CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Microservices

(3) Integrationsstile

Shared Database + REST

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 12

Microservice Architekturen

12.1 Eigenschaften von Microservices

12.2 Integrationsmuster für Microservices

- Datenbank-basierte Integration
- gRPC, REST
- API Versioning
- Event-Driven Architectures

12.3 Architekturelle Sicherheit

- Circuit-Breaker, Bulkhead
- Idempotente API-Operationen

12.4 Skalierung von Microservices

- Load Balancing
- Messaging
- Scaling for Reads/Writes
- Command Query Responsibility Segregation (CQRS)
- Caching

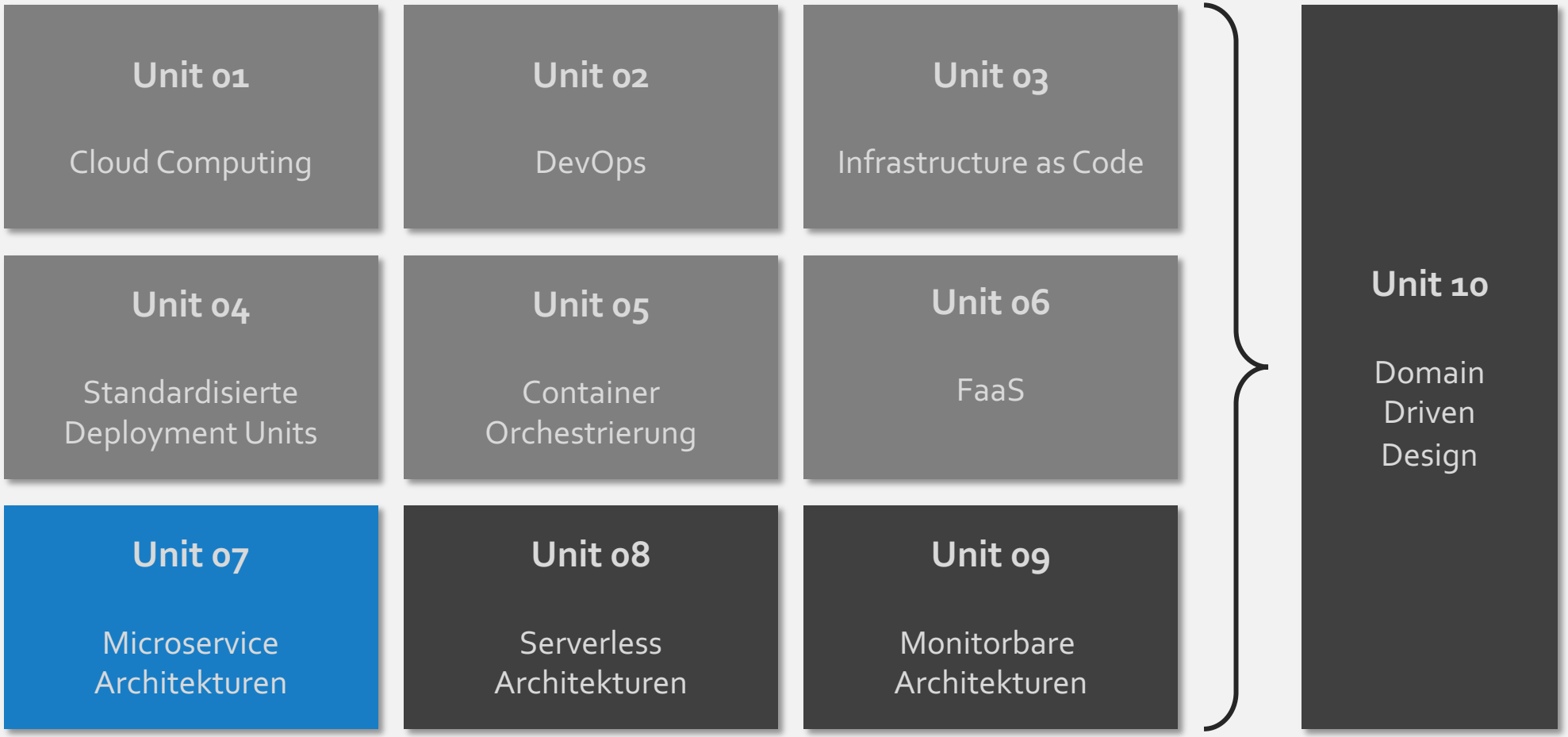
12.5 Prinzipien zur Entwicklung von Microservices

12.6 Serverless-Architekturen

- Architekturelle Konsequenzen von Serverless-Limitierungen
- Das API-Gateway Pattern
- Abgrenzung zu Microservices

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



INHALTE

CLOUD NATIVE COMPUTING

Microservices

- Was ist das für ein Architekturstil?
- Vor- und Nachteile von Microservices

Randbedingungen

- Lose Kopplung und hohe Kohäsion
- Bounded Context und Domain Driven Design
- Conways Law
- Service Ownership und Team-Strukturen

Integration

- Shared Databases
- (g)RPC, REST, GraphQL
- Messaging (Event-driven)

Skalierung von Microservices

- Failure is Everywhere
- Architectural Safety Measures
- Scaling
- Caching

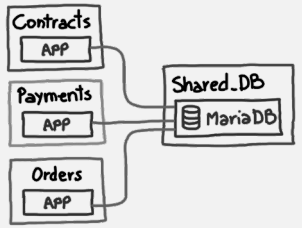
7 Prinzipien von Microservices

- Model Around Business Concepts
- Adopt a Culture of Automation
- Hide Internal Implementation Details
- Decentralize All the Things
- Independently Deployable
- Isolate Failure
- Highly Observable

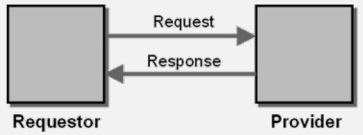
INTEGRATION

Integrationsstile

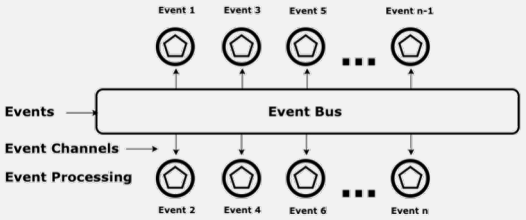
- **Daten-basierte Integration**
  - The shared database



- Request/Response-basierte Integration (synchron)
  - Remote Procedure Calls
  - REST
  - GraphQL



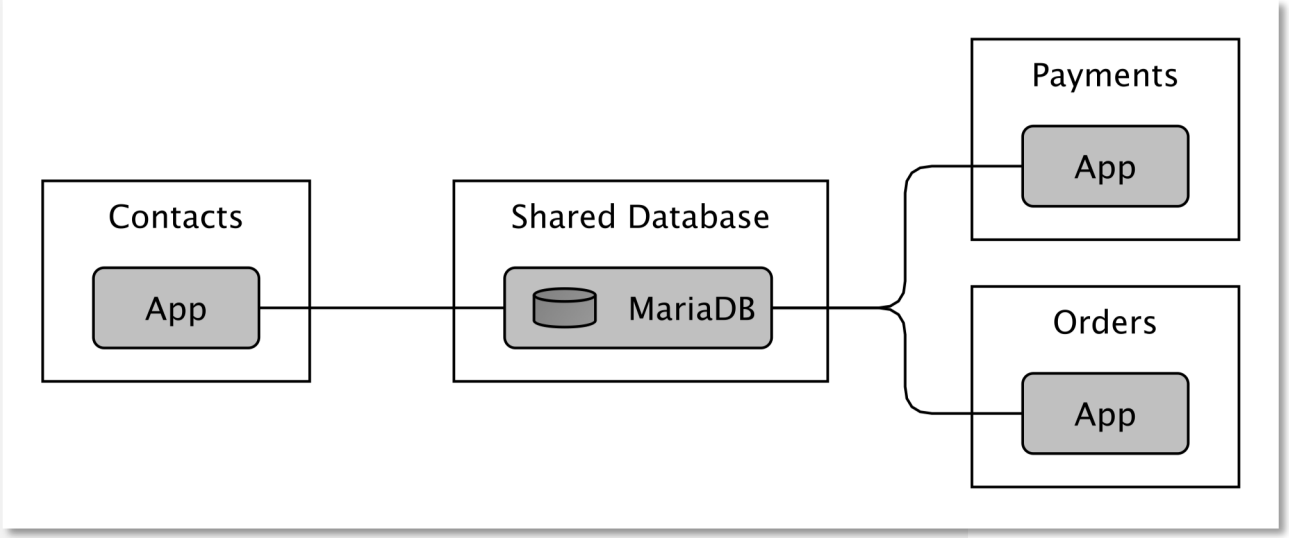
- Ereignis-basierte Integration (asynchron)
  - Asynchrone Architekturen
  - Reactive Programming



INTEGRATION

Daten-basierte Integration (Shared Database-Pattern)

- Datenbankintegration ist eine häufige Form der Integration
- Services können Informationen von einem Stateful Service über eine **gemeinsame Datenbank** abrufen
- Daten können in einer Datenbank aktualisiert oder erstellt werden
- Datenbankintegration ist schnell und beliebt
- Dieses Muster ist jedoch mit **Schwierigkeiten** behaftet

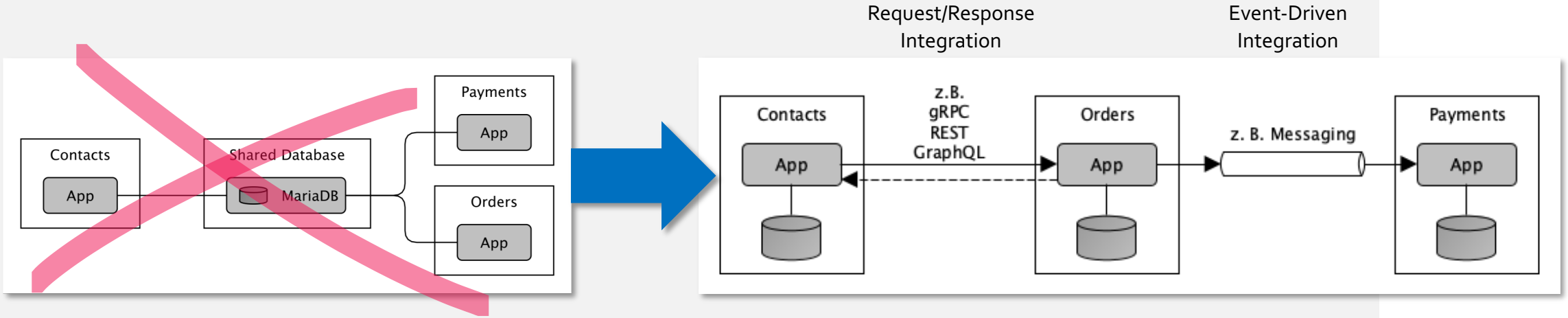


- Consuming Services werden gezwungen, sich an interne Implementierungsdetails zu binden
- Datenstrukturen der Datenbank werden vollständig geteilt
- Änderungen im Datenbankschema können Änderungen in allen Consuming Services nach sich ziehen
- Erhöht die Kopplung zwischen den Services

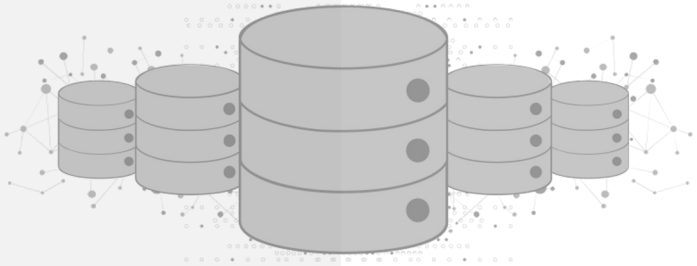
- Services kaum isoliert und autonom weiterentwickelbar
- Consuming Services sind an eine bestimmte Datenbanktechnologie gebunden
- Verwendung datenbankspezifischer Treiber kann bei Datenbankänderung problematisch werden

INTEGRATION

Vermeidung des Shared Database-Patterns



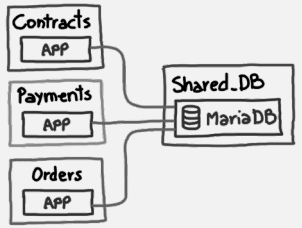
- Zur Reduktion der Kopplung verzichtet man daher auf eine zentrale Datenbank
- Jeder Service erhält seine eigene Datenbank im Sinne einer **polyglotten Persistenz**
- Der Datenaustausch erfolgt nun nicht mehr über die gemeinsame Datenbasis sondern muss über Protokolle zwischen den Services erfolgen
- Hier kann man grundsätzlich folgende Integrationspattern unterscheiden
  - Request/Response
  - Ereignis-gesteuert
- Diesen Integrationsmöglichkeiten haben einen unterschiedlichen **Einfluss auf den Grad der Kopplung**



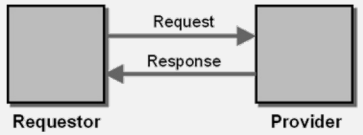
INTEGRATION

Integrationsstile

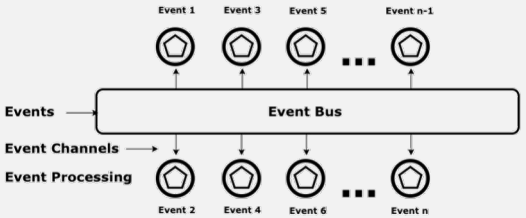
- Daten-basierte Integration
  - The shared database



- **Request/Response-basierte Integration**
  - Remote Procedure Calls
  - REST
  - GraphQL

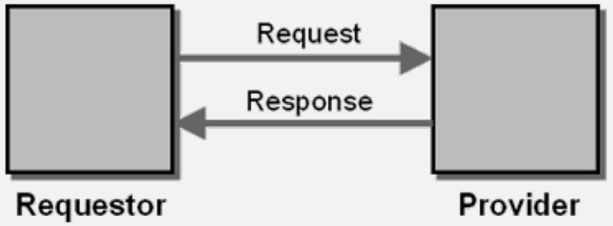


- Ereignis-basierte Integration
  - Event Driven Architecures
  - Reactive Programming



INTEGRATION

Request/Response-basierte Integration (synchron)



UDP TCP/HTTP

RPC

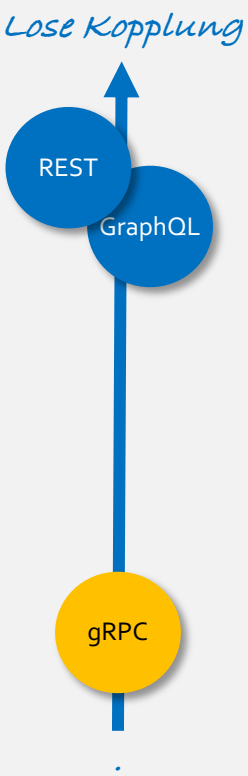
gRPC

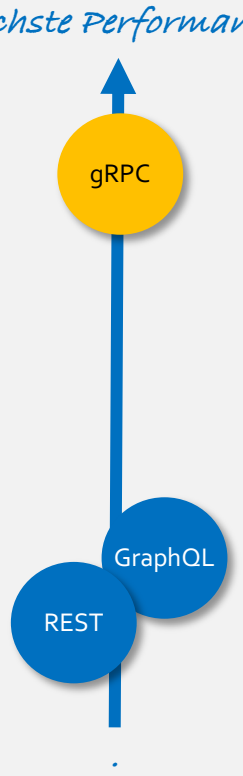
GraphQL

REST

IDL oder Schema
(erhöhen Kopplung zw. Komponenten)

Keine IDL erforderlich
(nur bei HATEOAS, ansonsten implizit)





IDL: Interface definition language

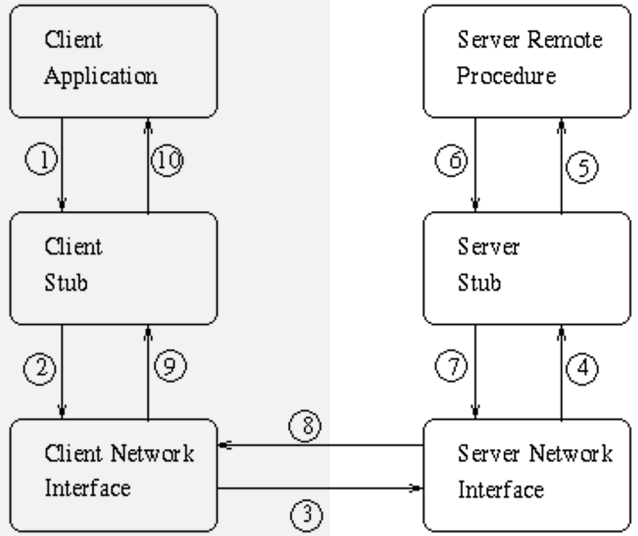
Häufig muss man also zwischen Performance und loser Kopplung abwägen.

Achtung:
Es gilt auch die „ewige" Empfehlung erst Performanceprobleme anzugehen, wenn es einen aus dem Betrieb ersichtlichen Grund dazu gibt!

INTEGRATION

Request/Response-basierte Integration: Remote Procedure Calls (RPC)

- **Remote Procedure Calls (RPC)** sind eine Technik zur **Interprozesskommunikation** und ermöglichen den Aufruf von Funktionen in anderen Adressräumen von Prozessen
- Aufgerufene Funktionen werden normalerweise auf einem anderen System ausgeführt. Der Remote-Aufruf unterscheidet sich (syntaktisch) kaum vom Aufruf lokaler Funktionen
- Mehrere Implementierungen von RPC existieren, die in der Regel nicht kompatibel sind. Oft müssen sogar Client-/Server-Prozesse in derselben Sprache entwickelt sein
- RPC setzt meist auf **UDP** auf, was Performance-Vorteile bietet
- Client-/Server-**Stubs** kapseln server- und clientseitige Vorgänge und werden aus **Interface Definitions (IDL)** generiert



Weil es für Cloud-native Anwendungen eher untergeordneten Charakter hat, betrachten wir RPC in seiner Reinform nicht weiter.

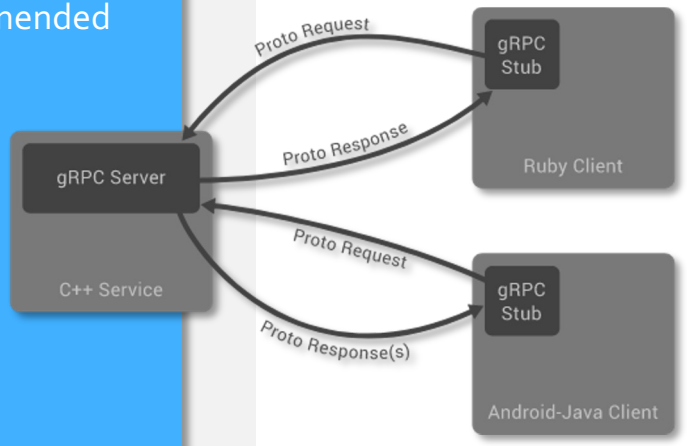
INTEGRATION

Request/Response-basierte Integration: gRPC Remote Procedure Calls

- gRPC (gRPC Remote Procedure Calls) basiert auf HTTP/2 und Protocol Buffers und ist ein Projekt der Cloud Native Computing Foundation.
- Mittels einer IDL wird die Schnittstelle unabhängig von einer konkreten Programmiersprache spezifiziert.
- Mittels des Protocol-Buffer-Compilers 'protoc' kann aus dieser Beschreibung Server- und Client-Code, sogenannte Stubs, generiert werden.

Language	OS	Compilers / SDK
C/C++	Linux, Mac	GCC 4.9+, Clang 3.4+
C/C++	Windows 7+	Visual Studio 2015+
C#	Linux, Mac	.NET Core, Mono 4+
C#	Windows 7+	.NET Core, NET 4.5+
Dart	Windows, Linux, Mac	Dart 2.2+
Go	Windows, Linux, Mac	Go 1.13+
Java	Windows, Linux, Mac	JDK 8 recommended
Kotlin/JVM	Windows, Linux, Mac	Kotlin 1.3+
Node.js	Windows, Linux, Mac	Node v8+
Objective-C	macOS 10.10+, iOS 9.0+	Xcode 7.2+
PHP	Linux, Mac	PHP 7.0+
Python	Windows, Linux, Mac	Python 3.5+
Ruby	Windows, Linux, Mac	Ruby 2.3+

Da gRPC auf HTTP (und somit auf Anwendungsebene definiert ist) und nicht UDP (wie RPC) beruht, ist es einfacher zu handhaben (bspw. mittels Kubernetes Ingress Rules).



Quelle: https://www.grpc.io

INTEGRATION

Request/Response-basierte Integration: gRPC Protocol Buffers

gRPC verwendet Protocol Buffers zum Serialisieren strukturierter Daten.

Hierzu müssen die zu serialisierenden Daten mittels einer **Interface Definition Language (IDL)** definiert werden.

Protocol Buffer Daten sind als Nachrichten strukturiert, wobei jede Nachricht aus einer Liste von Feldern (Key-Value Paare) besteht, die den Namen und den Datentyp inkl. Serialisierungsposition angeben.

Mittels des Protocol Buffer Compiler `protoc` lassen sich Protokolldefinitionen in Datenzugriffsklassen unterstützter Programmiersprachen generieren.

Nachricht

```
message Person {
  string name = 1;
  int32 id = 2;
  bool has_ponycopter = 3;
}
```

Service Definition + Nachrichten

```
// The greeter service definition.

service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Quelle: https://www.grpc.io

INTEGRATION

Request/Response-basierte Integration: gRPC Streaming

CLOUD NATIVE COMPUTING

1 **Unary RPCs**, bei denen der Client einen einzelnen Request an einen Service sendet und eine einzelne Response zurückerhält, genau wie bei einem normalen synchronen Funktionsaufruf.

```
rpc SayHello(HelloRequest)
returns (HelloResponse);
```

2 Bei **Server-Streaming-RPCs** sendet der Client einen Request an einen Service und erhält als Antwort einen Stream von Nachrichten. Der Client liest aus dem Rückgabe-Stream, bis keine Nachrichten mehr vorhanden sind. gRPC garantiert die Reihenfolge der Nachrichten innerhalb eines einzelnen RPC-Requests.

```
rpc LotsOfReplies(HelloRequest)
returns (stream HelloResponse);
```

3 Bei **Client-Streaming-RPCs** sendet der Client einen Stream von Nachrichten an den Service. Sobald der Client die Nachrichten fertig geschrieben hat, wartet er darauf, dass der Server sie liest und seine Response zurückgibt. gRPC garantiert die Reihenfolge der Nachrichten innerhalb eines einzelnen RPC-Requests.

```
rpc LotsOfGreetings(stream HelloRequest)
returns (HelloResponse);
```

4 Bei **Bidirectional Streaming-RPCs** senden beide Seiten Streams von Nachrichten. Die beiden Streams arbeiten unabhängig voneinander, sodass Clients und Server in beliebiger Reihenfolge lesen und schreiben können. Die Reihenfolge der Nachrichten in jedem Stream bleibt erhalten.

```
rpc BidiHello (stream HelloRequest)
returns (stream HelloResponse);
```

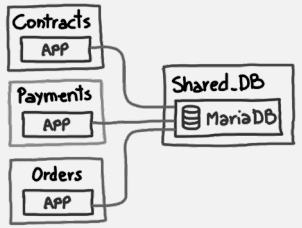
gRPC lässt sich auch asynchron, d.h. non-blocking, einsetzen.

Quelle: https://www.grpc.io

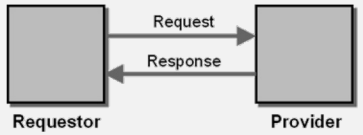
AUSBLICK

Integrationsstile

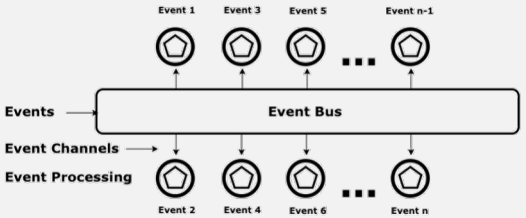
- Daten-basierte Integration
  - The shared database



- Request/Response-basierte Integration
  - Remote Procedure Calls
  - **REST**
  - **GraphQL**



- Ereignis-basierte Integration
  - Event Driven Architectures
  - Reactive Programming



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de