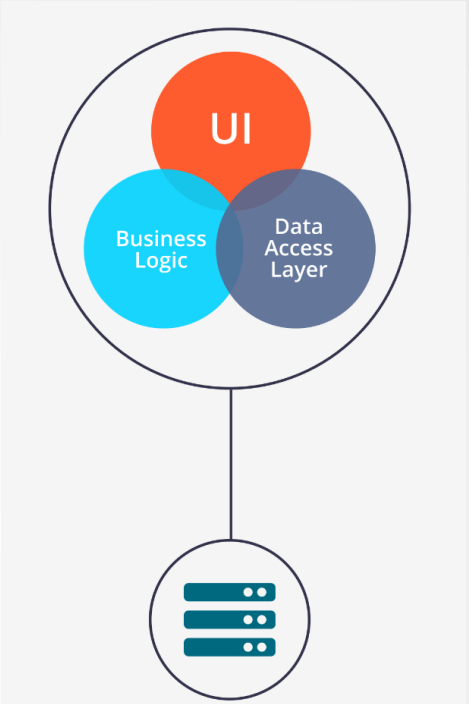
Architectural Styles

In Software Development

Monolithic

- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

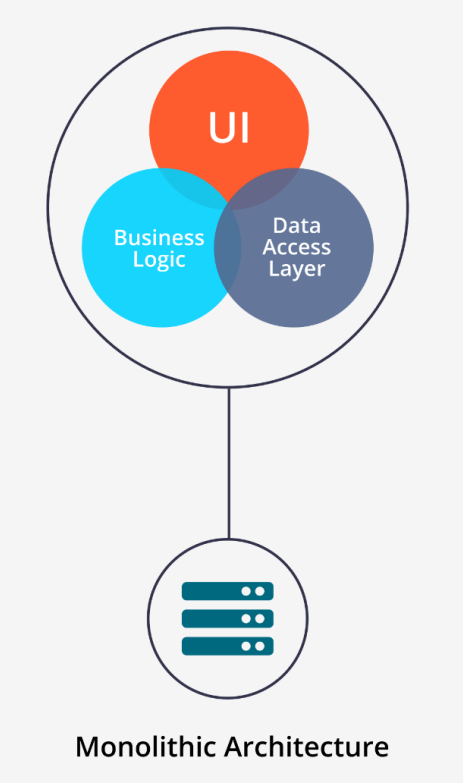
Monolithic



Monolithic Architecture

Die Applikation beinhaltet **alles**.
Intern ist sie schon getrennt (BL, DAL, UI, etc...)

Monolithic

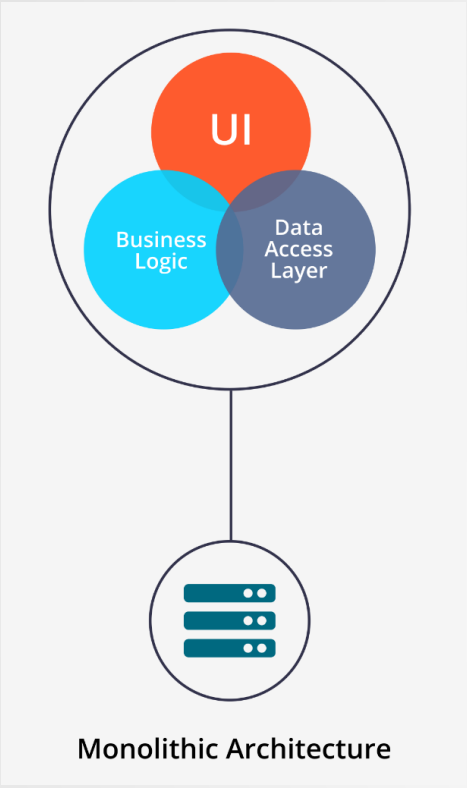


Monolithic Architecture

Vorteile

- Einfaches Deployment
  nur ein File
- Einfache Entwicklung
  nur eine Sprache, ein Team, ...
- Performanz
  braucht nichts zusätzliches
- Testing einfacher
- Debugging einfacher

Monolithic



Monolithic Architecture

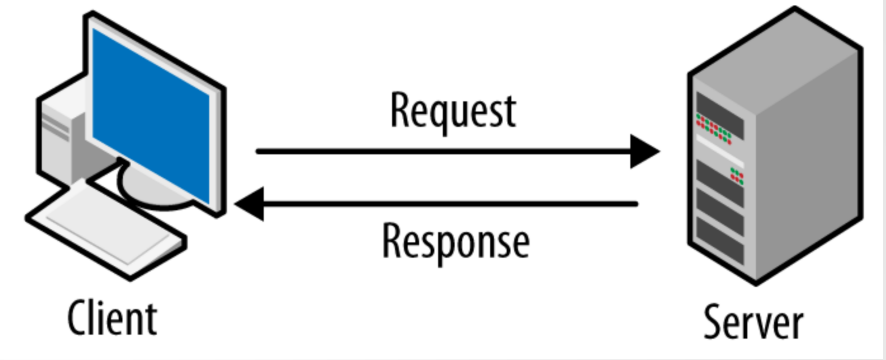
Nachteile

- Schnell zu groß -> Komplexität -> Änderungen werden zeitintensiv
- Extrem schwer skalierbar
- Reliability
  Wenn es steht, dann steht es
- Technology Adoption
  Wenn man Tech. wechselt, dann betrifft das gleich das gesamte Programm
- Deployment
  Eine kleine Änderung braucht gleich ein gesamtes Redeployment

Client -Server

- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

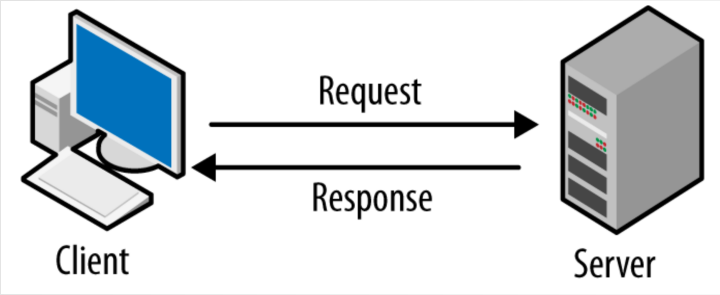
Client-Server



Request-Response Model.

Webserver, Appserver, Datenbankserver,
Fileserver, Proxies, DNS, Firewalls,
Mailserver, ...

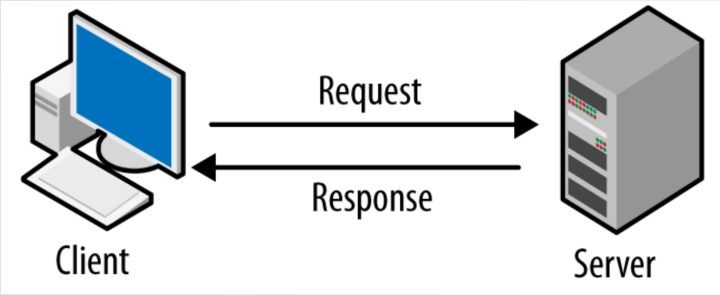
Client-Server



Vorteile

- Server erlaubt zentrales Management
- Scalability
  Es ist extrem einfach Clients hinzuzufügen und einfach mit neuen Servern
- Sicherheit
  Server müssen abgesichert sein
- Einfacher Betrieb
  Weil alles zentral ist

Client-Server



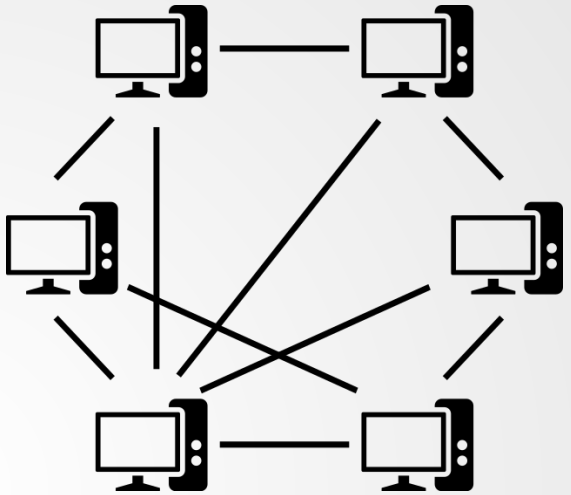
Nachteile

- Overloading
  Zu viele Requests zu einer bestimmten Zeit
- Kosten
  Server sind teurer als Clients
- Singel Point Of Failure
  Wenn der Server ausfällt dann steht alles still

P2P

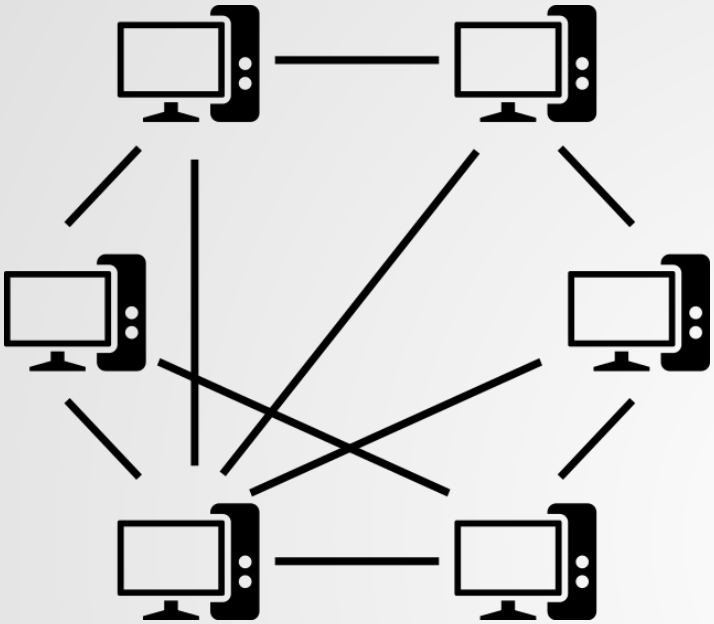
- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

P2P



Peers verbinden sich **miteinander**.
Kein Server.

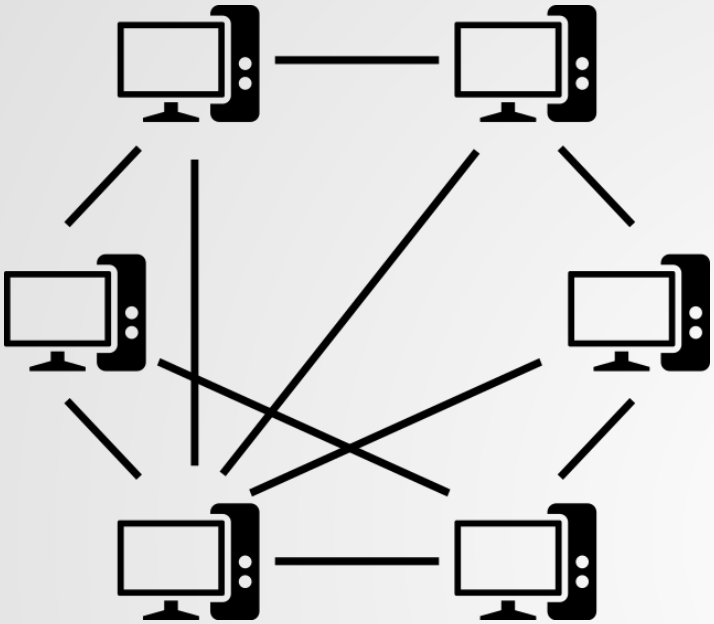
P2P



Vorteile

- Fehlertoleranz ist höher
  Kein Single Point Of Failure mehr
- Einfache Architektur
  Keine Unterscheidung zwischen Client und Server mehr
- Einfach aufzubauen

P2P



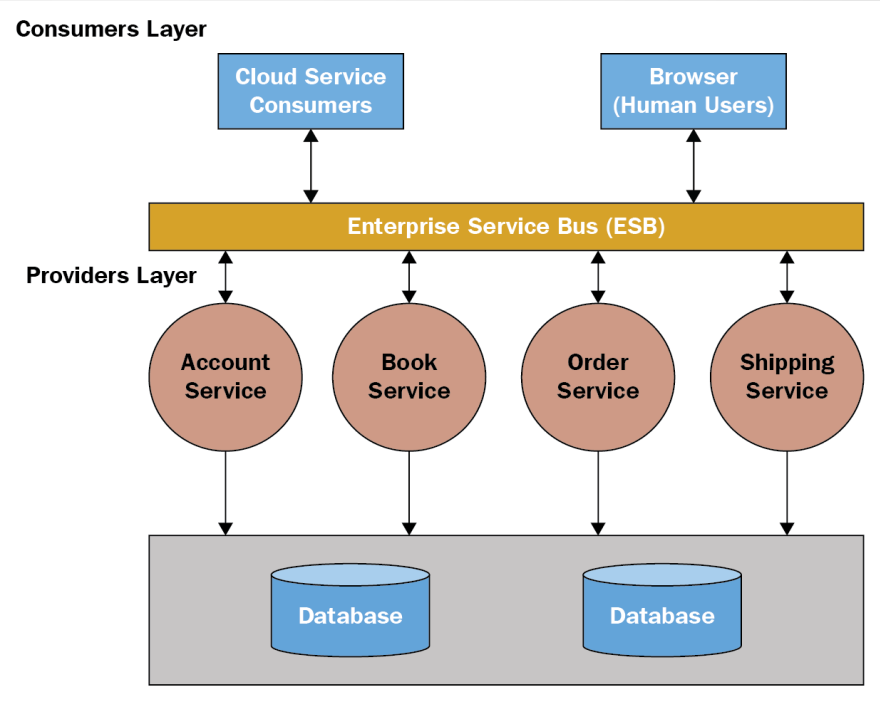
Nachteile

- Braucht mehr Netzwerk-Resourcen
  Zumindest am Client
- Schlechteres zentrales Backup
  Weil die Daten verteilt sind
- Netzwerkpartitionen
  Kein ACID mehr, nur BASE
- Schlechtere Security
  Da alles am Client läuft und wenn Dir die Maschine gehört auf der der Code läuft, dann hab ich als Entwickler verloren :(
- Komplexere Protokolle
  Alleine einen Leader zu wählen ist eine große Aufgabe

SOA

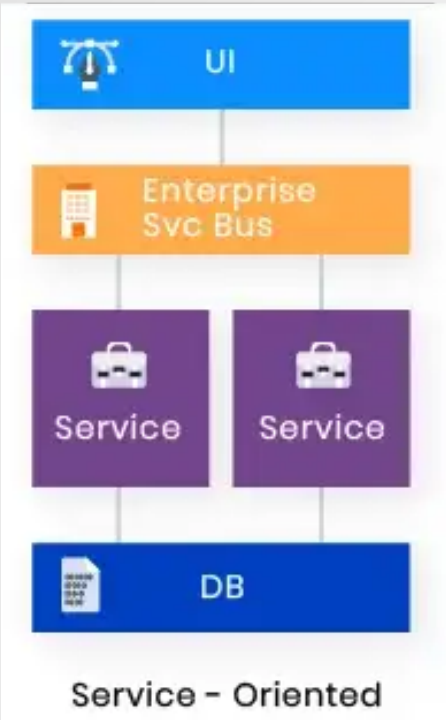
- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

SOA



Über einen **Enterprise Service Bus** (**ESB**) werden für viele verschiedene Applikationen **Services** angeboten.

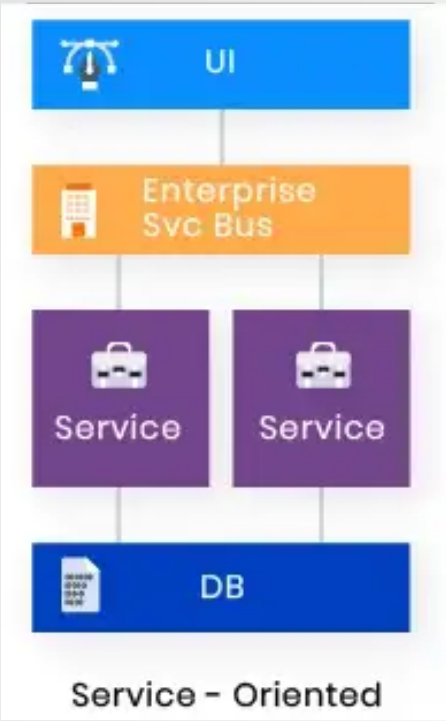
SOA



Vorteile

- Unabhängige Örtlichkeit
  Es ist egal wo die Services sind
- Wiederverwertbarkeit
  Die Services brauchen nur einmal entwickelt werden
- Improved Scalability
- Parallel Development
  An mehreren Services kann gleichzeitig gearbeitet werden

SOA

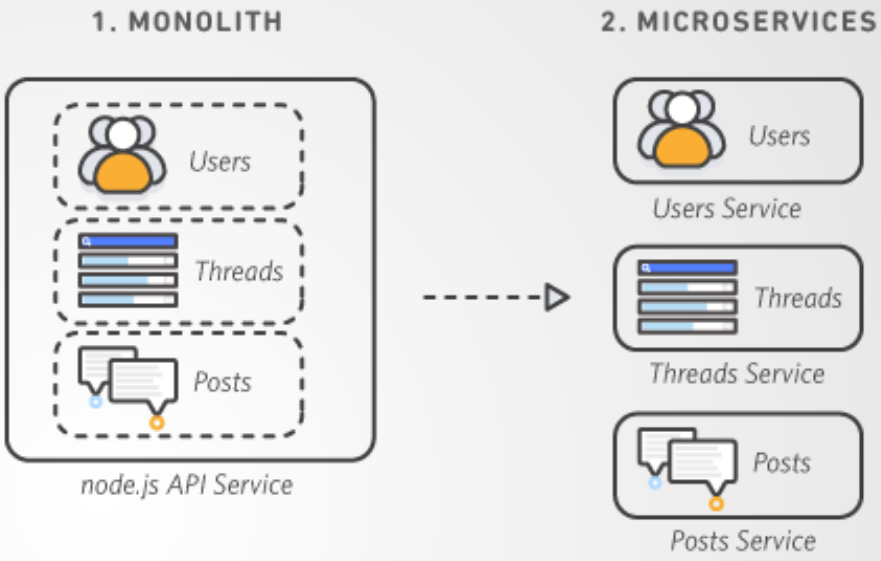


- **Nachteile**
  - Hohe Planungskosten
    Das muss man erst alles vorhersehen
  - Langsamere response-time
    Wenn Services kommunizieren, dann dauert das
  - Viele verschiedene Services
    Überblick zu behalten ist schwierig

MicroServices

- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

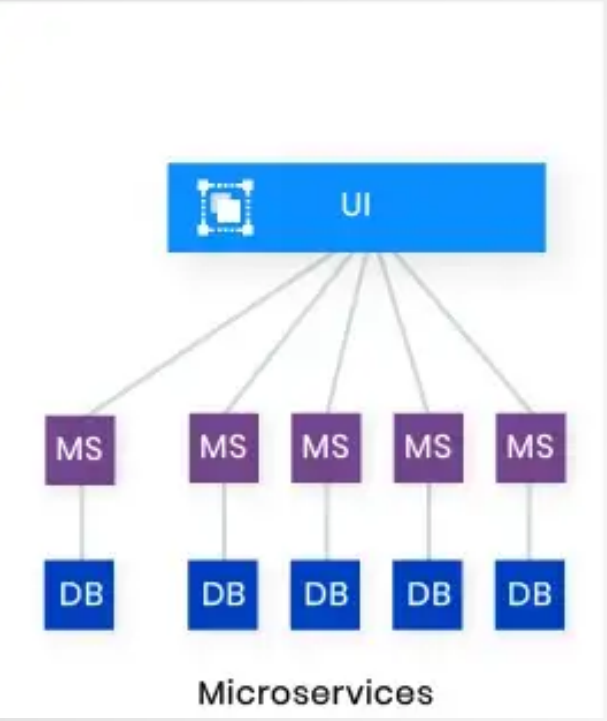
MicroServices



Einzelnen **Funktionen** einer Software werden abgetrennt, unabhängig von einander implementiert und mit **Schnittstellen** verbunden.

Andere Ebene als SOA (ist Applikation aufwärts im ESB).

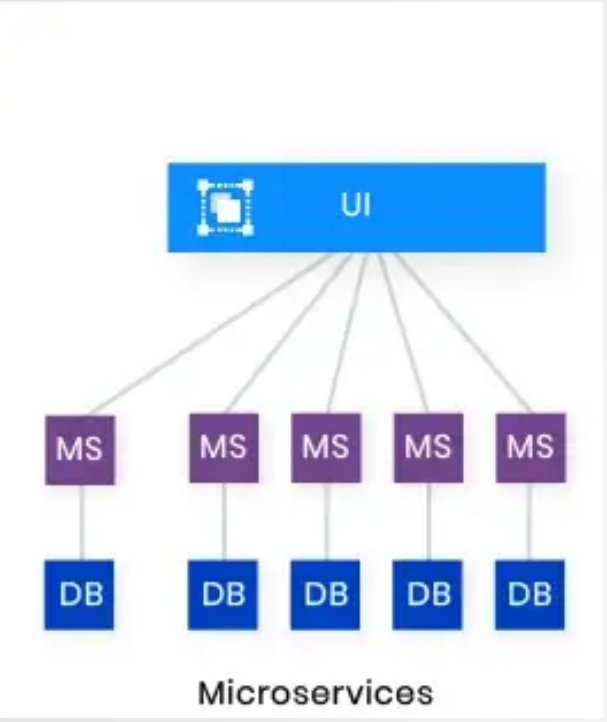
MicroServices



Vorteile

- Einfach skalierbar
  siehe Kubernetes
- Gute Resilienz
  Wenn Services entkoppelt sind
- Code ist einfacher verständlich
  Weil kleinere Module
- Gut zum experimentieren
  Einzelteile gut wiederverwendbar
- Unabhängiges Deployment der einzelnen Teile

MicroServices



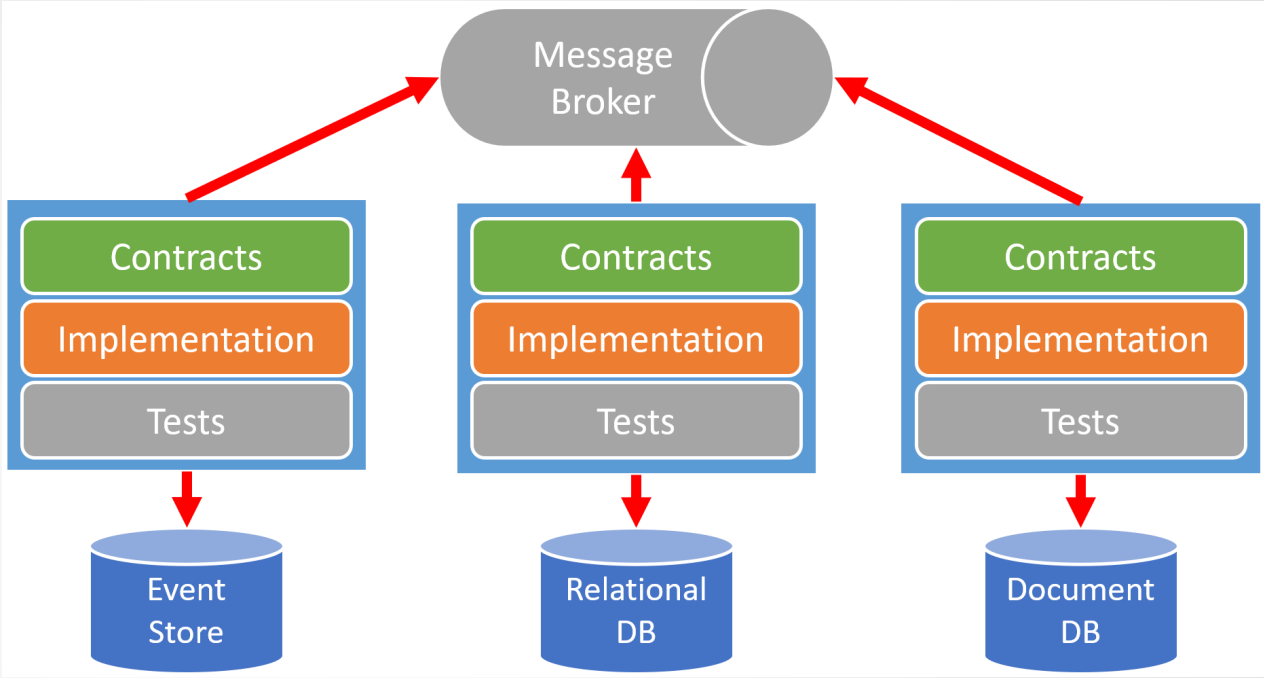
Nachteile

- Kommunikation ist komplexer
  Entkopplung über Queues, etc...
- Benötigt mehr Resourcen
  Weil viele Deployments
- Globales Testen und Debuggen ist schwieriger
  Wegen multiplen Deployments
- Nicht toll für kleine Applikationen
- Komplexes Deployment

Message Driven

- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

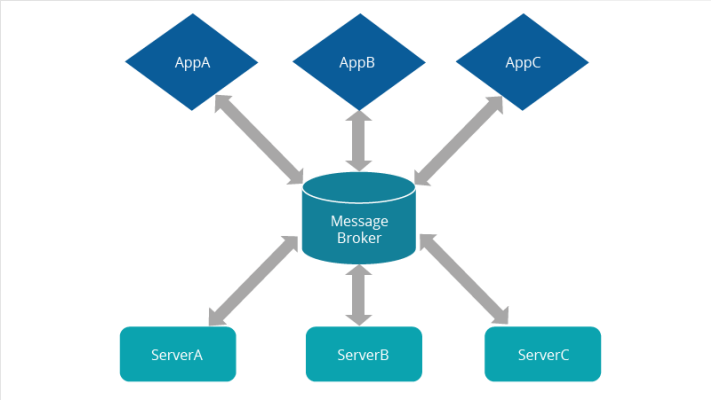
Message-Driven



Services sind getrennt. **Messages** (sync oder async) fungieren **als Interface**.

Jede Message hat einen **vordefinierten Empfänger**!

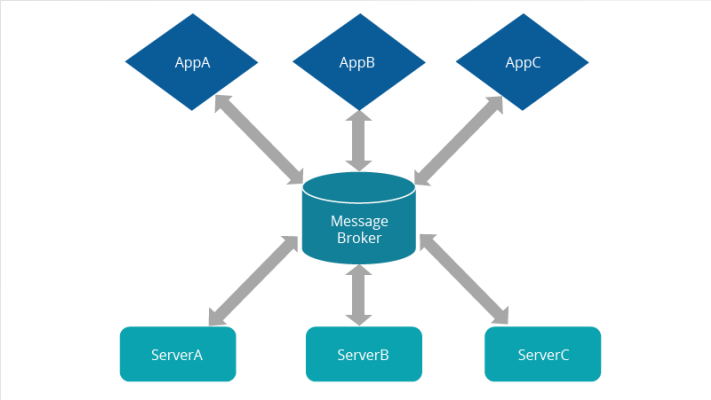
Message-Driven



Vorteile

- Resourcen können gleich wieder freigegeben werden, nachdem die Nachricht abgeschickt wurde.
- Lose Kopplung
- Bessere Skalierbarkeit
- Bessere Resilienz

Message-Driven



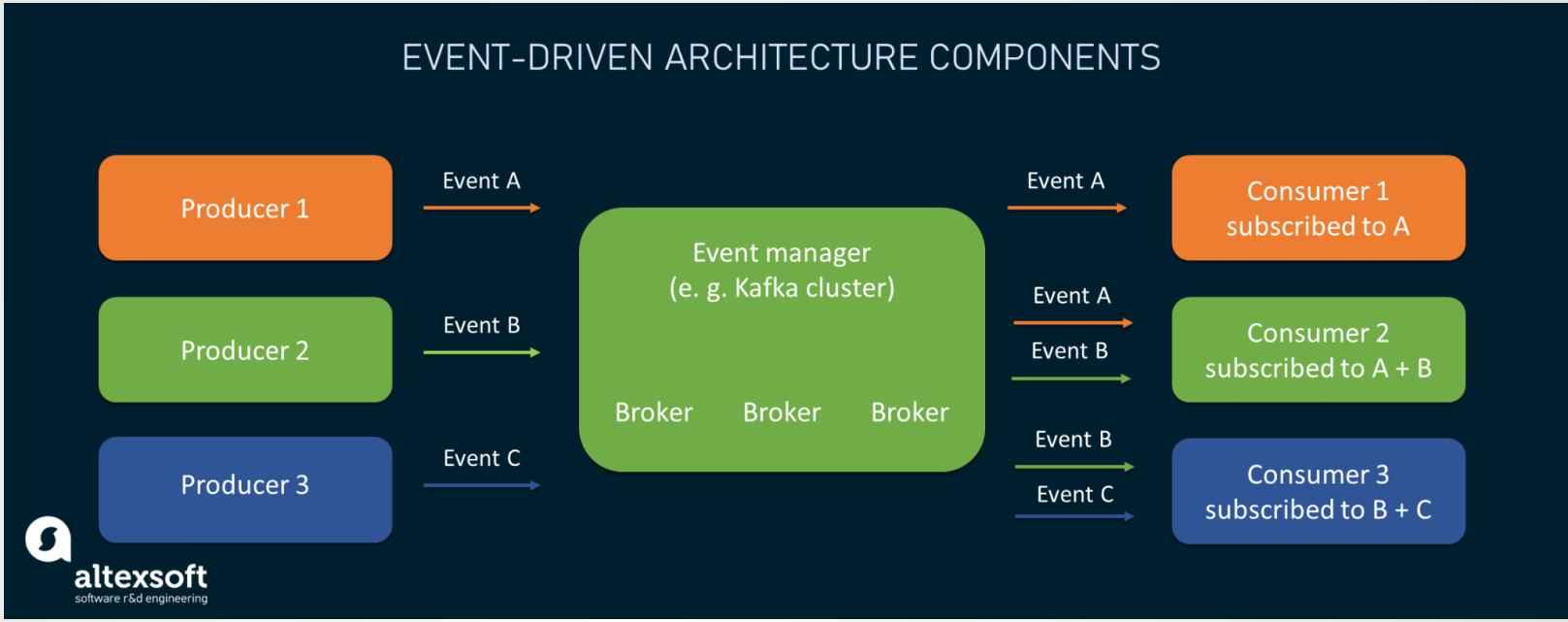
Nachteile

- Überschaubarkeit
  Große Systeme sind schwer zu überblicken
- Errorhandling
- Debugging
  Es gibt einen Formatwechsel zwischen den Services (eigentlich sogar 2: Domäne A auf Message auf Domäne B)

Event-Driven

- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

Event-Driven

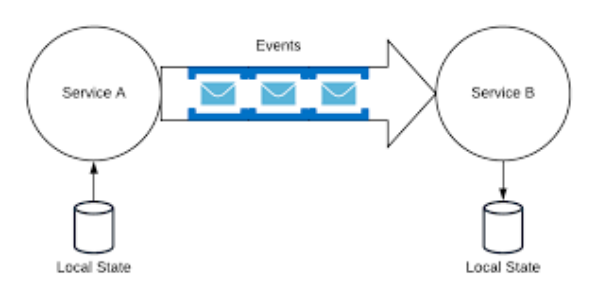


Clients subscriben **asynchron Events** (Topics) von **Brokern** und reagieren dann darauf.

Es gibt **keine vordefinierten Empfänger**!

Wird gerne mit **SOA** & als Erweiterung von **Message-Driven Architecture** verwendet.

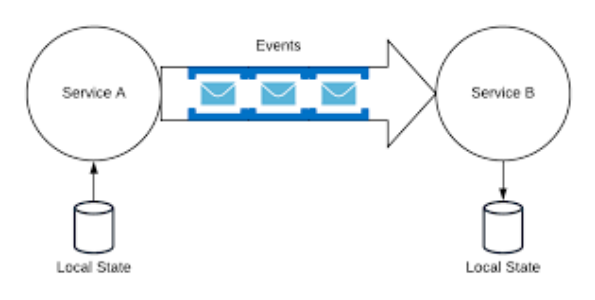
Event-Driven



Vorteile

- Lose Kopplung
  Über Message Queue
- Resilienz
  Über MessageQueue und Trennung
- Tolle Scalability
- Easy Fan-Out

Event-Driven



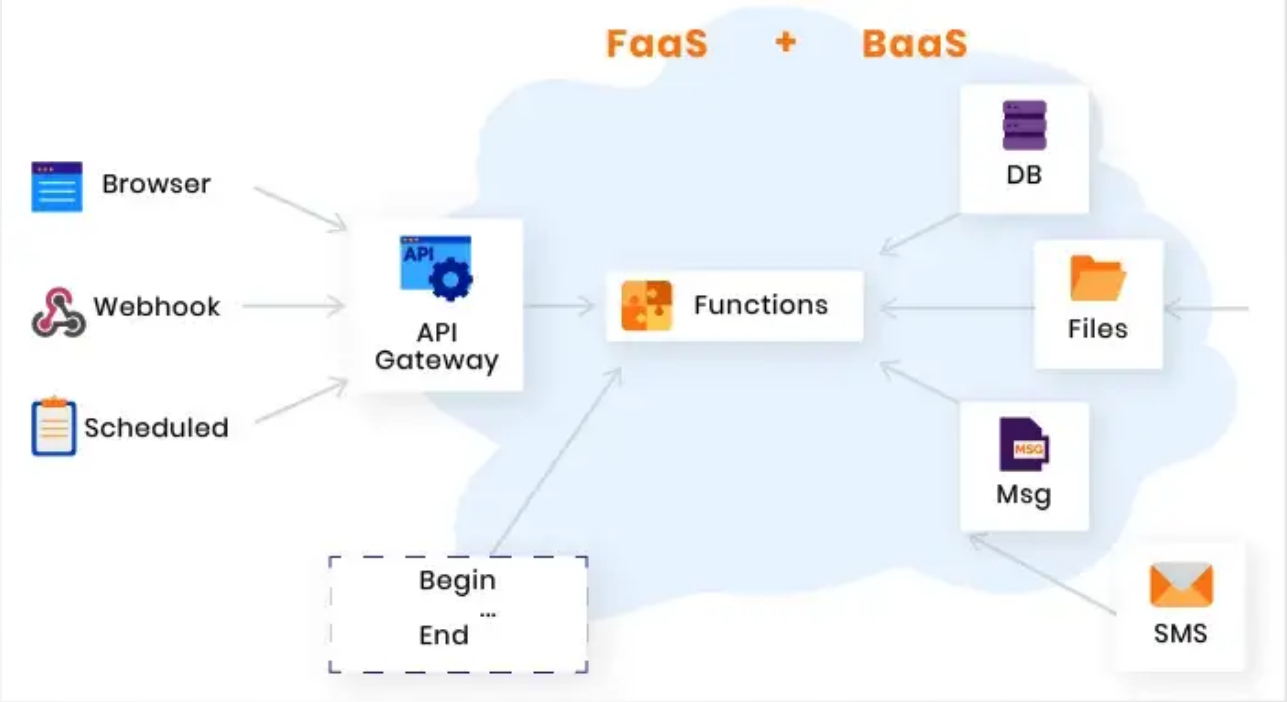
Nachteile

- Testbarkeit
  Es ist schwer vorauszusehen wer aller ein Event subscribed
- Überschaubarkeit
  Große Systeme sind schwer zu überblicken
- Errorhandling
- Debugging

Serverless

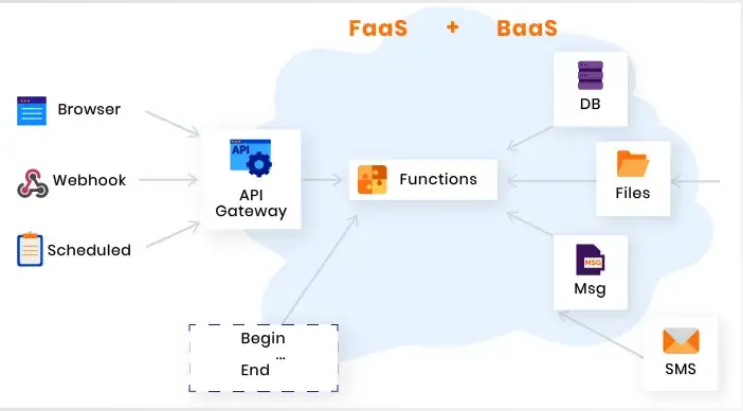
- Monolithic
- Client-Server Architecture
- Peer-to-peer Architecture (P2P)
- Service Oriented Architecture (SOA)
- MicroServices
- Message-Driven Architecture
- Event-Driven Architecture
- Serverless

Serverless



Ist ein Zusammenspiel zwischen Backend as a Service (**Baas**) und Function as a Service (**Faas**), das es erlaubt eine komplette Applikation zu erstellen, die nur auf der Hardware der jeweiligen Provider läuft.

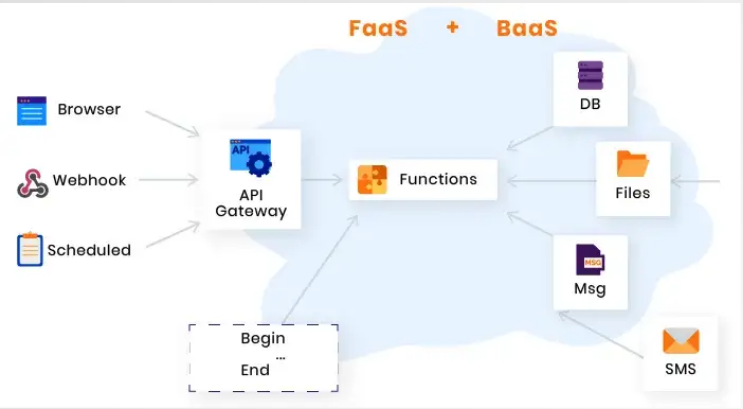
Serverless



Vorteile

- No Managing Infrastructure
- No Initial Hardware Costs
- Decoupling Modules Trough Events
- Application Scaling
- Short Time-To-Market
- Infrastructure Security

Serverless



Nachteile

- Price of Long-Running Workflows
- Cold-Start-Delay
- Provider-Dependency
- Higher Complexity
- Kleine Module ev. auf mehrere Provider verteilt (deployment, versioning, testing)
- Less Flexibility
  Bietet mein Provider das an? ...

References

- https://en.wikipedia.org/wiki/Monolithic_application
- https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith
- https://en.wikipedia.org/wiki/Client%E2%80%93server_model
- https://en.wikipedia.org/wiki/Peer-to-peer
- https://en.wikipedia.org/wiki/Service-oriented_architecture
- https://en.wikipedia.org/wiki/Microservices
- https://en.wikipedia.org/wiki/Event-driven_architecture
- https://en.wikipedia.org/wiki/Serverless_computing
- https://medium.com/@JalelTounsi/monolith-soa-microservices-or-serverless-43dd60e29756
- https://aws.amazon.com/microservices/