



# Event-Driven Systems on Azure

---

Done right

**Robin Konrad**

Enterprise & Solution Architect

# Different architectural styles

MONO

## Monolith

1. Single-tiered Application
2. UI, Logic, DataAccess combined
3. Deployed in one block

MICRO

## Microservices

1. Decoupled Services
2. Communication
3. Fallacies of Distributed Computing

# Event-Driven Architectures

In terms of a flavor of microservices

Event-driven architecture (EDA) is a software architecture paradigm promoting the production, detection, consumption of and reaction to events.

01

Uses events to trigger and communicate between **decoupled services**.

02

Consists of **Producers**, **Routers** and **Consumers**

03

Producer and Consumer Services are loosely coupled, can be scaled, updated and deployed independently!

**Advantage+**  
**Disadvantage:**  
**Scalable,**  
**Resilience,**  
**Flexible, but increased complexity,**  
**event ordering,**  
**lack of transactionality,**  
**monitoring.**

# Pattern - Different usings of Event (by Martin Fowler)...

## Notification



Decouple different systems by notify about state changes using an Event.

## Carried State Transfer



Upstream systems produces events for each change, Downstream systems store events they are interested in.

## Sourcing



Using fine granular events to capture any change to the state of an application as an event object.

## CQRS



Command Query Responsibility Segregation

# Different utilization of Event...

Implementation

Using fine granular events  
to capture **any change** to  
the **state** of an application  
as an **event object**.

Communication

Using **domain events** to  
communicate between  
decoupled systems.

# EDA What it is?

Communication Strategy

Implementation Strategy

No silver bullet!



“A good developer is like a werewolf: Afraid of silver bullets.”

Jochen Mader

# A lot of -ilities

Flexibility	Degradability	Customizability	Precision	Simplicity
Recoverability	Scalability	Modifiability	Predictability	Understandability
Auditability	Effectiveness	Fault-Tolerance	Testability	Traceability
Resilience	Durability	Reproducibility	Responsiveness	Stability

# Qualities - When to use Event-Driven Architectures

## Governance



Qualities that are supporting Governance & Compliance topics like **Auditability** & **Traceability** supporting the choice to use EDAs

## Maintainability



**Scalability**, **Recoverability** & **Resilience** are 1st class citizen of EDAs!

## Consistency



EDAs are eventual consistent!

## Be careful



EDAs are complex, you need matured teams to conquer the challenges!

# EDA only if...

-ilities make it affordable!

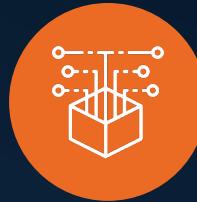
Team is able to handle complexity!

it's not used as a silver bullet!

“A good developer is like a werewolf: Afraid of silver bullets.”

Jochen Mader

# Question

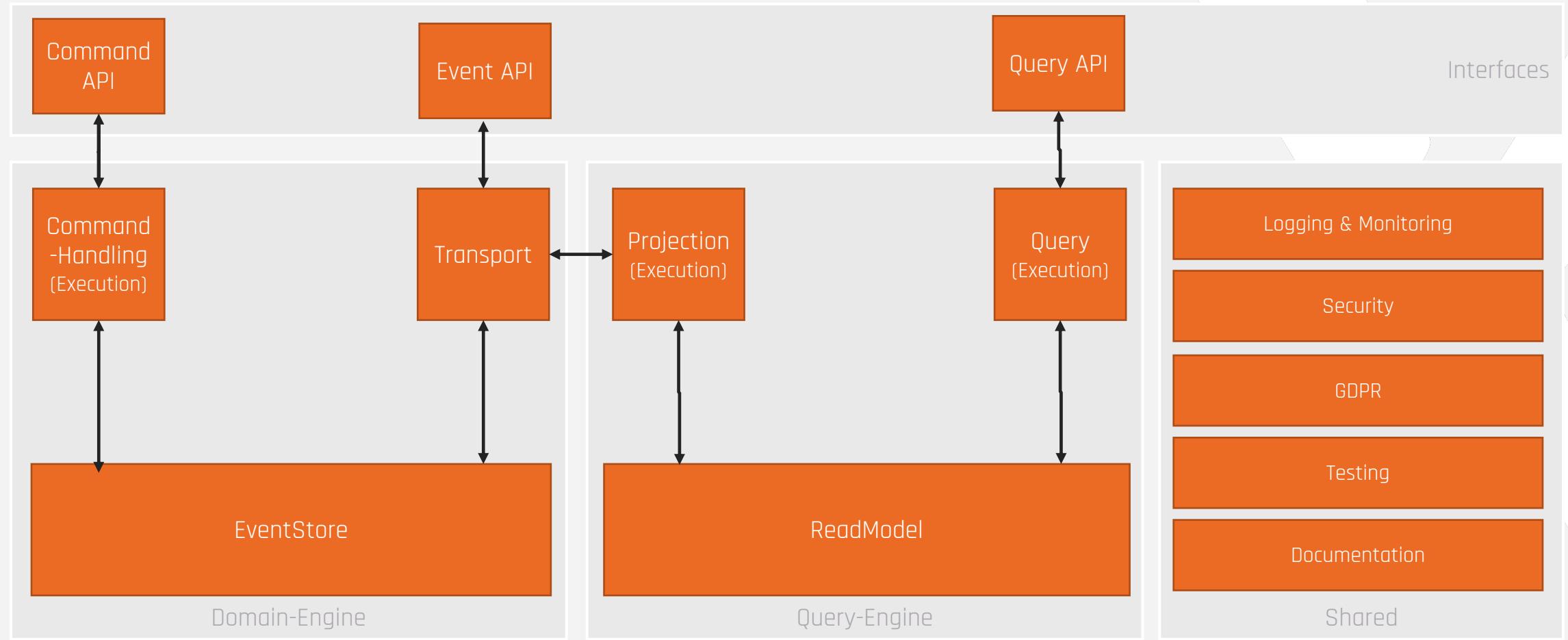


Do you already  
get in touch with  
Event-Driven  
Architecture?

---



# Components overview of an EventSourcing system



# ReadModels

- ▷ ReadModels are mostly stored in relational databases.
- ▷ Possible solution on Azure:
  - Azure SQL Database (serverless compute tier)



# Ups & Downs

- ▷ Consumption based and serverless are mostly the go-to option for cost optimization

## Down-Side

- ▷ Auto-Scale must be configured properly
- ▷ Auto-Pausing and Auto-Resume can lead to unexpected behavior on consumer-side

## Solution:

- ▷ Collect usage data and adjust scaling to it
- ▷ Avoid Auto-Pausing if it's causing a lot of trouble, but keep load as small as possible to do so



# EventStore

- ▷ EventStores can be easily implemented with object storages.
- ▷ Possible solution on Azure:
  - › Azure Cosmos DB



# Query Problem

- ▷ Querying Azure Cosmos DB can be **expensive**, if you don't care about partitioning.
- ▷ **Identity of Aggregate** is mostly a good choice
- ▷ Querying only **one partition** at a time is really **cheap!**



# Take also care about

- ▷ Change Feed Listener of Azure Cosmos DB can be used to implement the Out-Box-Pattern
- ▷ Be carefull
  - Use **replication** for resilience
  - Right **indexing strategy** is a key to good performance
  - Keep Azure Cosmos Db as **small as possible**

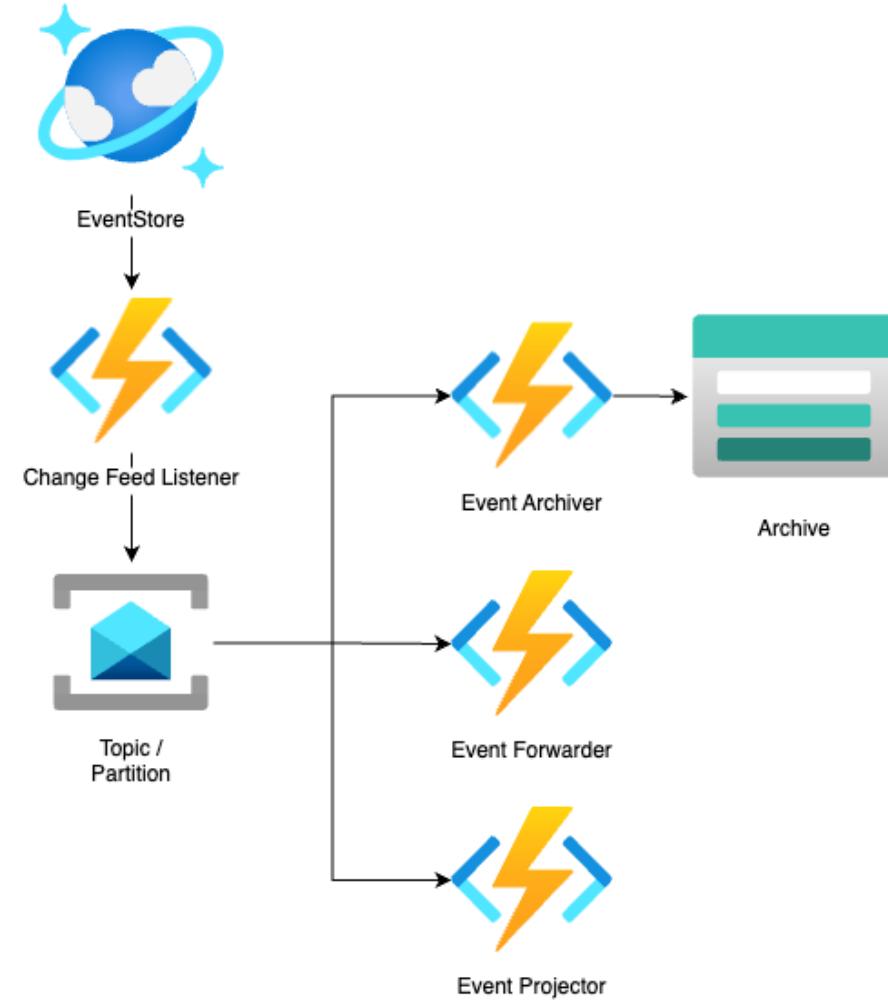


# Out-Box-Pattern made easy

- ▶ Change Feed Listener to implement Out-Box-Pattern and reduce complexity.

## Advantages

- ▶ Right settings for **scaling** avoids messing up event ordering
- ▶ Forward stored events to **an Azure Service Bus Topic** or **an Azure Event Hub Partition**
- ▶ Events emitted by the Change Feed Listener can be **archived** to keep Cosmos Db at a **valuable size**



# Pitfalls on EventStores

- ▷ Apache Kafka **does not** exist to be used as Event Store!
- ▷ Good solution for **event-streaming** @ scale
- ▷ But don't **underestimate** operations and consumptions!

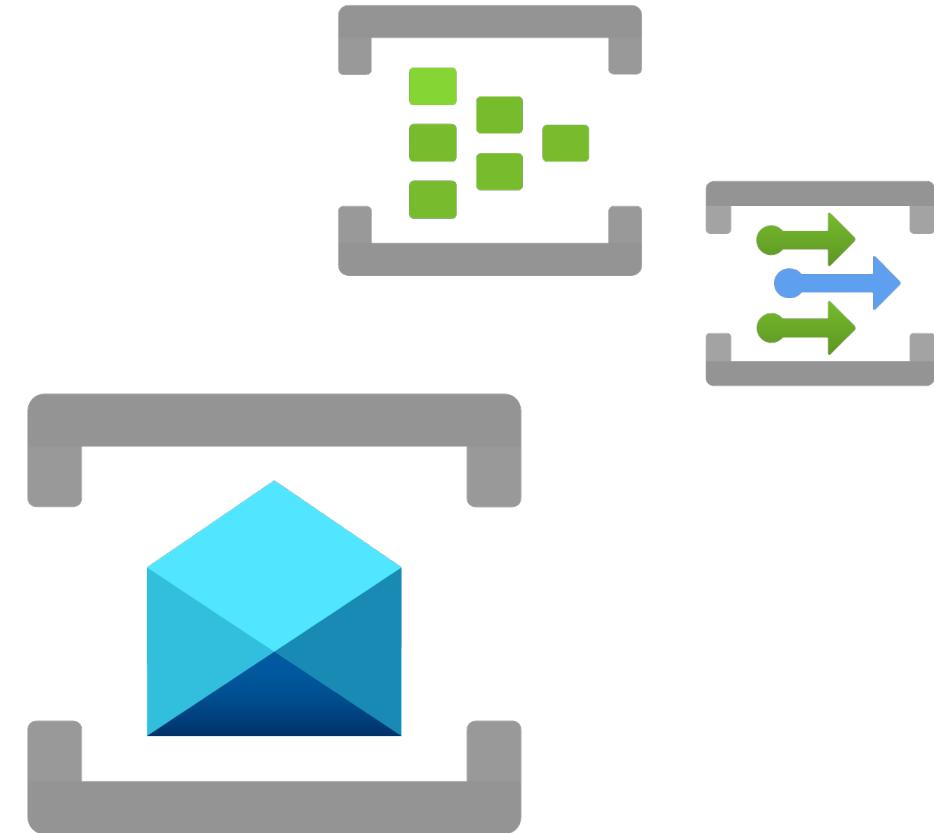


**Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.**

(Source: <https://kafka.apache.org/>)

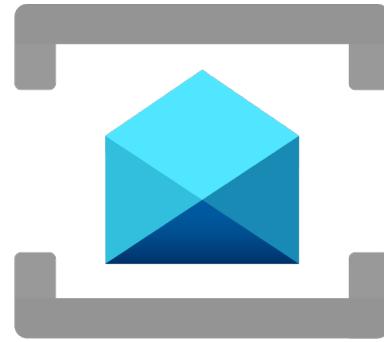
# Transport

- ▷ Transport of emitted events can get hard in terms of **message ordering** and **filtering**
- ▷ Possible solutions on Azure for transporting events:
  - > Azure Service Bus
  - > Azure Event Grid
  - > Azure Event Hub
  - > Azure Storage Queues



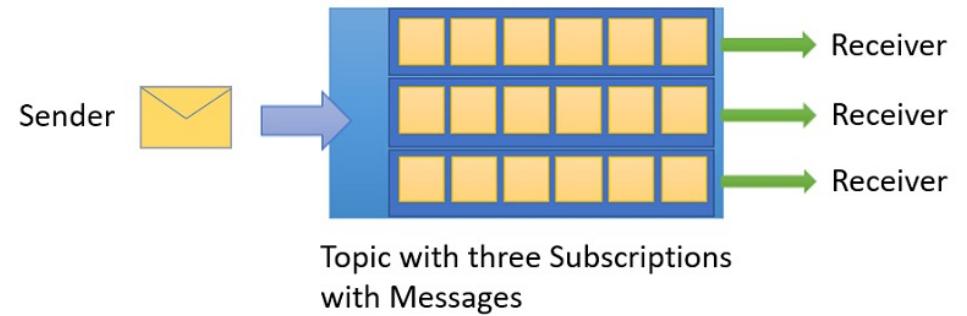
# Message ordering isn't easy

- ▶ Message Ordering isn't guaranteed in most services
- ▶ Only solution:
  - › Azure Service Bus



# Use Topics

- ▷ Use **Topics** to enable multiple subscribers to your event stream
- ▷ Use **SQL-Style filtering** to filter on subscription level



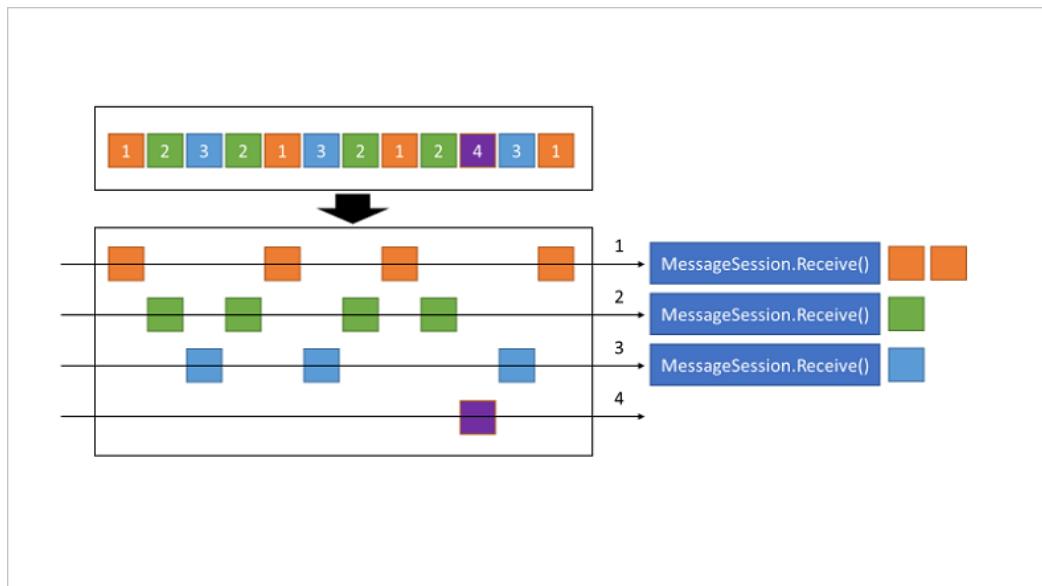
<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions>

# Use Sessions

- ▶ Use Sessions to guarantee message ordering!

- ▶ But be careful

- ▶ Choose the right **SessionId** to avoid too small or too big sessions.
- ▶ Identity of Aggregate is mostly a good choice.



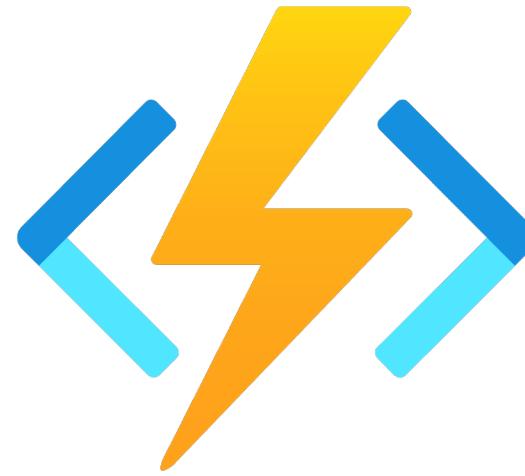
<https://learn.microsoft.com/en-us/azure/service-bus-messaging/message-sessions>

# Execution

- ▶ Execution of business logic can be easily done one Azure!

- ▶ Solution of choice

- › Azure Functions



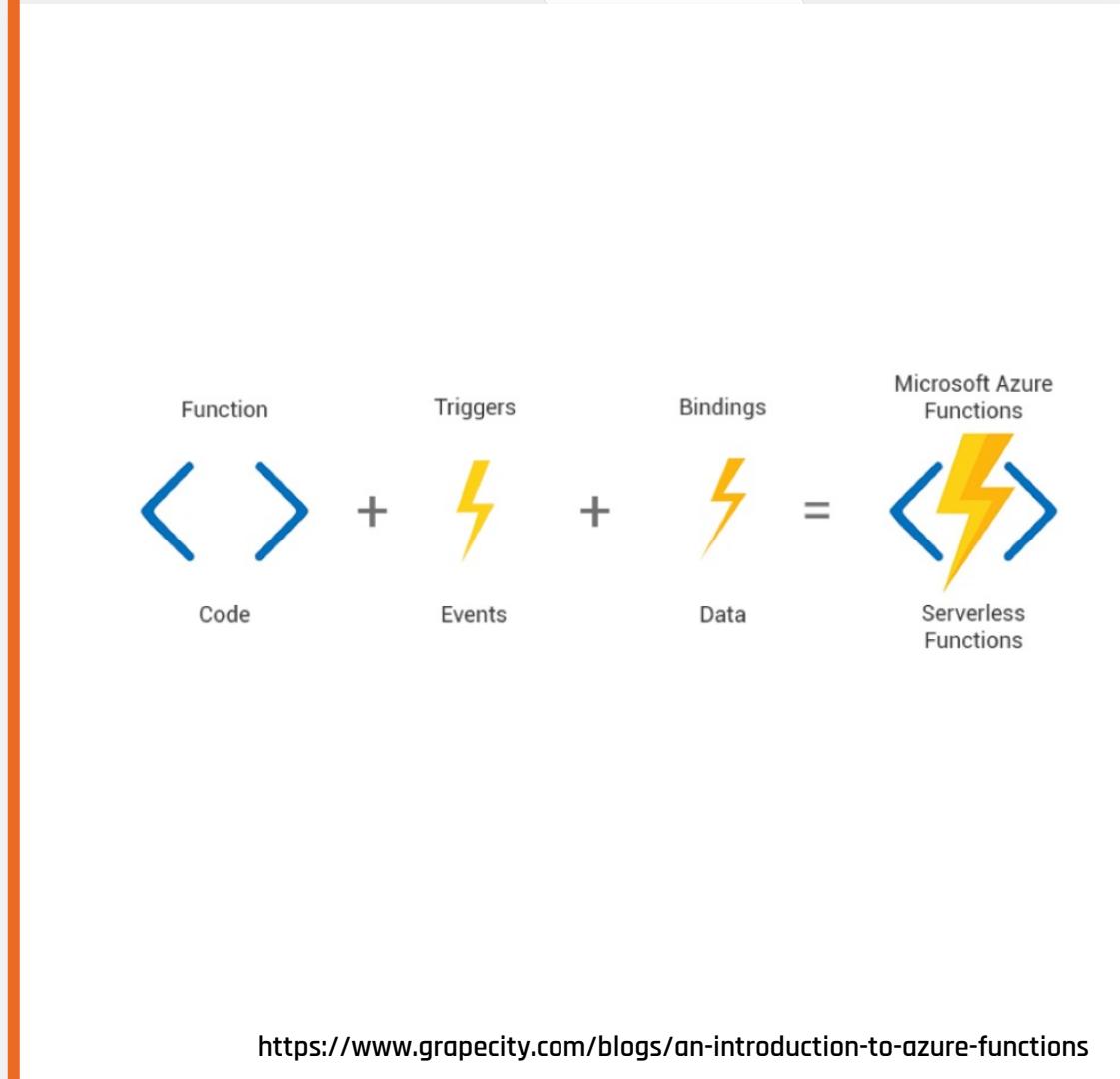
# Falling a sleep or not

- ▶ Azure Functions are not **pre-warmed** if you're not using Premium Tier.
- ▶ Using Time-Trigger to keep them awake

```
● ● ●  
1 [FunctionName("StillAlive")]  
2 public void StillAlive(  
3     [ILogger] ILogger logger,  
4     [TimerTrigger("30 */4 * * *", RunOnStartup = false)] TimerInfo timer  
5 )  
6 {  
7     if (timer.IsPastDue)  
8     {  
9         logger.LogInformation("StillAlive is running late!");  
10    }  
11    logger.LogInformation($"StillAlive triggered at: {DateTime.Now}");  
12 }
```

# Build in Trigger & Bindings

- ▶ Azure Functions provide a wide set of default **Trigger & Bindings**.
- ▶ Default Trigger & Bindings are **not optimized for performance**.
- ▶ Write **custom Trigger & Bindings** if you need to handle @ scale.



# Scaling execution

- ▷ Azure Functions doing a great job **on scaling!**
- 
- ▷ Analyze frequently using Application Insights to gather the right settings
- 
- ▷ AI can help you to auto detect **common pattern** for **peaks** and adjust scale-settings.

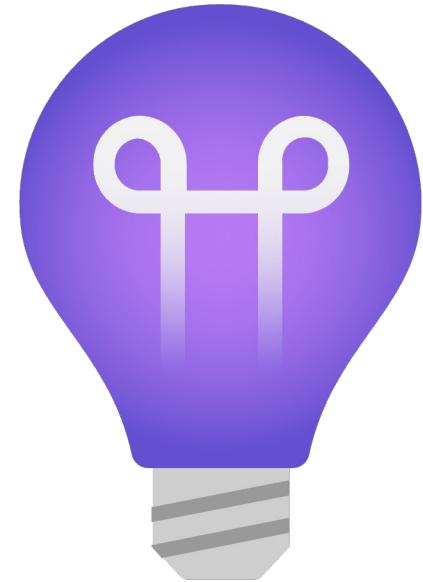


# Logging

- ▷ Getting a full overview of the system state and it's containing operations is essential.

- ▷ Solution of choice

- ▷ Application Insights



# Expenses

- ▶ Application Insights can really let explode your costs!
- ▶ Be careful what you log, in best case use **dynamic distributed settings** about log-level.
- ▶ Choose wisely on **Retention Period**



# Interfaces

- ▷ All requests to reach any API of your solution should have one manageable entry point!
- ▷ Solution of choice
  - Azure API Management Services
- ▷ Various advantages
  - Analyze usage
  - Providing different sets of functionalities to different consumers
  - Securing your solution



# Documentation



Distribution of various information is a key success factor!



Event-Definition, How-To Consume / Subscribe, Domain knowledge, Expectations



Possible solution

- › Using easy to access solutions
- › OpenAPI Definition
- › AsyncAPI
- › EventCatalog.dev

The screenshot displays the EventCatalog.dev interface, which includes:

- EventCatalog**: A dashboard showing a list of events with details like version, status, and descriptions.
- Events (9)**: A search bar and filter options for events across domains (Shopping, Orders) and services.
- Swagger Editor**: An editor for the Pet Store API, showing the OpenAPI 3.0 specification and code snippets for GET, PUT, and DELETE operations on the /user endpoint.
- Studio**: A dark-themed developer interface for managing Kafka topics, brokers, and schema definitions. It shows a tree view of servers and topics like "KAFKA-SECURE test" and "smartlylighting.streetlights".
- Streetlights Kafka API 1.0.0**: A detailed API documentation page for the Streetlights Kafka API, including its features and a "Check out its awesome features:" section.
- Servers**: A configuration section for a Kafka cluster, including fields for security protocols (SASL\_PLAINTEXT, SCRAM-SHA-256) and authentication methods (SCRAM-SHA-256).
- Operations**: A section for managing Kafka brokers and topics.

# Testing & Debugging

- ▷ Event-Driven Architectures are hard to debug and test! Use **abstraction** wherever possible!
- ▷ Satisfied by:
  - Use a **correlation Id** in every call you do!
  - **Abstract** as much as meaningful within your code
  - Heavily use **IaC** to deploy independent test environments for each run!
  - Go **BDD -> SpecFlow as solution!** Early, execute frequently.
  - Do **CDCT**, every single time!



# Security

► Securing distributed systems can be hard!

► Security is always a First Class Citizen!

- › Use Service Principals and managed identities **every time** possible!
- › Use **Azure KeyVault** to store secrets!
- › Secure **every call** within the module / service / component!



# GDPR

- ▷ Handling of **GDPR** relevant information can be hard in EDAs, specially if storing events.

- ▷ Possible solution

- Only distribute events to notify about a state change
- Distribute hydrated events & encrypt sensitive fields



# Wrap up!

Choose services wisely!

Use the right tool for the job!

Keep an eye on consumption!

“A good developer is like a werewolf: Afraid of silver bullets.”

Jochen Mader

# Thank you



Blog : <https://robinkonrad.de>



Twitter : @robin\_Konrad\_



GitHub : @robinkonrad



LinkedIn : <https://www.linkedin.com/in/robin-konrad/>

# Let's connect

---



<https://robinkonrad.de>



@robin\_konrad\_



@robinkonrad



<https://www.linkedin.com/in/robin-konrad>

Let's connect!



## Robin Konrad

Enterprise Architect  
Solution Architect

<https://www.linkedin.com/in/robin-konrad>  
rkonrad@xpirit.com