

# MULTI-CLOUD SOFTWARE ARCHITECTURE AND DESIGN

Christoph Bussler, Robert Bosch LLC, USA

IT Days 2022, Cluj-Napoca, Romania, November 10<sup>th</sup>, 2022



# Definition of multi-cloud

## ► Hybrid cloud vs. multi-cloud

► <https://www.vmware.com/topics/glossary/content/hybrid-cloud-vs-multi-cloud.html>:

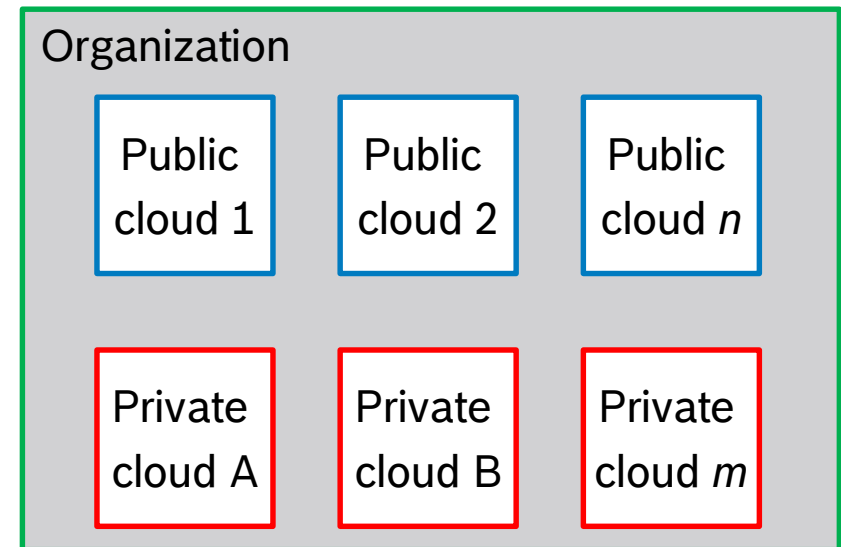
- **Hybrid clouds** always include a private cloud and are typically managed as one entity
- **Multi-clouds** always include more than one public cloud service, which often perform different functions. Multi-clouds do not have to include a private cloud component, but they can, in which case they can be both **multi-cloud and hybrid cloud**.

## ► Private cloud

- Synonym: on-premises

## ► For the purpose of this presentation

- An organization uses two or more clouds, public and/or private



# Definition of cloud

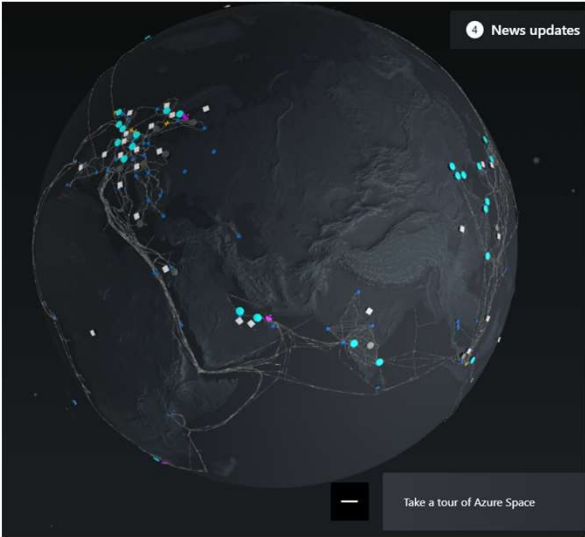
- ▶ For purposes of this presentation
  - ▶ A cloud is a set of regions
  - ▶ A region is composed of several zones



<https://cloud.google.com/about/locations>



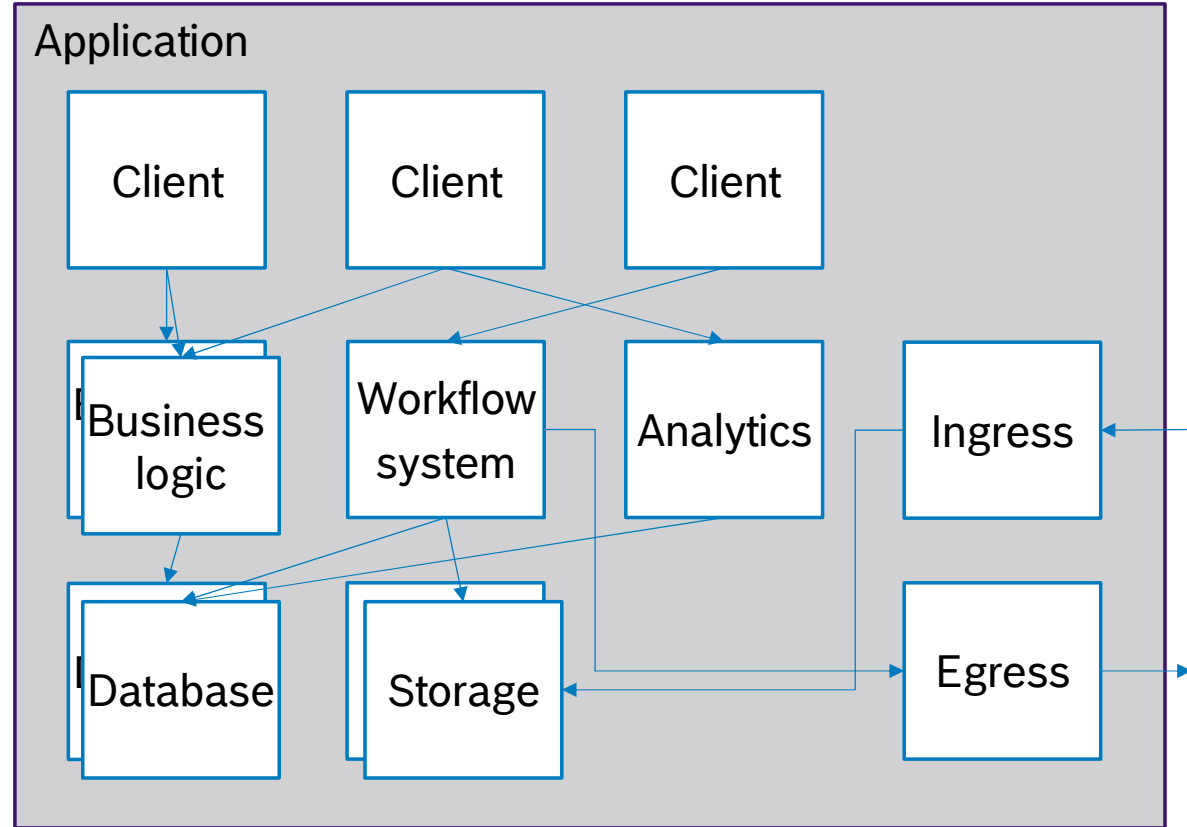
<https://aws.amazon.com/about-aws/global-infrastructure/>



<https://infrastructuremap.microsoft.com/explore>

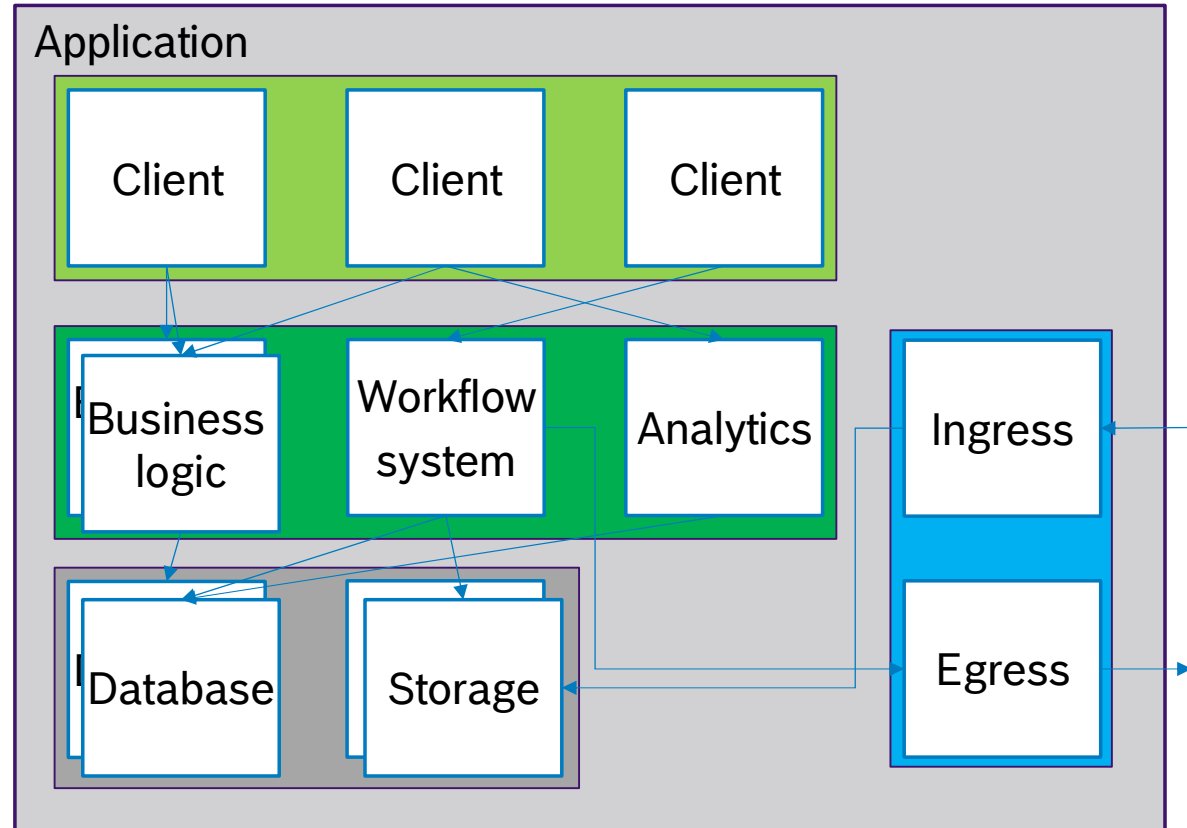
# Prototypical application

- ▶ Application
  - ▶ Several components
  - ▶ Invocation relationships
  - ▶ Interaction with other systems
- ▶ Additional components possible, not mentioned for simplicity
  - ▶ Messaging
  - ▶ API gateways
  - ▶ Functions (e.g., AWS Lambda)
  - ▶ ...



# Prototypical application

- ▶ Application
  - ▶ Several components
  - ▶ Invocation relationships
  - ▶ Interaction with other systems
- ▶ Additional components possible, not mentioned for simplicity
  - ▶ Messaging
  - ▶ API gateways
  - ▶ Functions (e.g., AWS Lambda)
  - ▶ ...

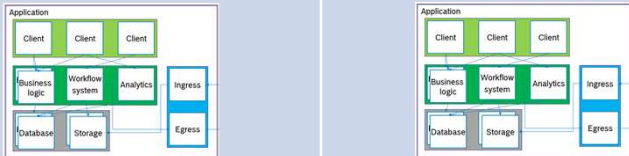
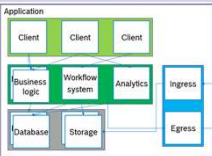
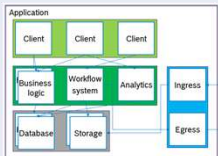
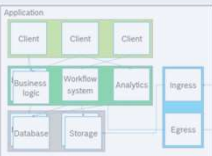
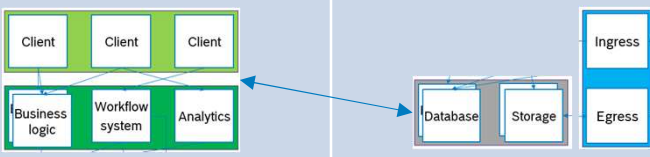
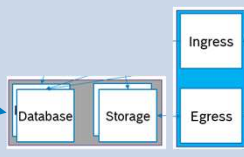
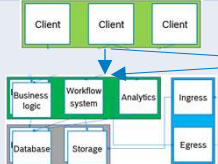
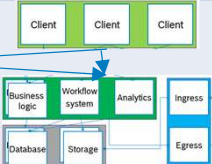


# Prototypical application – wait, there is more!

- ▶ Application architecture aspects often not shown on architecture diagrams
  - ▶ Security
  - ▶ Networking
  - ▶ Security key management
  - ▶ Development environments
  - ▶ Continuous integration / continuous deployment
  - ▶ Cloud projects, subscriptions
  - ▶ User account and organization management
  - ▶ ...
- ▶ Spoiler
  - ▶ Every (!) single (!) aspect will be in focus when architecting and designing multi-cloud applications

# Multi-cloud architecture patterns: structure

Cloud 3 Cloud 4

Architecture type	Cloud 1	Cloud 2	Notes
Portable application			Same code can be deployed on all clouds that are in scope
Migrate-able application			Code deployed at one cloud, expected to run in another “easily” with little additional engineering effort
Distributed application			Different parts of application deployed on different clouds and communicate
Highly available application			Each client can access any logic layer (any cloud to any cloud)

# Multi-cloud architecture patterns: behavior

Cloud 3 | Cloud 4

Architecture type	Cloud 1	Cloud 2	Notes
Disaster recoverable application			Hot, warm, cold standby of equivalent system in another cloud
Resource bursting application			Based on dynamic need, additional resources in another cloud are used
Partitioned application			Clients directed to cloud that stores “their” data



# Portable multi-cloud architecture pattern

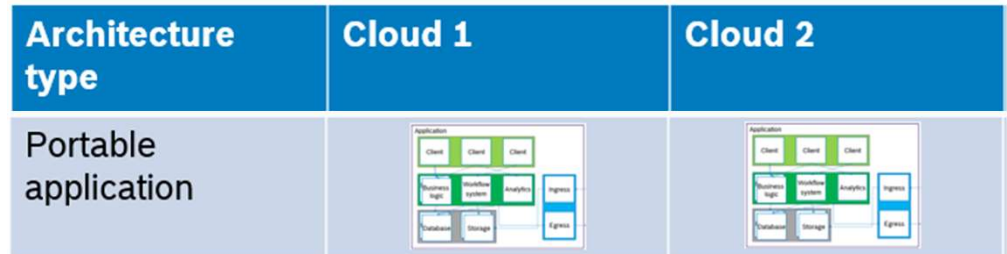
- ▶ “Same” application can be deployed into different clouds

- ▶ Spectrum of “same”

- ▶ Cloud agnostic: same bits, meaning same code, and same components
- ▶ Cloud proprietary: same functionality, but implemented by specialization with services specific to a cloud

- ▶ Example

- ▶ Cloud agnostic: PostgreSQL (but not 100% for specific extensions)
- ▶ Cloud proprietary: Google Cloud Cloud Spanner, Azure CosmosDB, Amazon DocumentDB



# Note on architecture component spectrum

## ▶ Discourse

- ▶ Agnostic: exact same behavior, one code base, might require installation and operations
- ▶ Proprietary: different behavior causes tendency to limit to least common denominator, benefit might be different on clouds (e.g., scale, management, operations)

## ▶ Example decisions

- ▶ Cloud agnostic: PostgreSQL
  - Some extensions are not available in all managed PostgreSQL cloud services: requires non-managed approach
- ▶ Cloud proprietary: Google Cloud Cloud Spanner, Azure CosmosDB, Amazon DocumentDB
  - What is the common denominator that can be used to implement abstraction so that application can run on any of those?

## ▶ Exercise – what would you do?

- ▶ Blob storage
- ▶ Message queuing
- ▶ Main memory caching
- ▶ Load balancing
- ▶ Compute
  - VMs, Kubernetes
- ▶ Serverless functions

## ▶ And

- ▶ Security
- ▶ Networking
- ▶ ...

# Migrate-able multi-cloud architecture pattern

- ▶ “Same” application might be deployed into another cloud in the future

- ▶ Risk

- ▶ To make this future migration possible, the application design
  - Is as cloud agnostic as possible from its beginning
  - Or has abstractions built in that makes it possible to add proprietary services in a structured way
- ▶ The risk is that the two bullets above are intentions that are not being tested in the CI and CD pipelines and on the target clouds
  - Expectations of relative “ease of migration” are raised, but not implemented and tested for

- ▶ In the best case, migration is easy and swift

- ▶ In the worst case, a significant re-design and re-implementation might be required

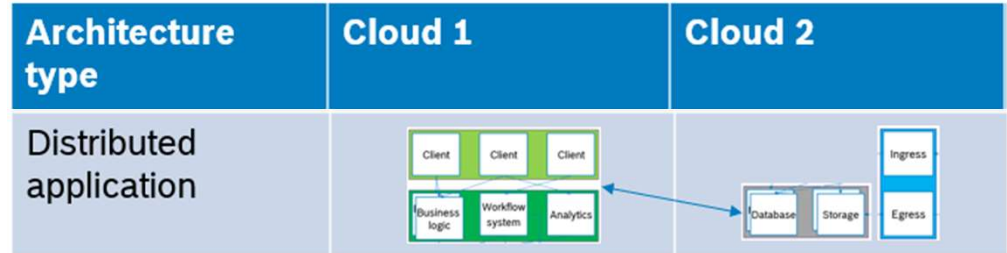


# Distributed multi-cloud architecture pattern

- ▶ Different components of the application are deployed into different clouds and have remote invocation relationships

- ▶ Units of distribution

- ▶ Infrastructure (example shown in the diagram)
  - Like queueing, database
  - Also dedicated “clouds” (e.g., MongoDB Atlas, Confluent)
- ▶ Business logic of application
  - Not only infrastructure, but also application logic
  - Like customer address management, accessible by an API
  - Also, SaaS services (Software-as-a-Service)

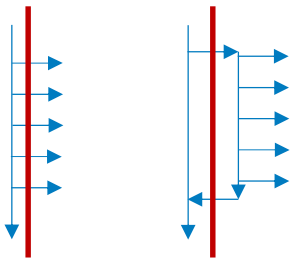


- ▶ Best of breed systems
  - Like analytics, machine-learning systems that are preferable on a specific cloud
- ▶ Hardware dependencies
  - Specific type of hardware that is only available on specific clouds
- ▶ ...

# Note on units of distribution

## ► Delegation

- Example: a workflow in cloud A invokes 5 steps in cloud B, one after another
- Alternative: a workflow in cloud A calls a sub-workflow in cloud B that in turn calls the steps locally
- Benefit
  - Less chatter, less cost
  - Easier to monitor, trace and possibly debug



## ► Cost

- Each invocation incurs egress cost twice
- Scaling up invocations means increasing communication cost

## ► “Chatter”

- Each invocation takes place across cloud crosses network and organizational boundaries
- Each invocation – if long-distance – adds speed-of-light latency
- Scaling might start becoming an operations topic, e.g., large amount of concurrent invocations
- Recoverability across clouds might be hard due to difference in monitoring, tracing, etc.
- Minimizing chatter might be a good design goal

# Note on state management

## ► Example

- The application maintains state in each cloud

## ► State

### ► Partitioned

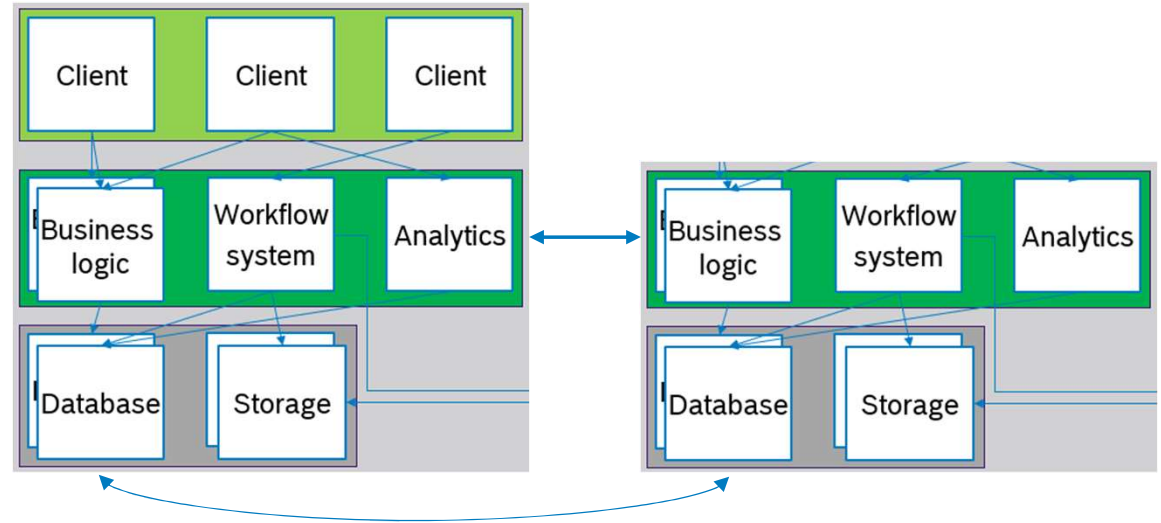
- State in databases has no relationship

### ► Consistent modification

- Update in one database is applied to other database in same transaction

### ► Asynchronous replication

- Update in one database is asynchronously replicated to the other database
- Delayed consistency



## ► Backup

- Consistency of backup
- Impossible if independent database instances are used (unless system is quiesced)

# Note on non-functional aspects

## ▶ Monitoring

- ▶ In a distributed deployment, where do you monitor?
- ▶ In only one cloud, in several clouds?

## ▶ Dashboarding

- ▶ Similar question for dashboards
- ▶ In one cloud, in several clouds?
- ▶ Or, in one cloud for overall view, and in particular cloud for the part of the application in that cloud?

## ▶ Misc.

- ▶ Knowledge and skill development
- ▶ Application architecture/design/implementation divergence
- ▶ ...

# Note on architecture and design fundamentals

- ▶ Key aspect of distributed multi-cloud application
  - ▶ Distributed system
  - ▶ Heterogeneous infrastructure
    - No assumption on similarity of behavior of cloud services possible
  
- ▶ “Simpler” distributed architecture
  - ▶ Distributed system
  - ▶ **Homogeneous** infrastructure
  - ▶ For example, different regions of the same cloud
    - Using the same cloud services in each region
    - But that is not multi-cloud anymore as defined earlier



# Highly available multi-cloud architecture pattern

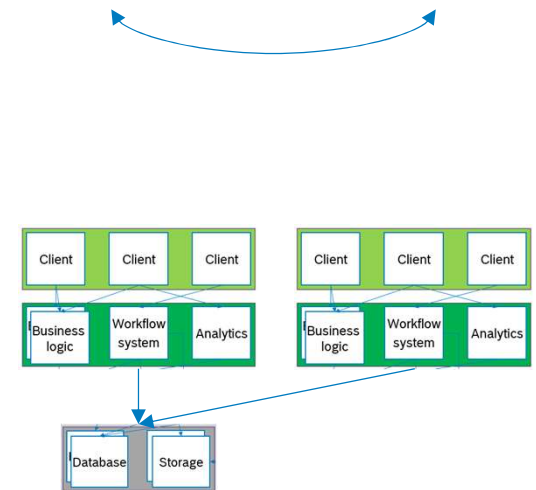
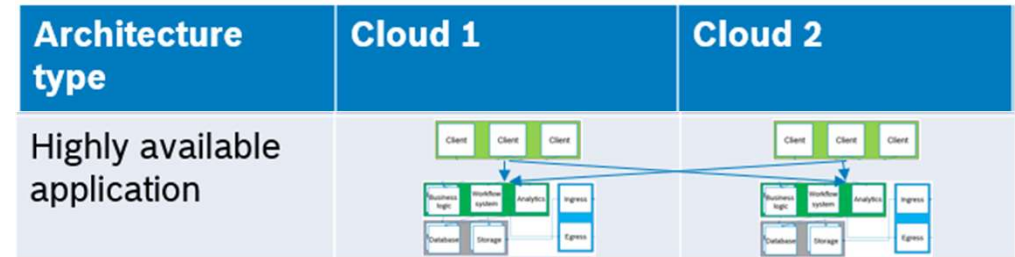
## ► Deployment

- Same application in several clouds
- Any client can access each cloud for each invocation (assuming stateless application)
- Application is available in failure case (e.g., one cloud cannot be reached anymore)

## ► Portable application

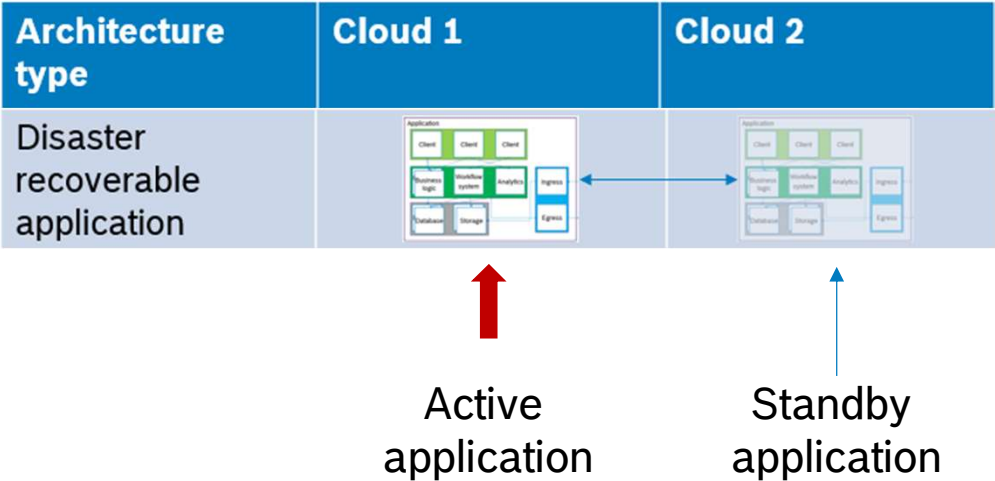
## ► State

- Consistent modification
- Replicated state
- Shared state
  - Located in one or the other cloud
- Not: partitioned, because it would not be HA anymore



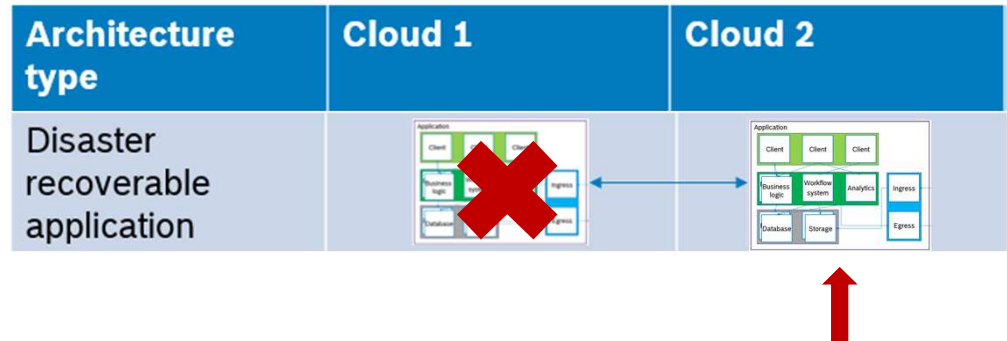
# Disaster recoverable multi-cloud architecture pattern

- ▶ Same application becomes available in cloud 2 if application fails in cloud 1



# Disaster recoverable multi-cloud architecture pattern

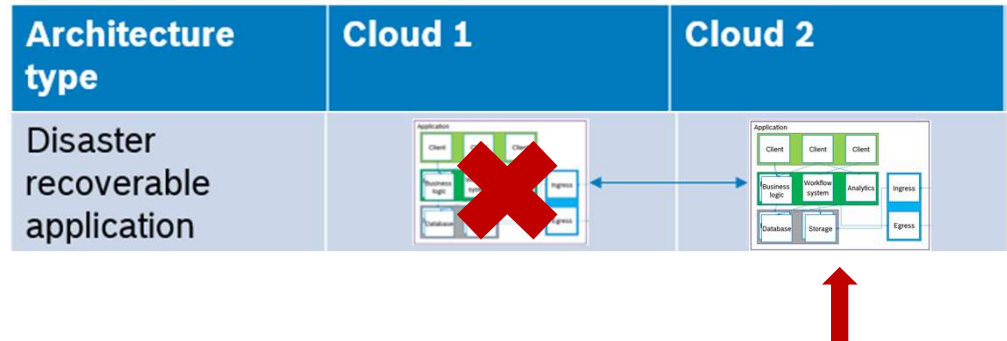
- ▶ Same application becomes available in cloud 2 if application fails in cloud 1
- ▶ Standby application
  - ▶ Hot (running, fully available to take over immediately)
  - ▶ Cold (must be installed first)
  - ▶ Warm (installed, but not running, must be started first)
- ▶ Process
  - ▶ Failover: if application fails, the standby takes over (active fails over to the standby)
  - ▶ Fallback (optional): if former active is available again, current active becomes standby
- ▶ Not multi-cloud, but similar
  - ▶ Disaster recovery within a cloud across regions



# Disaster recoverable multi-cloud architecture pattern

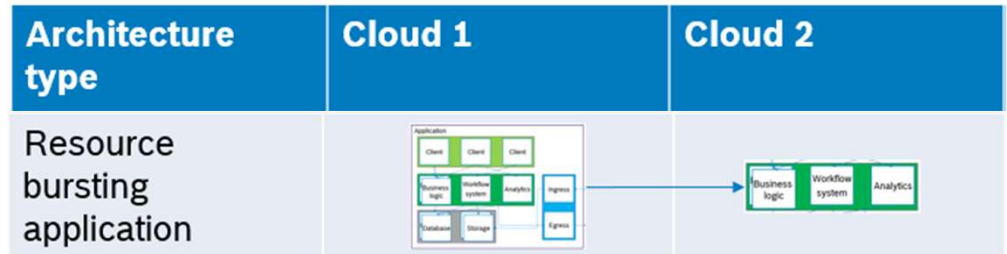
## ► Note on state management

- Consistency and flashback depending on how close the state of the standby matches that of the active application
- Example: weekly backup
  - Cloud 1: weekly backup
  - Cloud 2: when becoming active, backup might be up to one week old: up to one week flashback
- Example: continuous backup
  - Cloud 1: continuous backup
  - Cloud 2: when becoming active, in the best case no transaction is lost, but could be more than one
- Example: synchronous transactions
  - Cloud 1: synchronous transactions
  - Cloud 2: always same state as cloud 1



# Resource bursting multi-cloud architecture pattern

- ▶ Application is fully functional in cloud 1 and can access additional resource in cloud 2 as needed



- ▶ Full application bursting

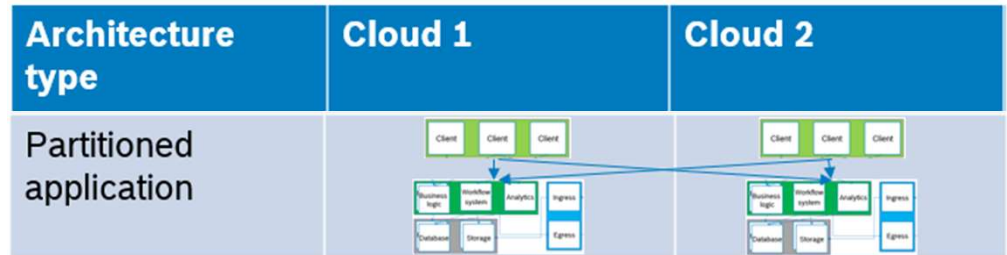
- ▶ The whole application will become available in a second cloud in addition to the first cloud
  - Will result in highly available multi-cloud application
  - Warning: state management will be important aspect: bursting state means consistent data management

- ▶ Partial application bursting (see figure)

- ▶ A part of the application will become available in a second cloud in addition to the first cloud
- ▶ For example, additional compute resources
- ▶ This requires the application to temporarily invoke additional resources on cloud 2

# Partitioned multi-cloud architecture pattern

- ▶ Partitioning not on an architecture level, but functionality or data level
- ▶ Data partitioning
  - ▶ Example: US data in cloud 1, EU data in cloud 2
    - Client access must be routed accordingly
  - ▶ Example: master data in one cloud, analytics and data archiving in another cloud
- ▶ Functionality partitioning
  - ▶ Example: Order to invoice process in cloud 1, payment to fulfillment in cloud 2
  - ▶ Example: Client facing functionality in cloud 1, back-office functionality in cloud 2



- ▶ Partitioning enforcement
  - ▶ Privacy laws require data in specific geographic region (data residency)
- ▶ “Natural” partitioning
  - ▶ Company operating in cloud 1 acquires company operation in cloud 2

# Now a more interesting multi-cloud application ...

- ▶ Multi-cloud application

# Now a more interesting multi-cloud application ...

- ▶ Multi-cloud application
  - ▶ Distributed



# Now a more interesting multi-cloud application ...

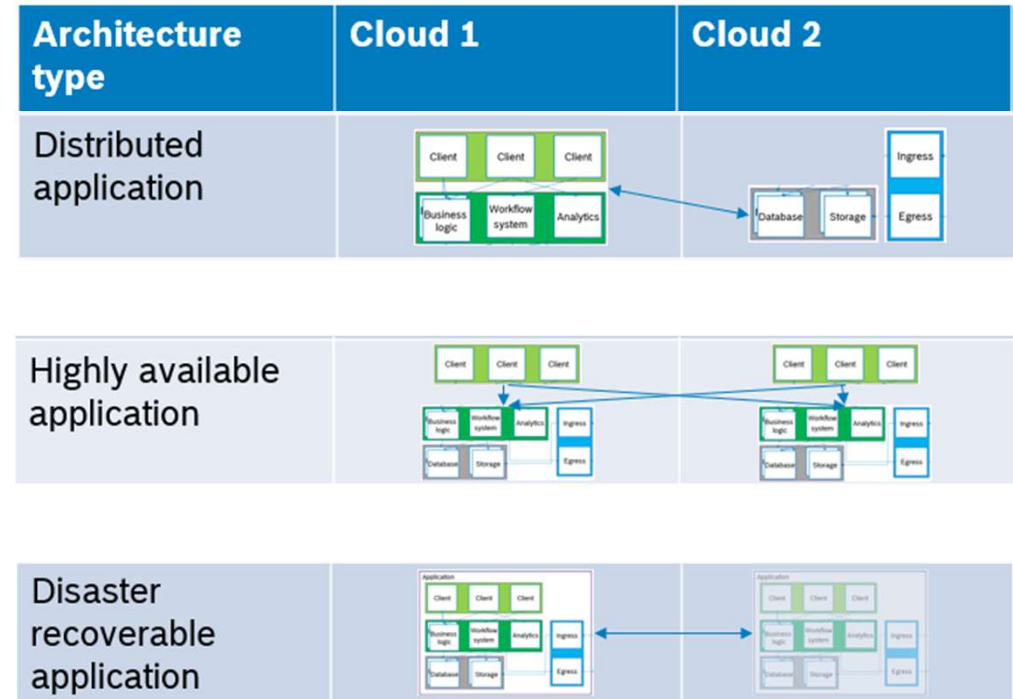
- ▶ Multi-cloud application
  - ▶ Distributed
  - ▶ Highly available

# Now a more interesting multi-cloud application ...

- ▶ Multi-cloud application
  - ▶ Distributed
  - ▶ Highly available
  - ▶ Disaster recovery

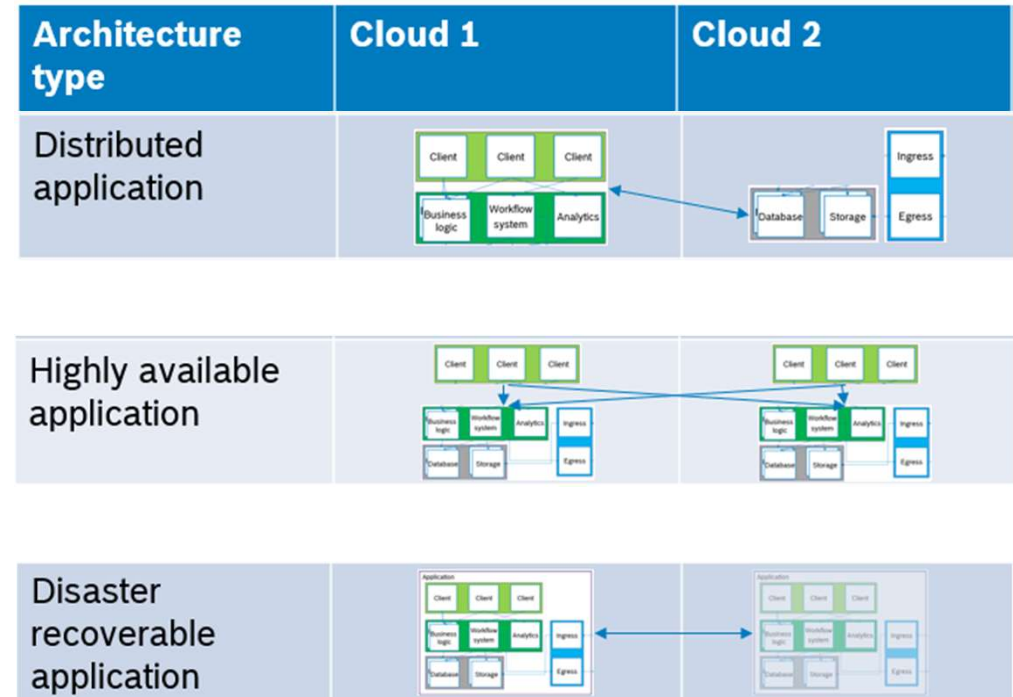
# Now a more interesting multi-cloud application ...

- ▶ Multi-cloud application
  - ▶ Distributed
  - ▶ Highly available
  - ▶ Disaster recovery



# Now a more interesting multi-cloud application ...

- ▶ Multi-cloud application
  - ▶ Distributed
  - ▶ Highly available
  - ▶ Disaster recovery
  
- ▶ How many clouds?
  - ▶ What do you think?



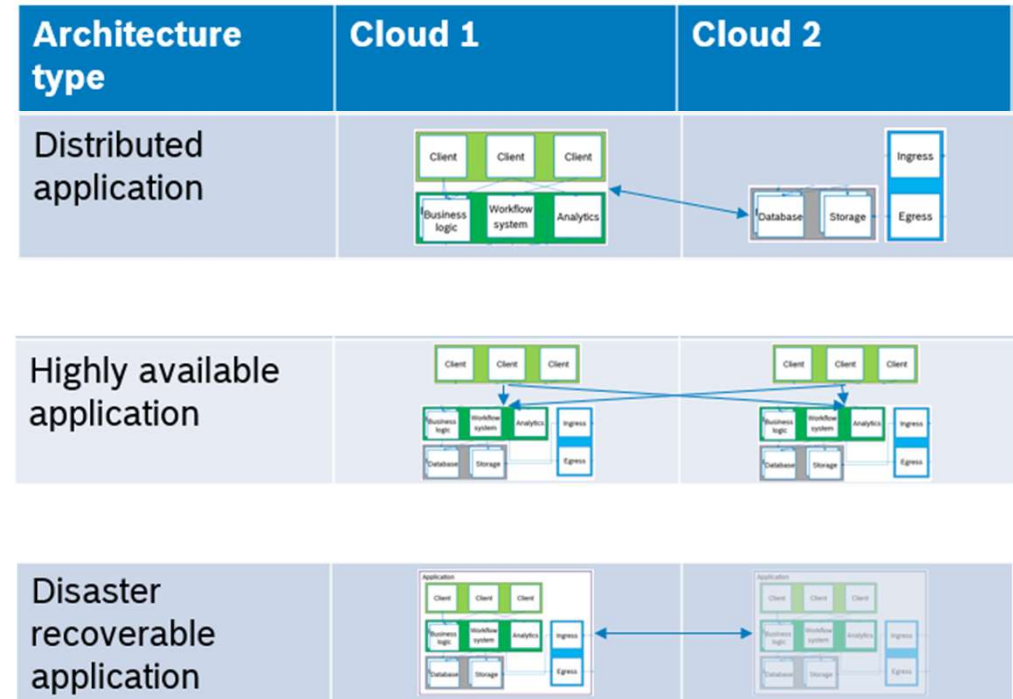
# Now a more interesting multi-cloud application ...

## ► Multi-cloud application

- Distributed
- Highly available
- Disaster recovery

## ► How many clouds?

- Disaster recovery: cloud 1, 2
  - After disaster in cloud1: cloud 2, 3 so that it is disaster recoverable again
- Distributed: cloud 1, 2
  - After disaster in cloud 1: cloud 2, 3
- Highly available: cloud 1, 2, 3, 4
  - Unless one cloud runs two parts
  - After disaster in cloud 1: cloud 2, 3, 4, 5



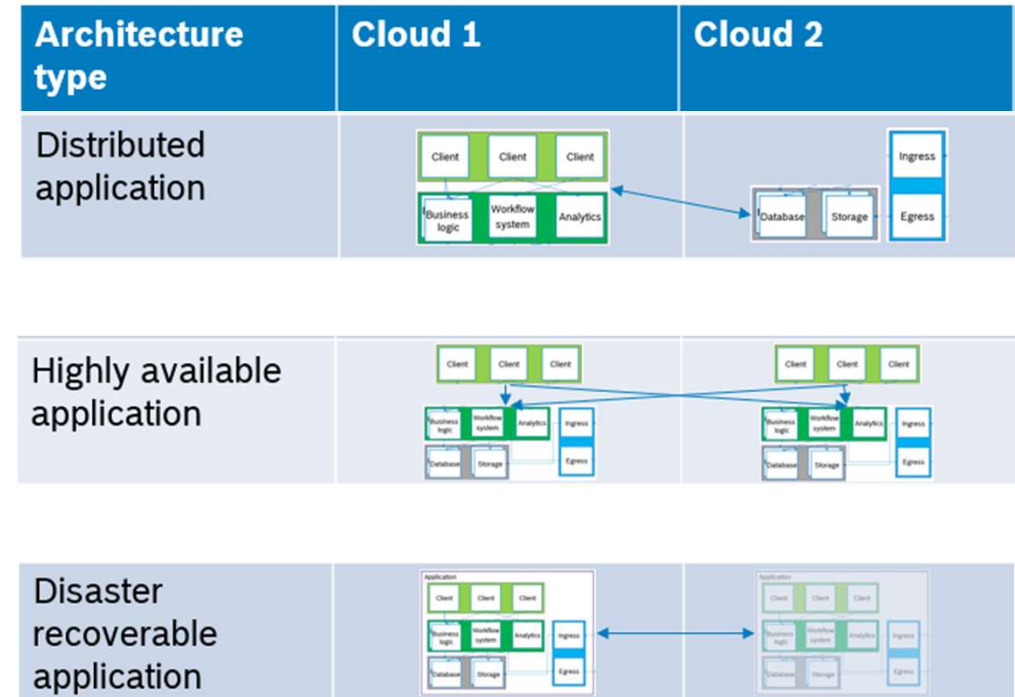
# Now a more interesting multi-cloud application ...

## ► Multi-cloud application

- Distributed
- Highly available
- Disaster recovery

## ► How many clouds?

- 5 does not sound right!
- What does disaster in cloud 1 mean?
  - Leaving a global cloud outage aside, the event to plan for is a region or zone outage
  - Failover in this situation does not necessarily mean failing over to another cloud, but to another region within the same cloud
- 2 clouds suffice
  - Unless application must be present on additional clouds



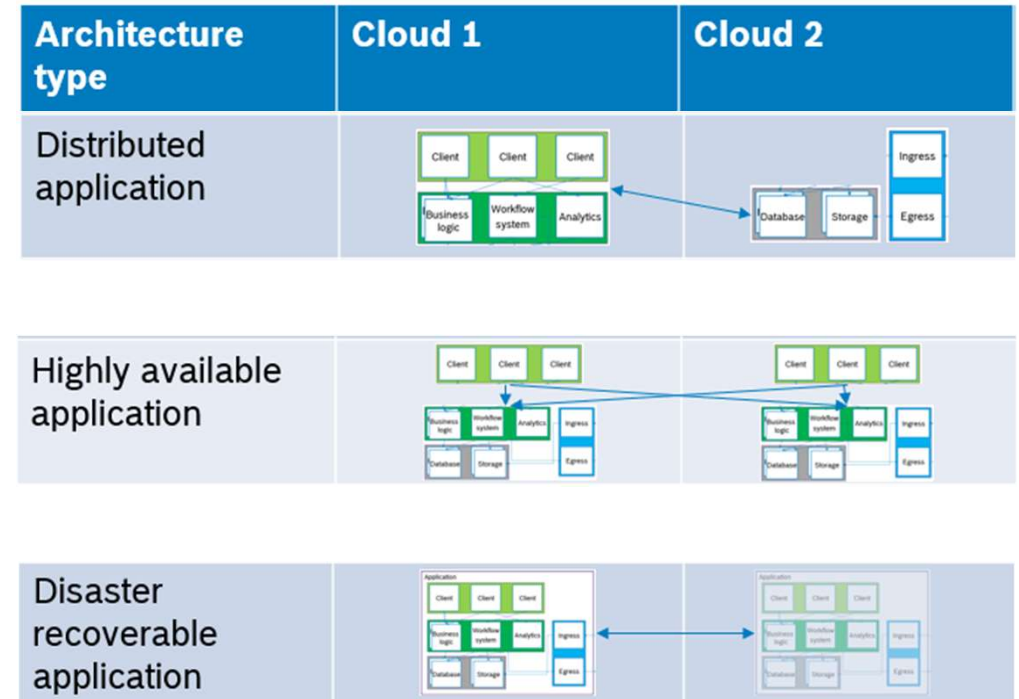
# Now a more interesting multi-cloud application ...

## ► Multi-cloud application

- Distributed
- Highly available
- Disaster recovery

## ► How many clouds?

- 5 does not sound right!
- What does disaster in cloud 1 mean?
  - Leaving a global cloud outage aside, the event to plan for is a region or zone outage
  - Failover in this situation does not necessarily mean failing over to another cloud, but to another region within the same cloud
- 2 clouds suffice
  - Unless application must be present on additional clouds



## ► Homework

- Which application or parts must be available on the 2 clouds and on how many regions within a cloud?

# Recurring architecture themes

- ▶ State management across clouds
  - ▶ Partitioned, synchronous, asynchronous
- ▶ Location of global functionality
  - ▶ Monitoring
  - ▶ Dashboarding
- ▶ Clouds, regions and zones
  - ▶ How to utilize the various scopes of independence in a (complex) multi-cloud application
- ▶ Cloud proprietary functionality by definition
  - ▶ Security, networking, ...
- ▶ Disaster recovery and high availability
  - ▶ Cloud, region and zone selection
  - ▶ Application and state management across these locations
- ▶ Expertise and skill set
  - ▶ Within cloud, across clouds



# Note on cloud abstraction

- ▶ Cloud abstraction is a desirable topic in multi-cloud application architecture
- ▶ Abstraction technologies (examples)
  - ▶ Networking
    - <https://aviatrix.com/cloud-network-platform/#multi-cloud-architecture/>
  - ▶ Security
    - <https://ermetic.com/platform/>
- ▶ Build your own abstractions
  - ▶ As needed by your applications

# Development, continuous integration and deployment

- ▶ CI and CD are systems in themselves
  - ▶ One CI system that integrates for all clouds, one CD system that deploys into all clouds that are in scope
  - ▶ For each cloud in scope: one CI and one CD system
- ▶ Relationship CI/CD and multi-cloud application architecture
  - ▶ Portable multi-cloud application
    - Possibility: one code base and one CI system
  - ▶ Distributed multi-cloud application
    - Possibility: one+ code bases, and one+ CI system (might be one each for each unit of distribution)
  - ▶ No right/wrong answer: decided by engineering organization
- ▶ Code base
  - ▶ One code base for all clouds in scope
  - ▶ One code base for each cloud in scope

# Drivers for multi-cloud: why?

## ▶ Business

- ▶ Meet the customers where they are
- ▶ Cost management
- ▶ Cost “relocation” (CapEx -> OpEx)
- ▶ Cost transparency
- ▶ Competition by cloud provider in business domain expected
- ▶ Acquisition of business (using a different cloud)
- ▶ Lock-in avoidance
- ▶ Flexibility, agility

## ▶ Engineering knowledge and skills

- ▶ Cost of re-training vs. cost of multi-cloud

## ▶ Technology

### ▶ Resources

- Best of breed
- Availability
- Capacity
- Bursting (short term)
- Managed services

### ▶ Failure mitigation

- Increase of SLAs values
- Disaster recovery (failover)

## ▶ Legal / regulatory

- Regulatory requirements (2 regions in one country from different clouds)

## ▶ It is cool 😊

# Drivers for multi-cloud: why?

Driver	Architecture
Meet the customers where they are	Portability: present in all clouds where customers are
Acquisition of business (using a different cloud)	Distribution: use applications without first migrating
Competition by cloud provider in business domain expected	Portability: fast switch between clouds as necessary
...	...

- ▶ One driver
  - ▶ Might have different possible architecture implementations
- ▶ Several drivers
  - ▶ Might require a combination of architecture patterns
  - ▶ For example
    - Meet the customers where they are -> portable
    - Reliability policy requirement -> high available architecture
    - Sum: Portable high available architecture

# To multi-cloud or not to multi-cloud?

- ▶ Degrees of freedom of choice based on drivers
  - ▶ E.g., existing on-premises data center, but expansion necessary
    - Choice: cloud complement or expand on-premises or move to cloud completely
  - ▶ E.g., existing cloud, edge devices must be integrated
    - No choice: integrate edge into cloud
  - ▶ E.g., regional application in one cloud, must be made disaster resilient
    - Choice: same cloud, different cloud
    - No choice: must be two regions in same country: might not be provided by one cloud alone
- ▶ Don't get locked up into avoiding lock-in (Gregor Hohpe)
- ▶ Portable or even distributed multi-cloud applications require significant and long-term effort
  - ▶ For each cloud: cloud cost, skills, development environments, testing, ...

# Closing

- ▶ Multi-cloud application architecture and design
  - ▶ Technically challenging and fascinating (constantly 😊)
  - ▶ Organizationally a significant investment requiring long-term commitment
- ▶ My recommendation
  - ▶ Make a conscious decision to go or not to go multi-cloud
    - In the voluntary as well as involuntary case
    - Consider tradeoffs and choices: e.g., migration of an application might be “easier” than a long-term multi-cloud strategy

# Q&A