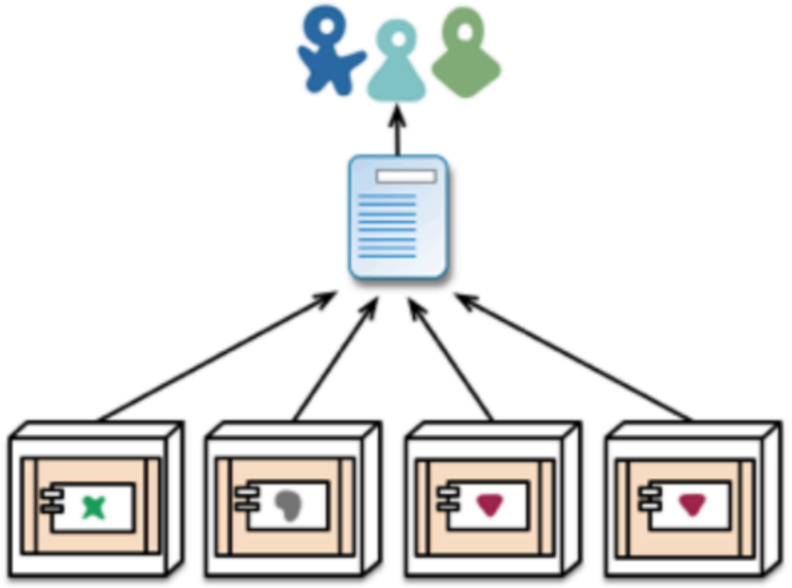
microservices - application databases

Credit: James Lewis and Martin Fowler, Microservices

Module Deployment



monolith - multiple modules in the same process

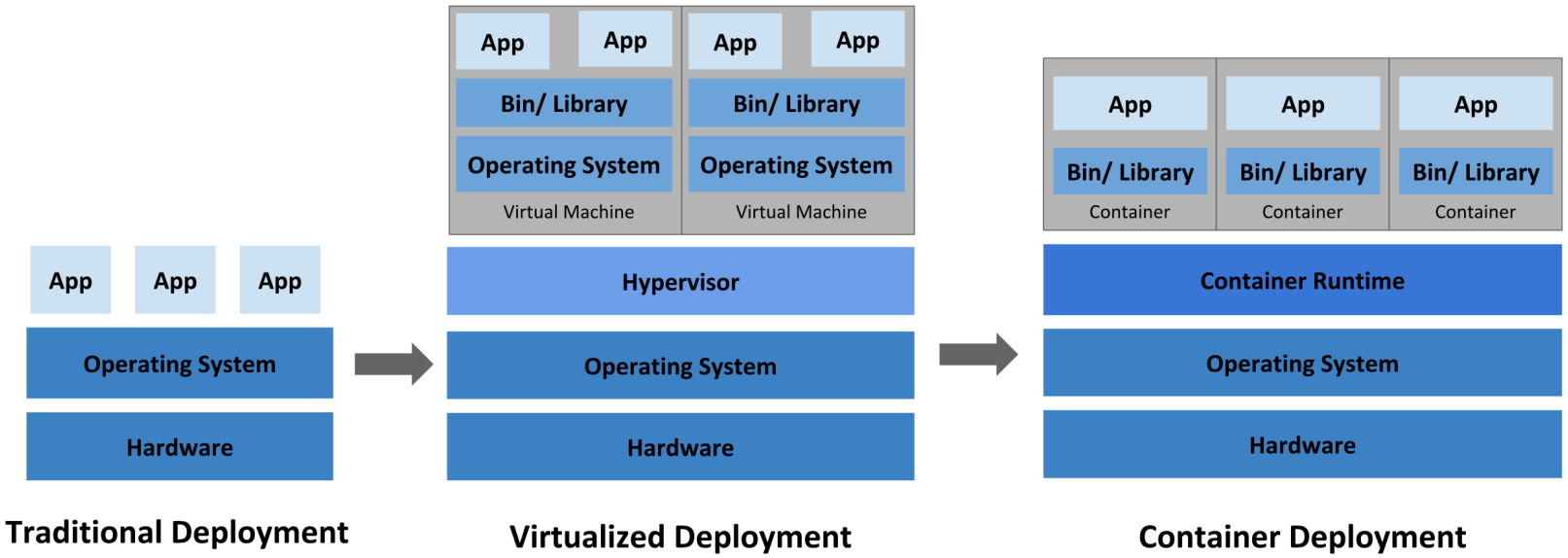


microservices - modules running in different processes

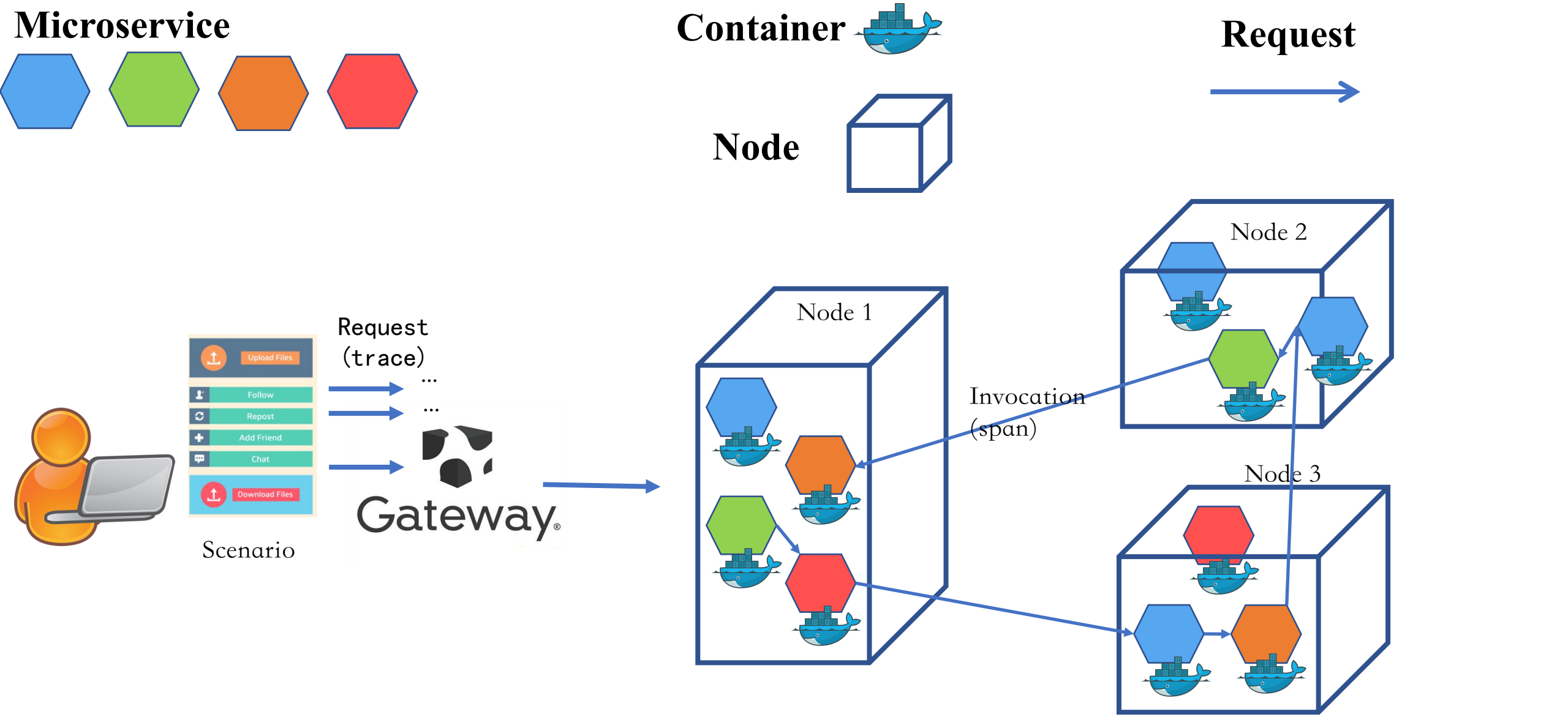
Credit: James Lewis and Martin Fowler, Microservices

Container

- Containers provide a way to package software in a format that can *run* ISOLATED on a SHARED operating system.
  - Libraries and settings required to make the software work
  - Lightweight, self-contained, standard, secured systems
  - Guarantees that software will always run the same



Container is an enabler of microservice



Credit: CodeWisdom Group @Fudan University

Micro-Service Architecture: suites of independently deployable services

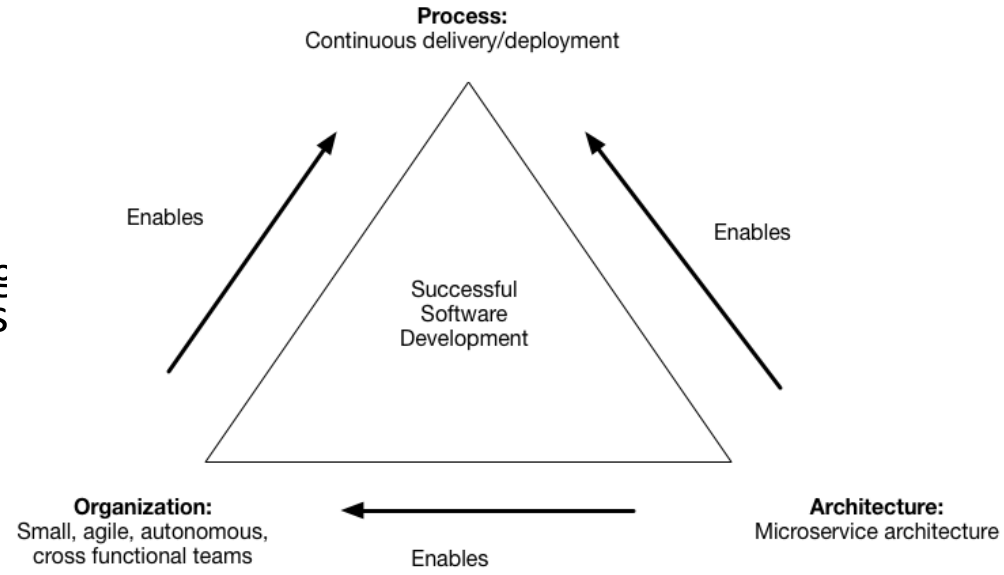
- A means to an end: enabling continuous delivery/deployment.
- Characteristics (J. Lewis and M. Fowler)
  - Using services as building blocks (components) through Published Interfaces.
  - Organized around business capabilities.
  - Development team takes full responsibility for the software in production.
  - Smart endpoints and dumb pipes
  - Decentralized control of languages

James Lewis

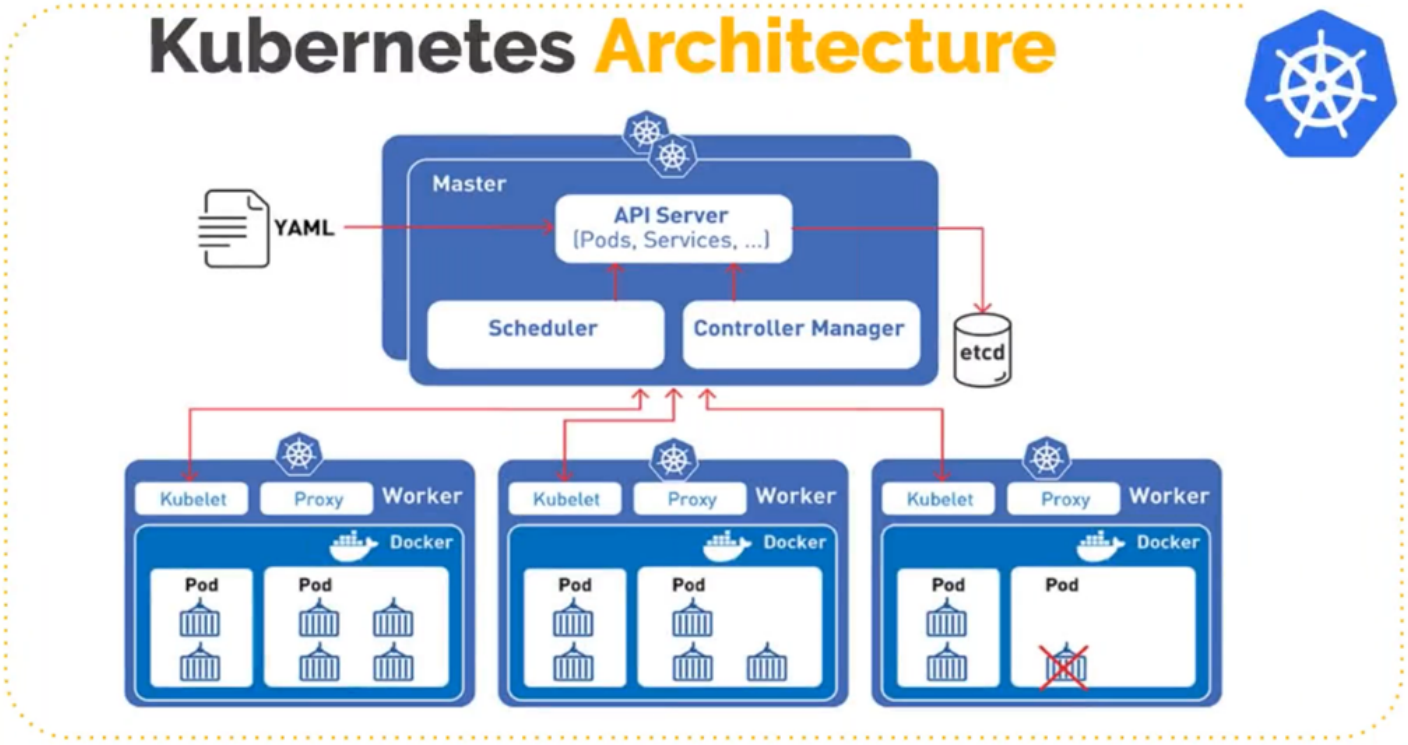
Martin Fowler

Micro-Service Architecture

- Decentralized database
  - Each service manage its own database, either different instances of the same database technology, or entirely different database system – an approach called **Polyglot Persistence**.
- Infrastructure automation
- Design for failure
  - Microservice teams would expect to see sophisticated monitoring and logging setups for each individual service such as dashboards showing up/down status and a variety of operation and business relevant metrics.
- Evolutionary Design
  - See service decomposition as a further tool to enable application developers to control changes in their application without slowing down change.
  - Microservcies can have independent replacement and upgradeability.



Kubernetes: container orchestration and scheduling



Kubernetes provides you with a framework to run distributed systems resiliently:

- **Service discovery and load balancing**
- **Storage orchestration**
- **Automated rollouts and rollbacks**
- **Automatic bin packing** (allocate containers to nodes)
- **Automatic Scaling**
- **Self-healing**

Four generations of microservice architecture:

(a) Container orchestration.

(b) Service discovery and fault tolerance.

(c) Sidecar and service mesh.

(d) Serverless architecture.

Credit: Jamshidi et al., Microservices-– The Journey So Far and Challenges Ahead

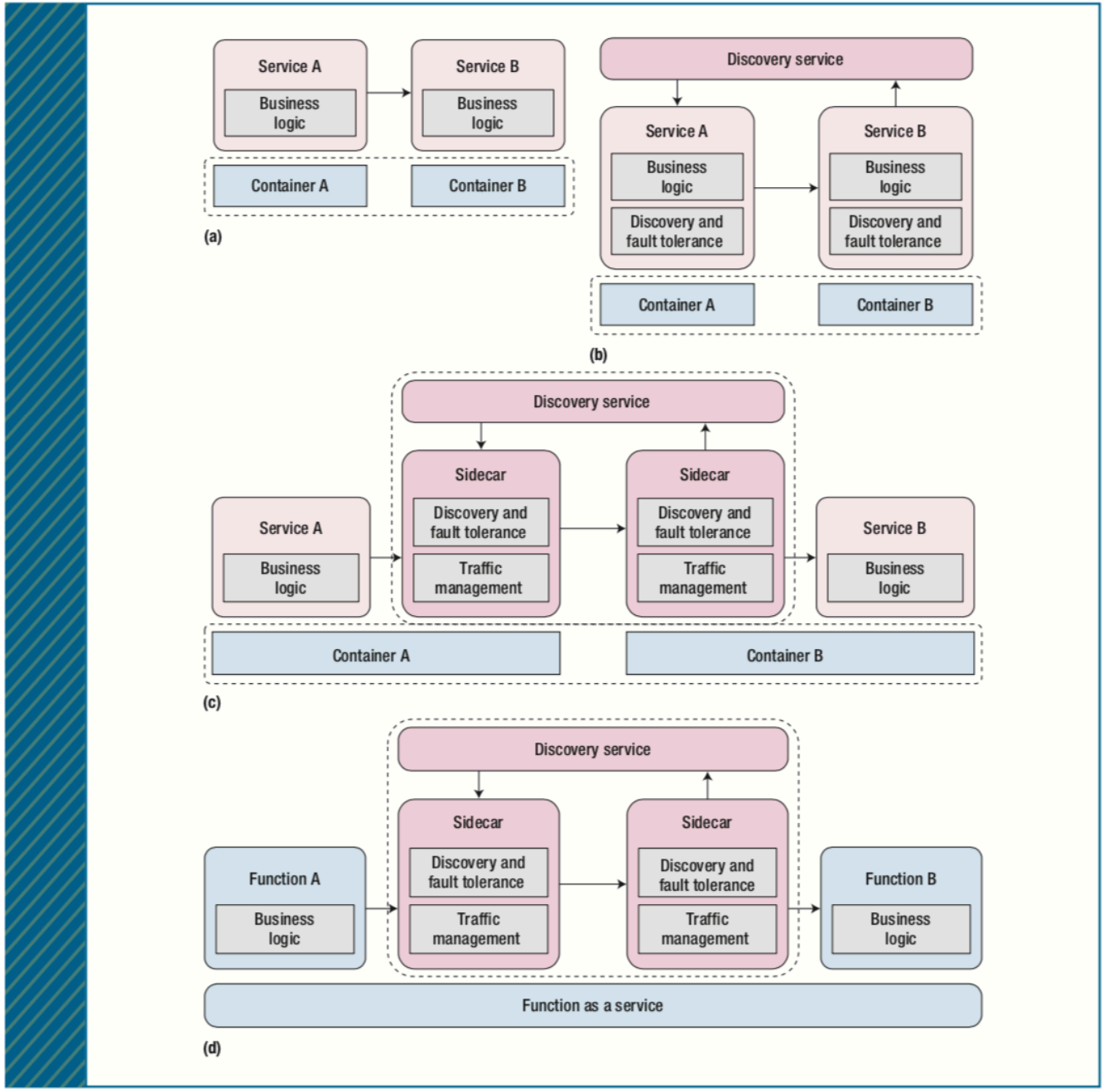
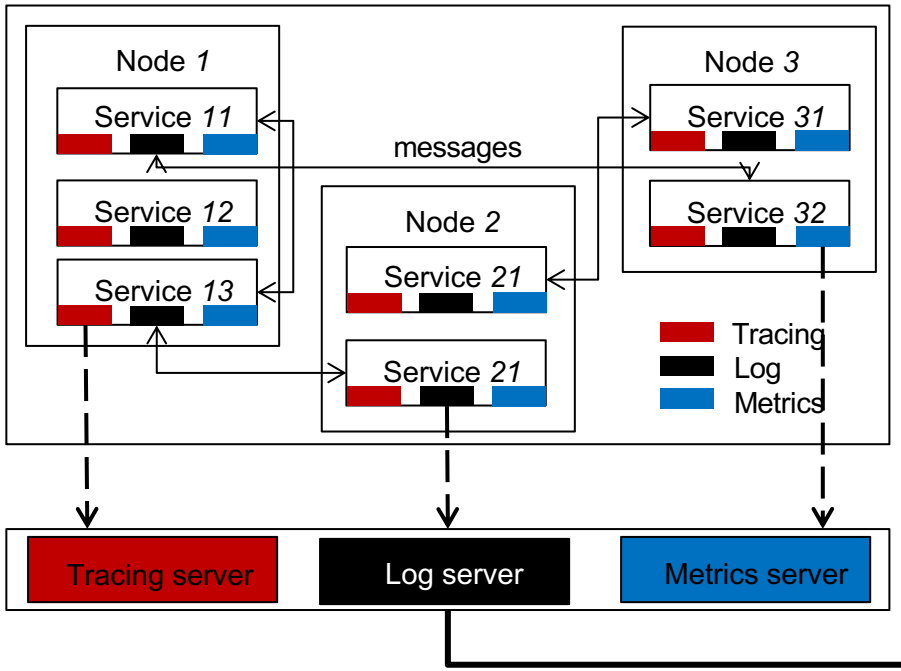


FIGURE 2. Four generations of microservice architecture. (a) Container orchestration. (b) Service discovery and fault tolerance. (c) Sidecar and service mesh. (d) Serverless architecture.

Observability Data



Anomaly Detection:
1. On the joint representation
2. Independently → result integration

Root Cause Analysis

Data Integration

Output: $S = \{(service,\ values) \mid values \subseteq \{T, L, M\}\}$

- time synchronization
- Service synchronization: which trace (or events) corresponds to which log and metric data (generated by the same action)
- Correlation between samples from each type of data
- Correlation within each type of data

Trace data

$T = \{E_{11}, E_{31,} \dots E_{12}\}$

- event timestamps
- service response time
- textual: {host, service, project, group}
- Other meta-data

Log data

$L = \{(k_1, v_1), \dots, (k_n, v_n)\}$

- system states and significant events at various critical points to help debug anomalies and perform root cause analysis
- Textual and numerical

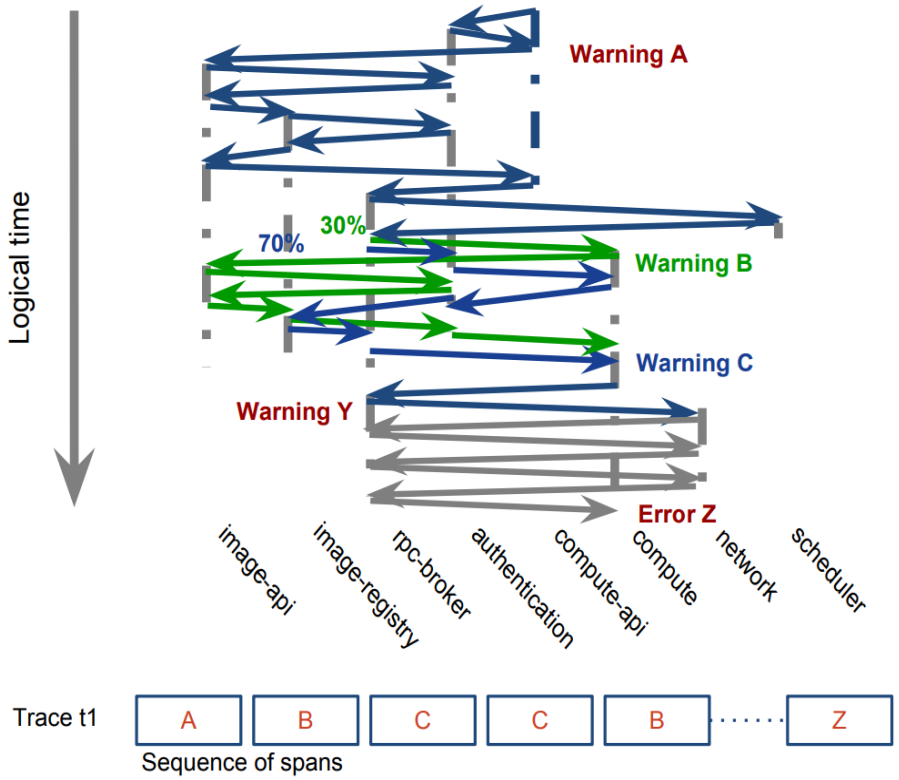
Metric data

$M = \{(cpu_1, mem_1), \dots, (cpu_n, mem_n)\}$

- cross-layer system metric data: CPU, memory, disk, network data etc.
- Metrics collected on physical and VM layer.

Credit: Anomaly Detection from Tracing Data using Multimodal Deep Learning, IEEE Cloud 2019

Observability data: traces, logs, metrics



"**traceId**": "dbd9a634c6c6faff71d6d85191b30db0",
"**name**": "get",
"**timestamp**": 1529396975572,
"**parentId**": "9a8c3402add170fa",
"**duration**": 26238,
"binaryAnnotations": [
{"key": "**host_ip**", "value": "255.24.137.124"},
{"key": "**http.status_code**", "value": "200"},
{"key": "**http.url**", "value": "https://e4b74c/v2.0/vpc/9874af&448d69"},
{"key": "**project**", "value": "neutron"},
{"serviceName": "neutron-server-cascading", "ipv4": "30.55.50.51"}},

Defines an endpoint (i.e., same function)

Credit: Anomaly Detection from Tracing Data using Multimodal Deep Learning, IEEE Cloud 2019