Xpirit

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 loosly coupled, can be scaled, updated and deployed independently!

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

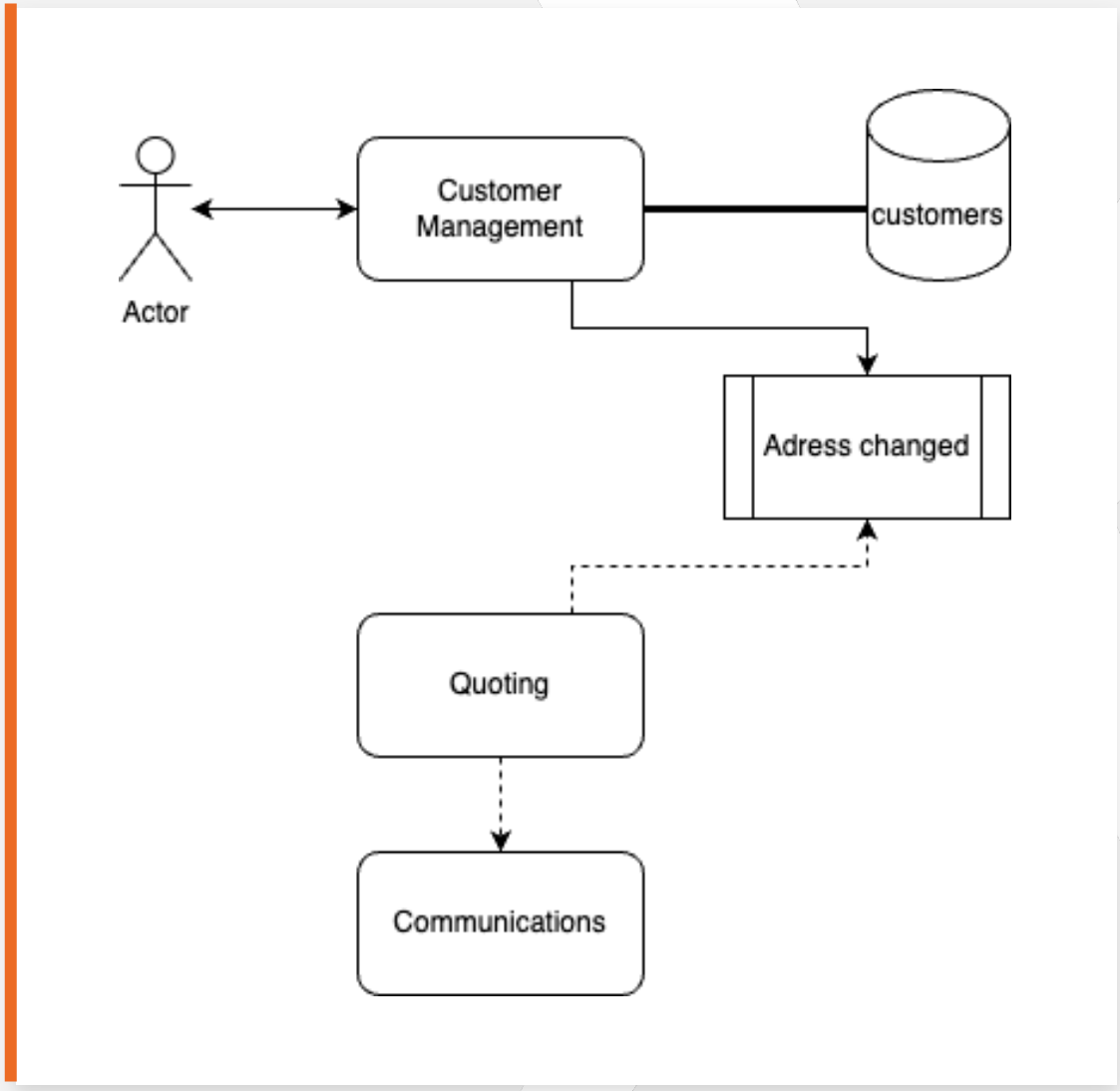
Pattern – Different usings of Event (by Martin Fowler)...

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

Event Notifications

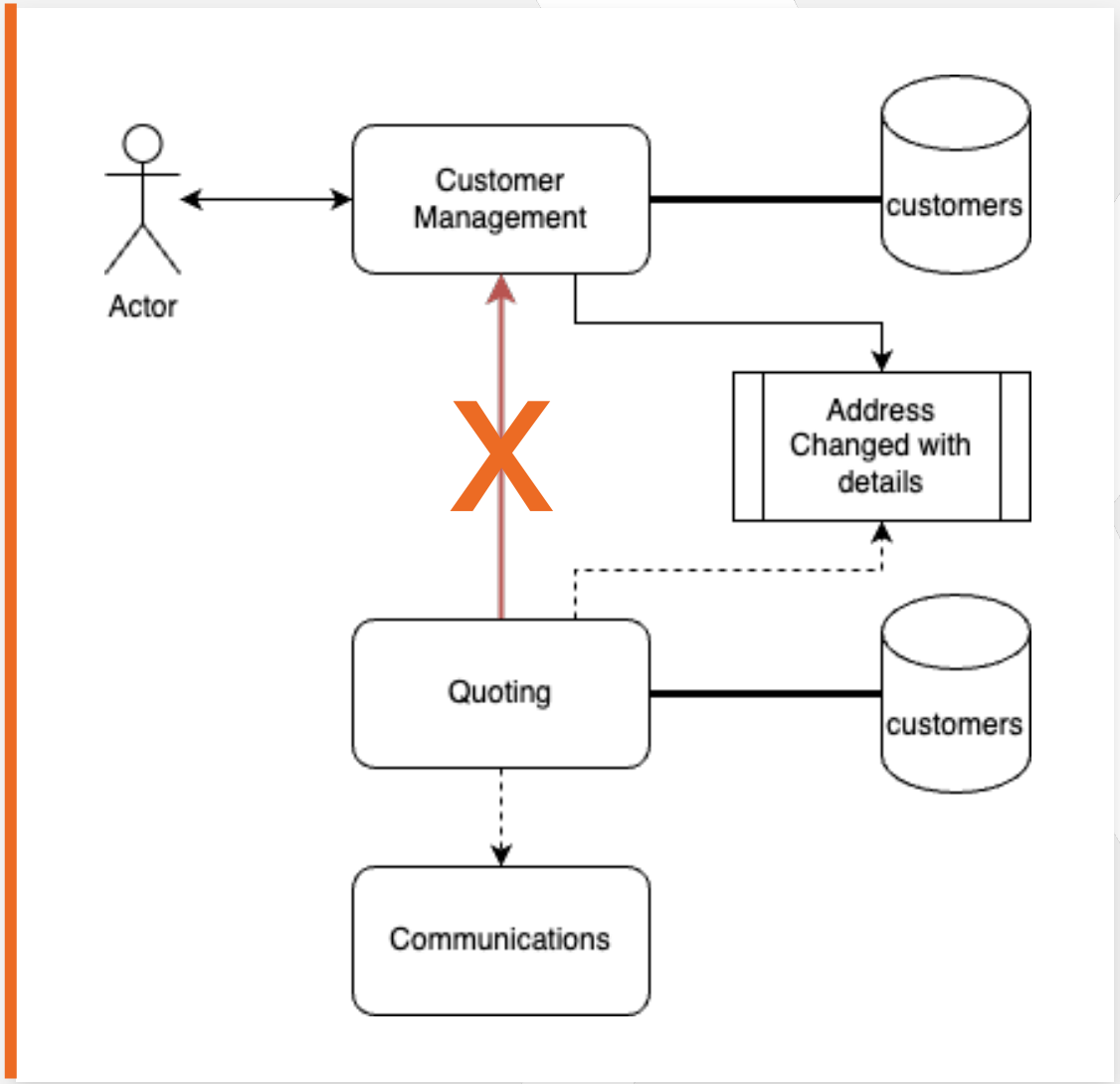
- Decouple different systems by notify about state changes using an Event.

- Pros & Cons
  - Decouple receiver from sender
  - No statement of overall behavior



Event Carried State Transfer

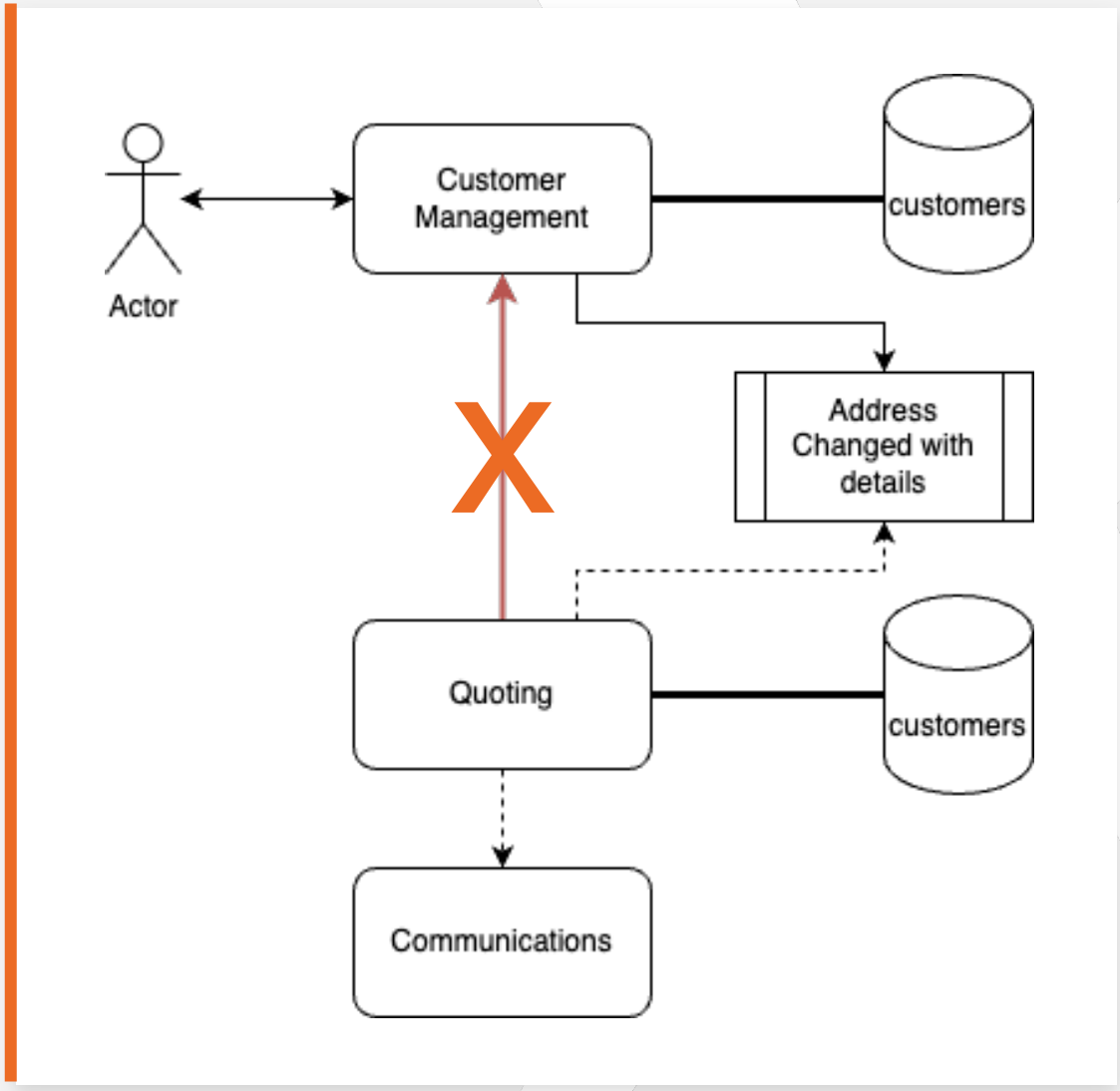
Upstream systems produces events for each change, Downstream systems store events they are interested in.



Event Carried State Transfer

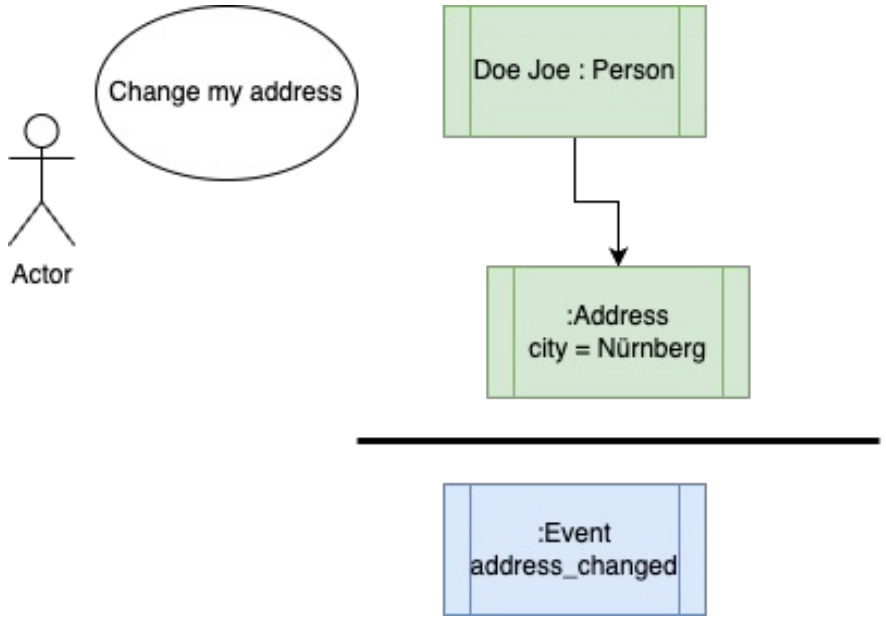
- Mostly used when reliability is increased, but it has pay offs.

- Pros & Cons
  - Decouple receiver from sender
  - Reduced traffic
  - Replicated Data -> Eventual Consistency



Event Sourcing

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

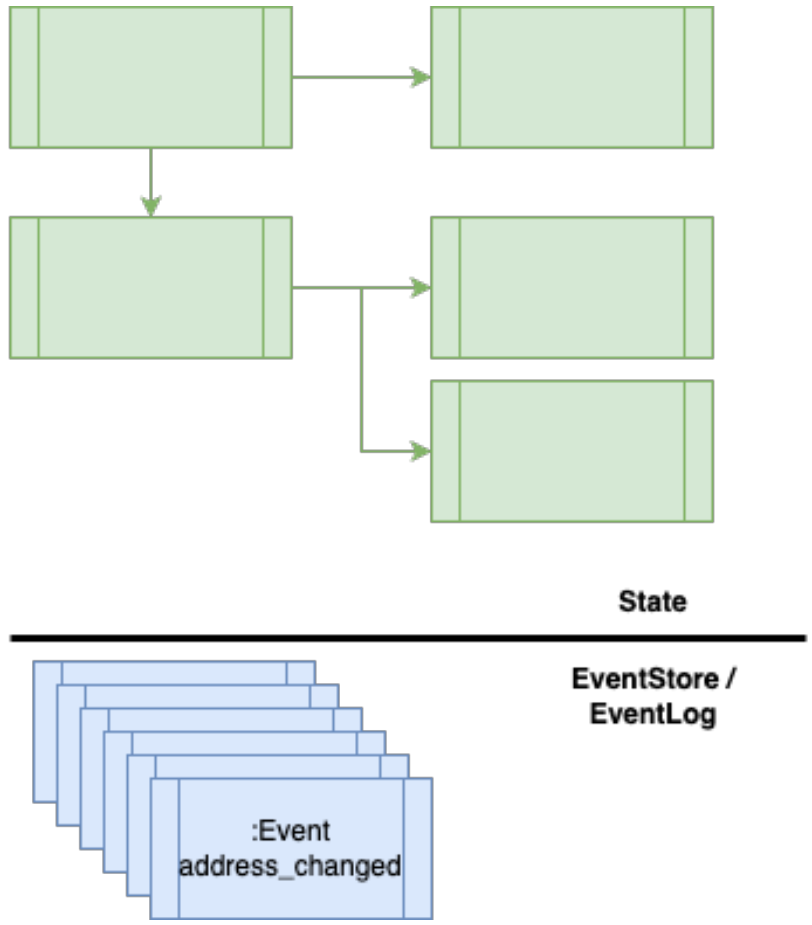


Event Sourcing

State of the application can be restored at any time to any state using the events from the EventStore / EventLog.

Pros & Cons

- Audit safe & Historic State available
- Unfamiliar to most developers
- Versioning & Async topics make things hard

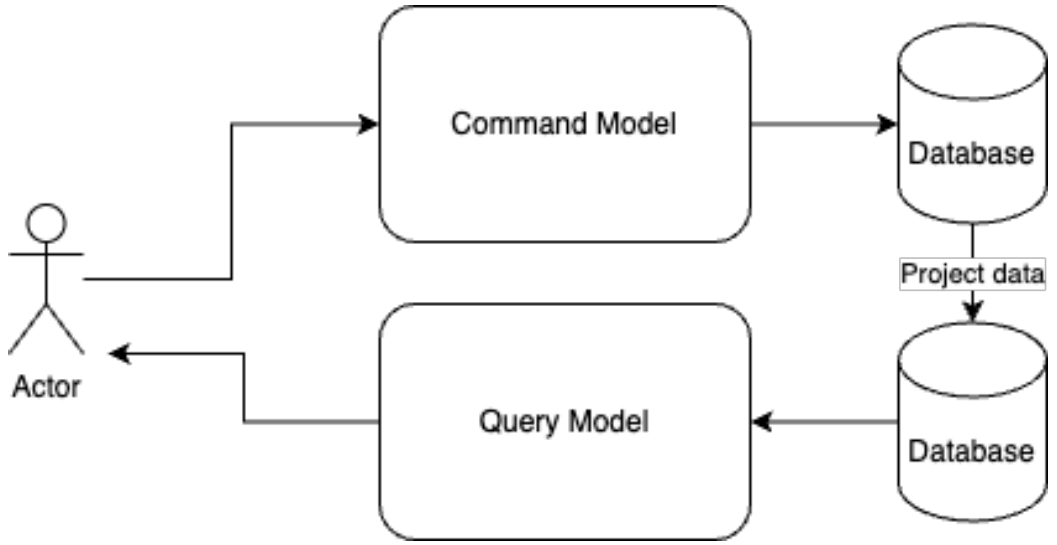


~~Event~~ CQRS

Command Query Responsibility Segregation

Pros & Cons

- CQRS isn't only an implementation strategy
- Complexity is again increasing
- No new pattern e.g. Reporting Database are years in the field!
- Models on Execution and Read are completely different!



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?

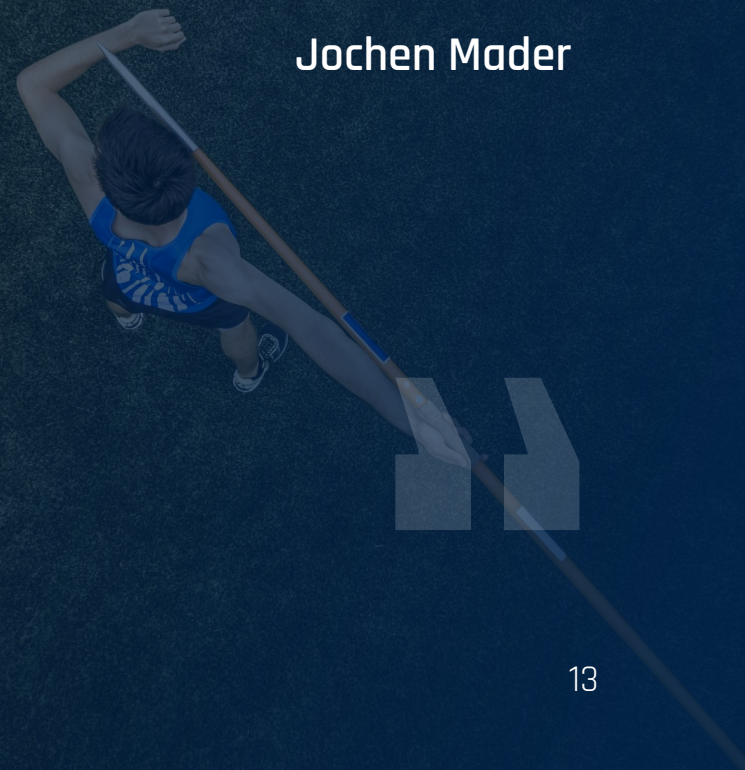
Communication Strategy

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	Understand-ability
Auditability	Effectiveness	Fault-Tolerance	Testability	Traceability
Resilience	Durability	Reproducibility	Responsiveness	Stability

Qualities – When to use Event-Driven Architectures

Governance	Qualities that are supporting Governance 6 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!

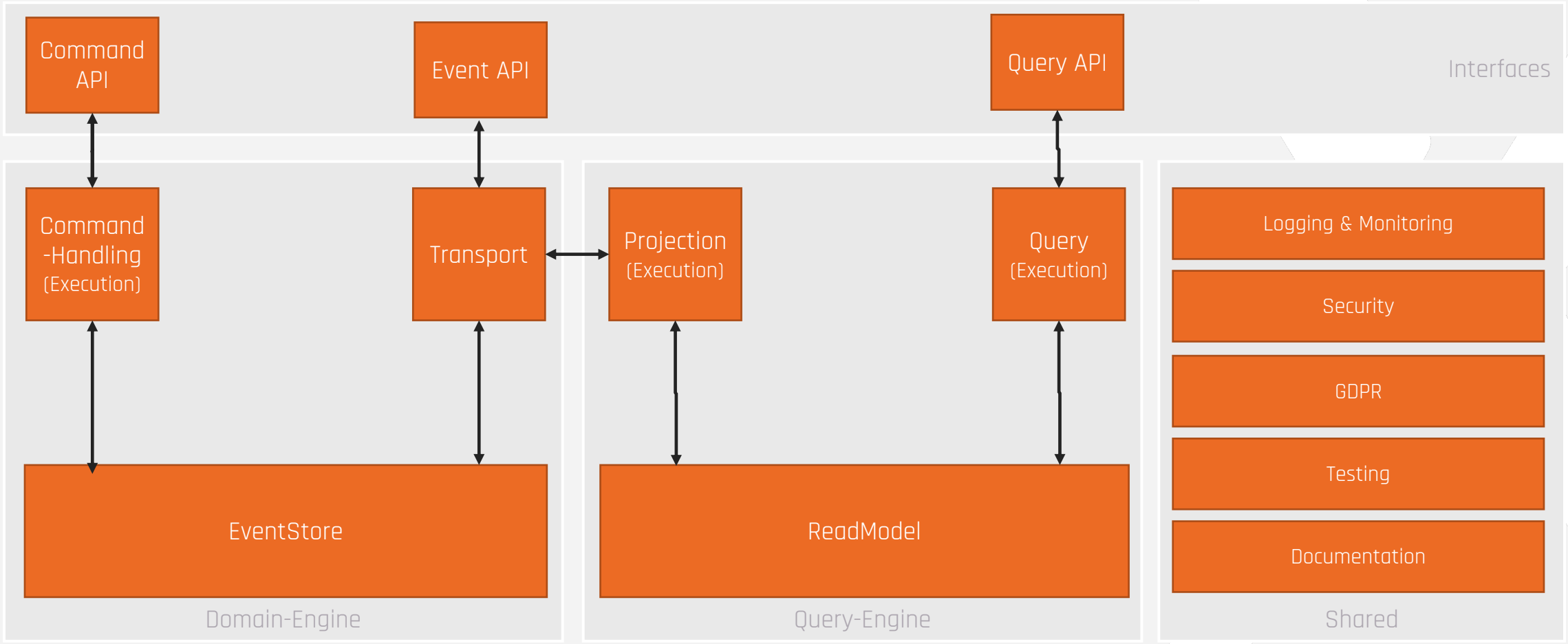
it's not used as a silver bullet!

"There is no silver bullet. There are alwaxs options and the options have consequences."

Ben Horowitz

Xpirit

Components overview of an EventSourcing system



ReadModels

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

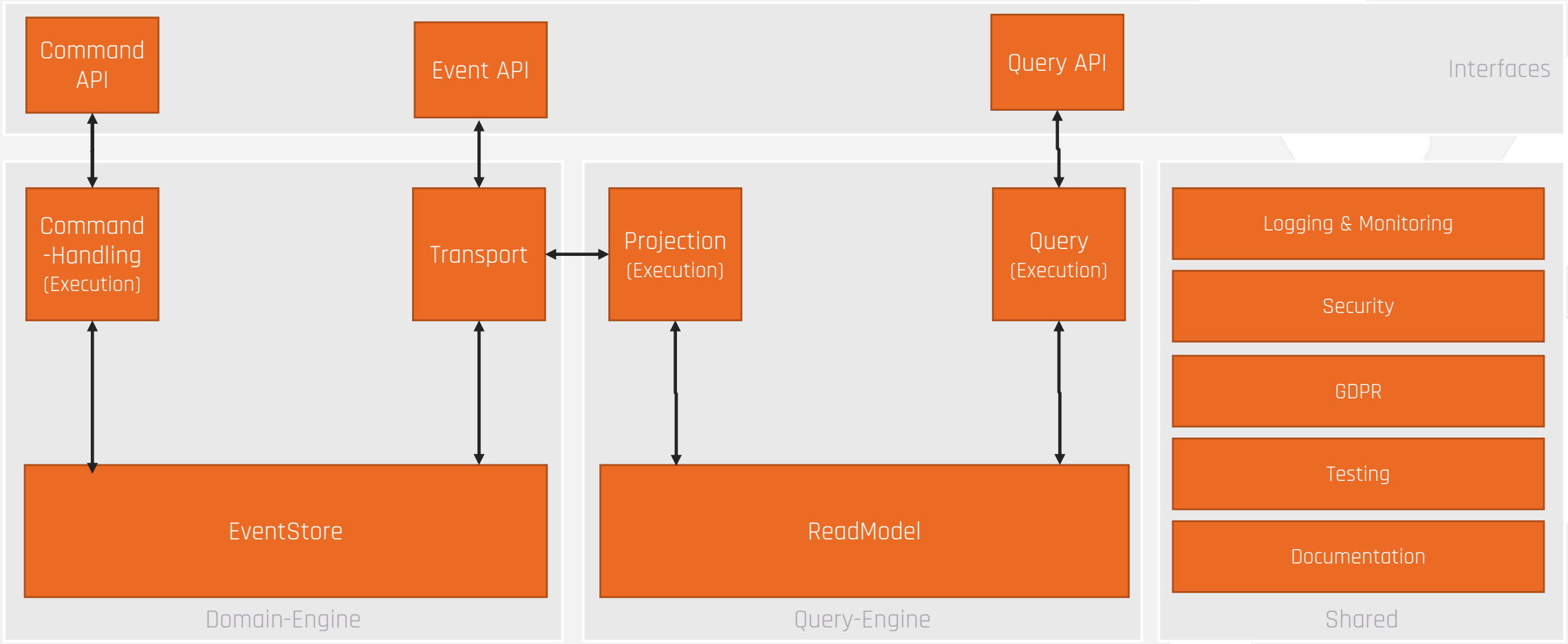
Ups & Downs

- 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

Components overview of an EventSourcing system



EventStore

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

Query Problem

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

Take also care about

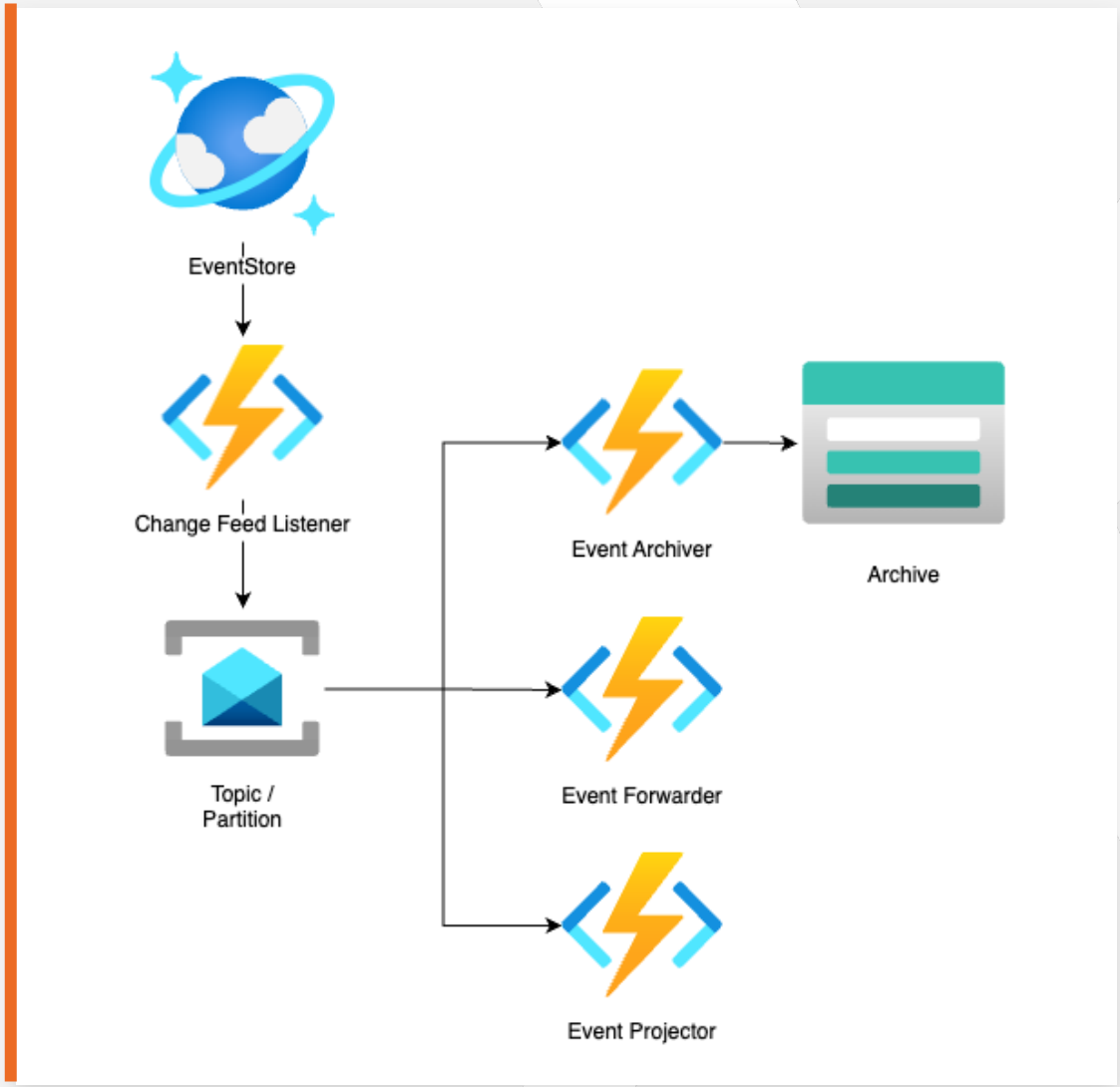
- Change Feed Listener of Azure Cosmos DB can be used to implement the Out-Box-Pattern
- Be carefull
  - Use replication for resilience
  - Right indexing strategy is a key to good performance
  - Keep Azure Cosmos Db as small as possible

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



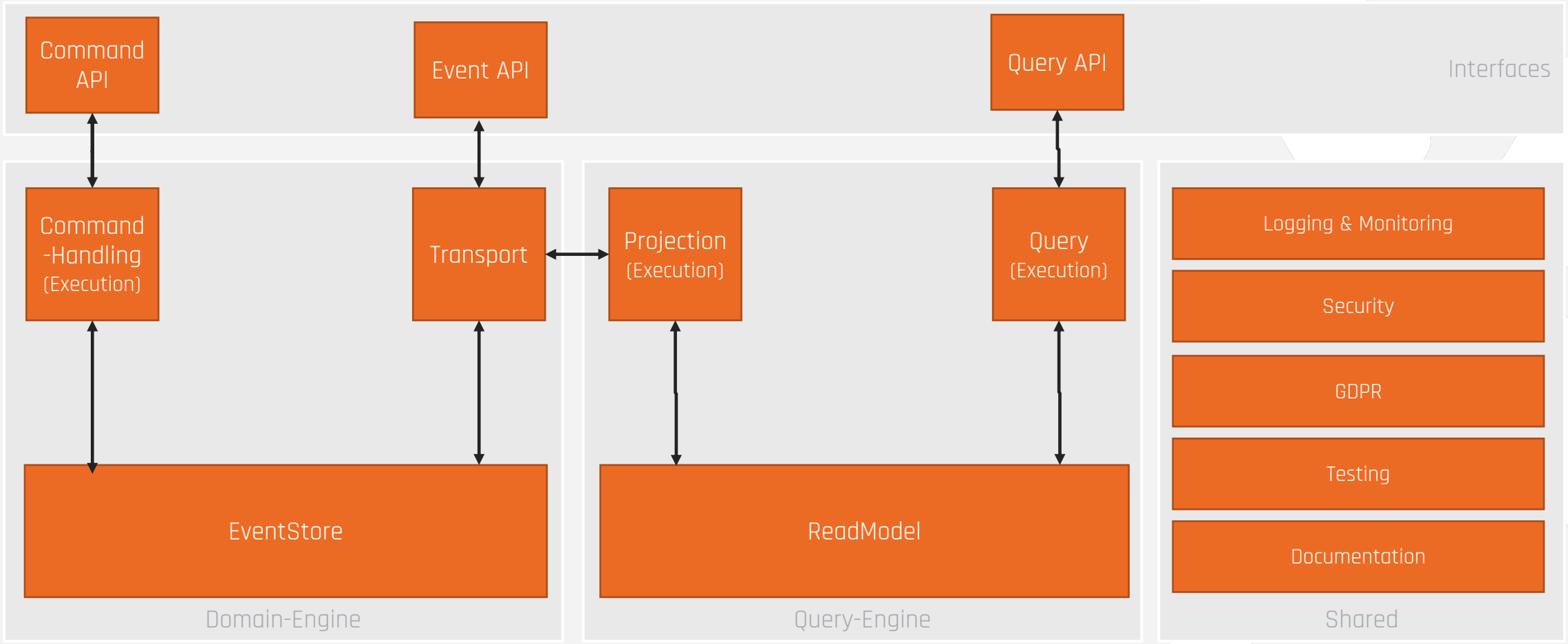
Pitfalls on EventStores

- 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.

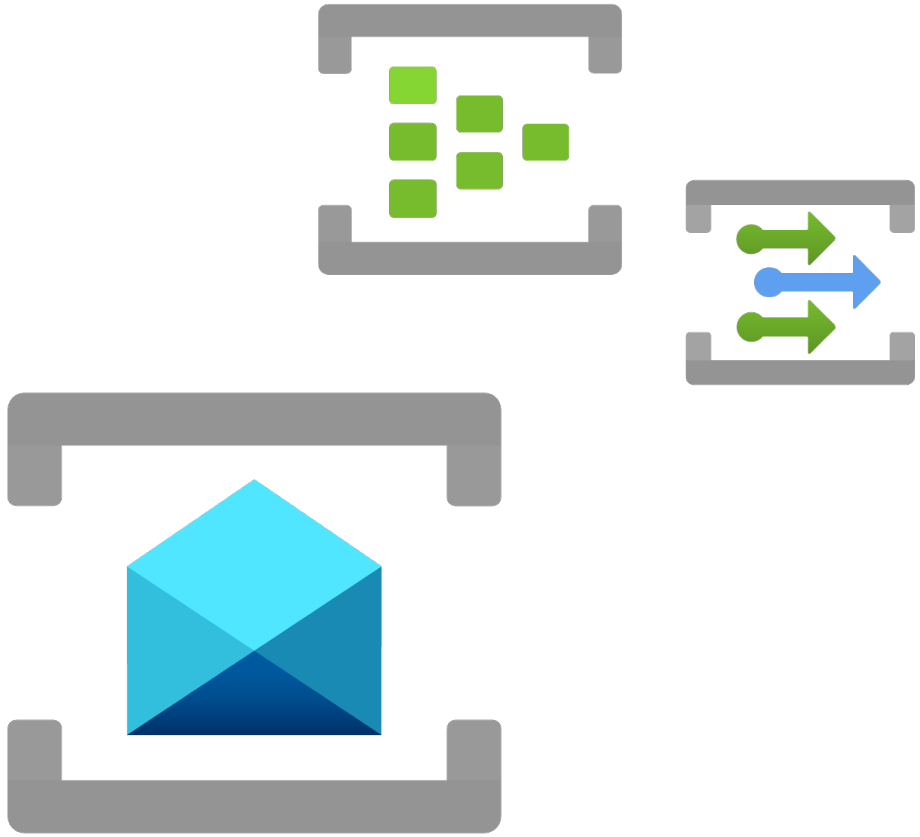
(Source: https://kafka.apache.org/)

Components overview of an EventSourcing system



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

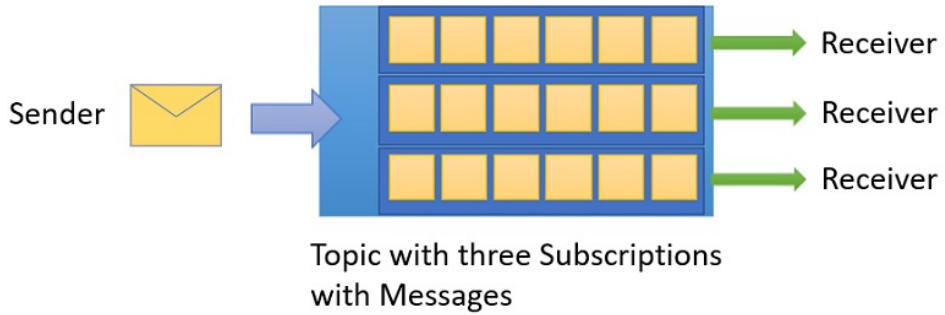


Message ordering isn't easy

- 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

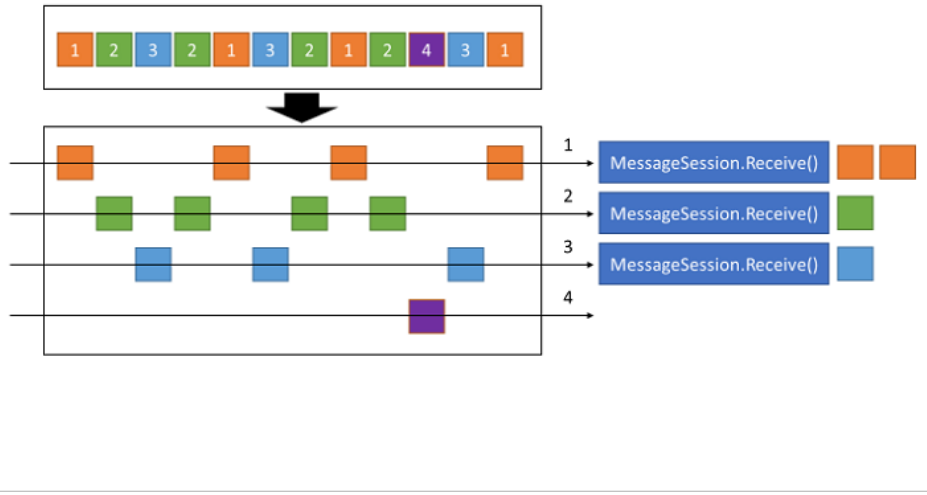


https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions

Use Sessions

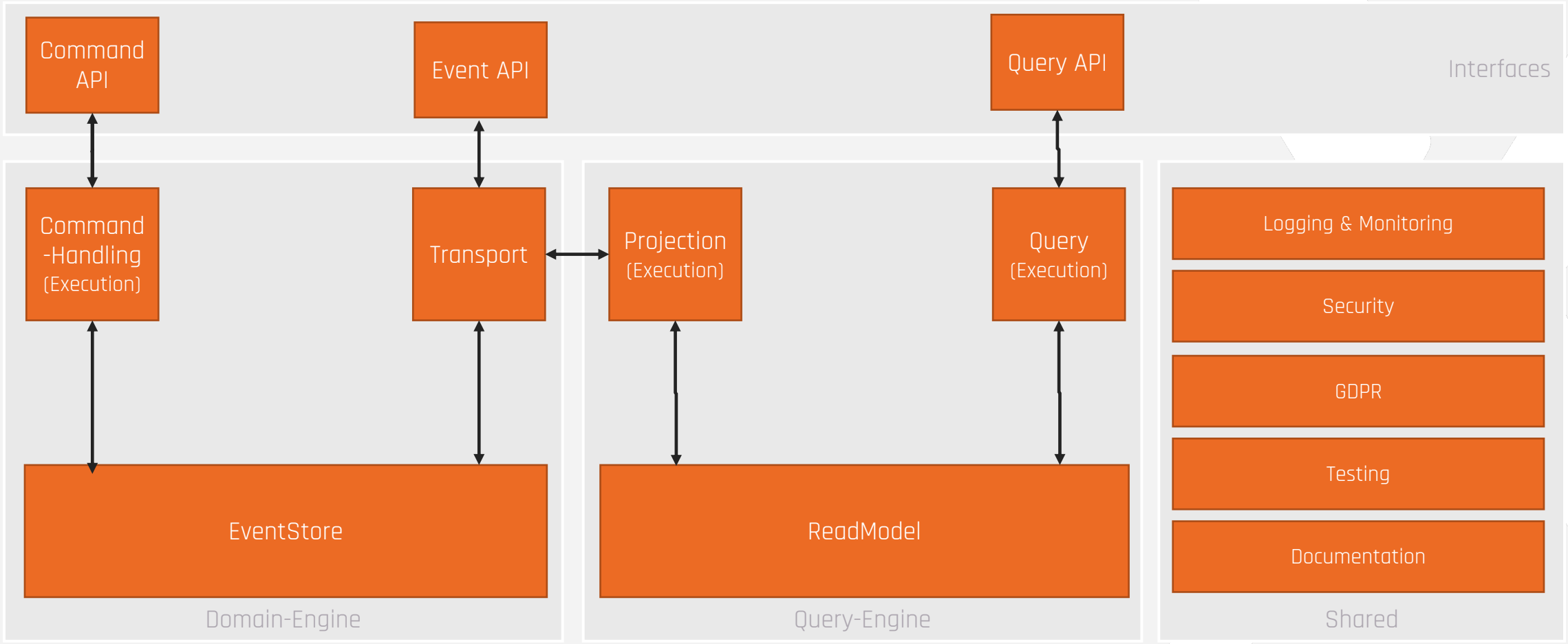
- Use Sessions to guarantee message ordering!

- 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

Components overview of an EventSourcing system



Execution

- Execution of business logic can be easily done one Azure!
- Solution of choice
  - Azure Functionsç

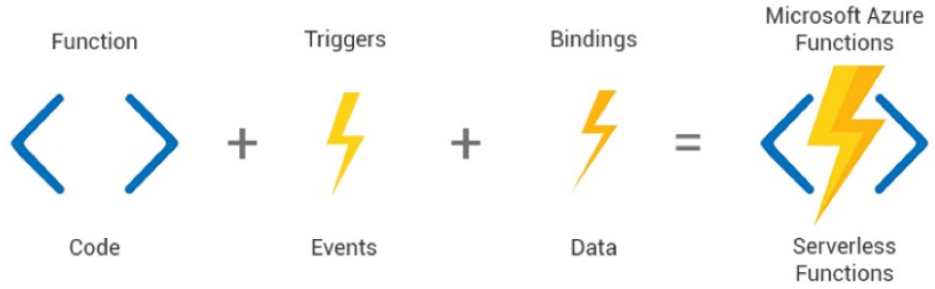
Falling a sleep or not

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

```
[FunctionName("StillAlive")]
public void StillAlive(
    [ILogger] ILogger logger,
    [TimerTrigger("30 */4 * * * *", RunOnStartup = false)] TimerInfo timer
)
{
    if (timer.IsPastDue)
    {
        logger.LogInformation("StillAlive is running late!");
    }
    logger.LogInformation($"StillAlive triggered at: {DateTime.Now}");
}
```

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.

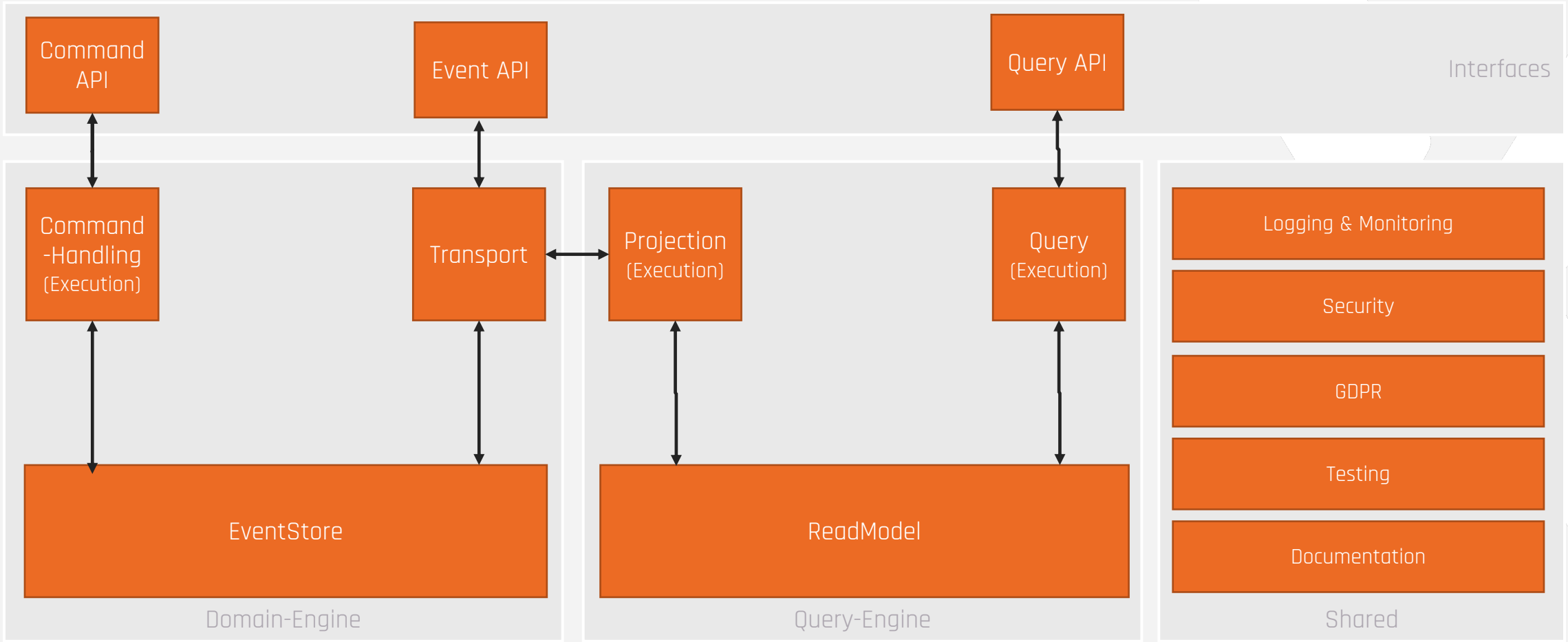


https://www.grapecity.com/blogs/an-introduction-to-azure-functions

Scaling execution

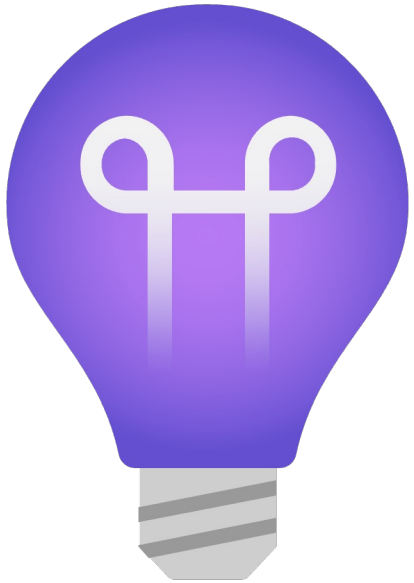
- Azure Functions doing a great job on scaling!
- Analyze frequently using Application Insights to gather the right settings
- AI can help you to auto detect common pattern for peaks and adjust scale-settings.

Components overview of an EventSourcing system



Logging

- Getting a full overview of the system state and it's containing operations is essential.
- Solution of choice
  - Application Insights



Expenses

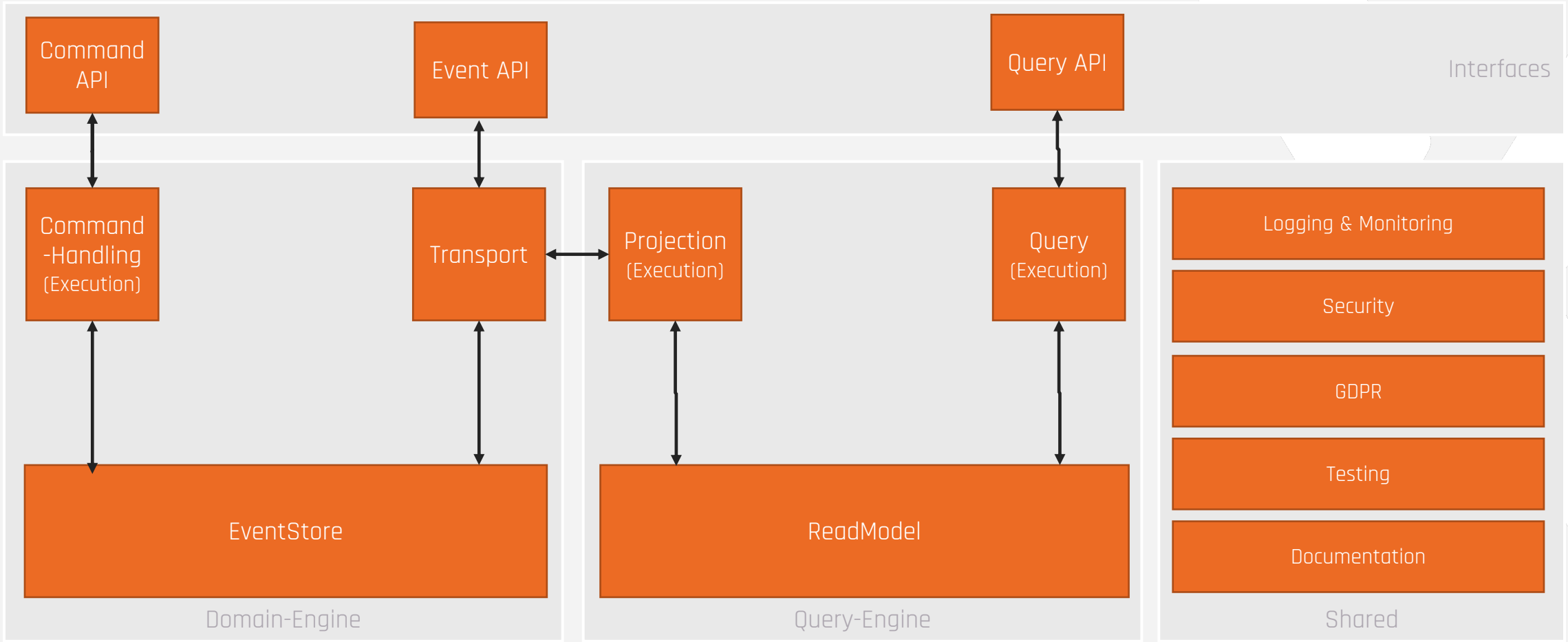
- Application Insights can really let explode your costs!
- Be careful what you log, in best case use dynamic distributed settings about log-level.
- Choose wisely on Retention Period

Dynamically change log-level

- net6 introduces IOptions<>, IOptionsMonitor<> or IOptionsSnapshot
- IOptionsMonitor<> allows you to update settings during runtime!

```
using Microsoft.Extensions.Options;

public class MyService: IDisposable
{
    private readonly IOptionsMonitor<MySettings> _options;
    private readonly IDisposable _optionsChangeToken;

    public MyService(IOptionsMonitor<MySettings> options)
    {
        _options = options;
        _optionsChangeToken = _options.OnChange((newOptions) => {
            // Perform actions based on the new options
        });
    }

    public string GetOptionValue()
    {
        return _options.CurrentValue.MyOptionValue;
    }

    public void Dispose()
    {
        _optionsChangeToken.Dispose();
    }
}

```

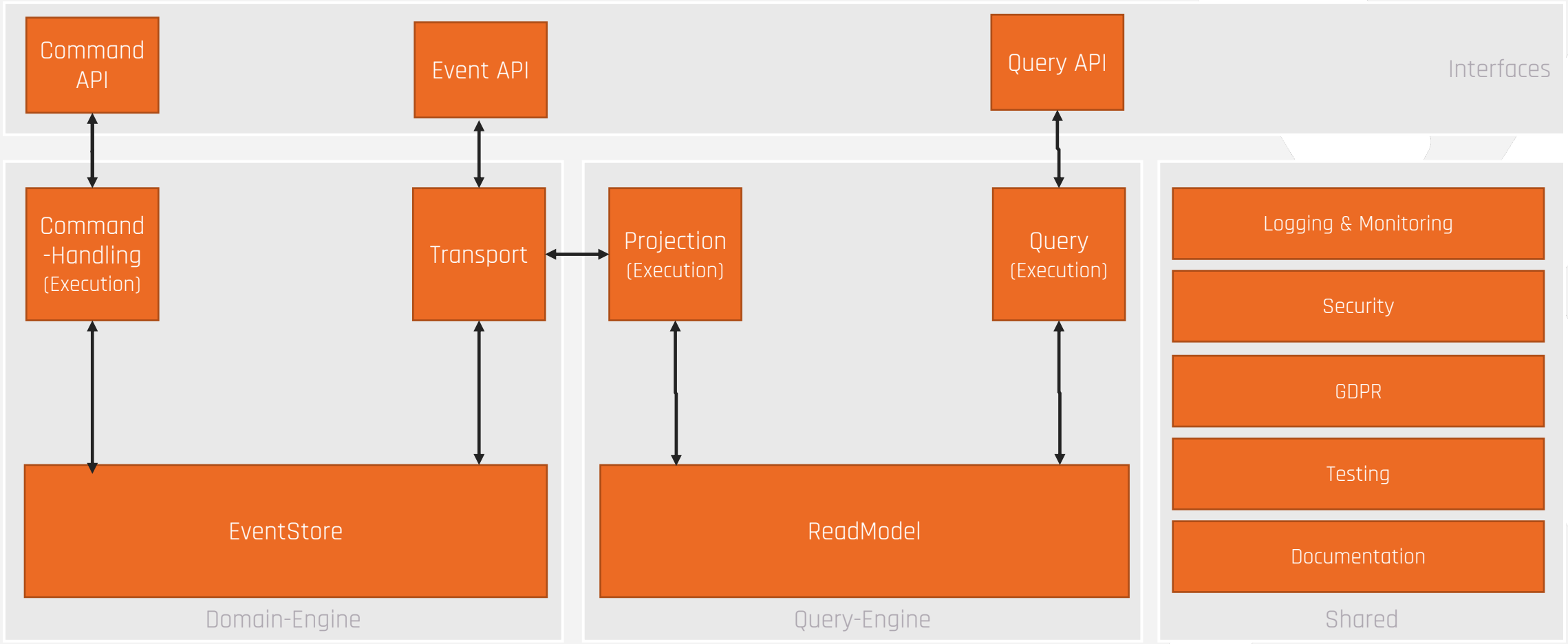
Components overview of an EventSourcing system



Interfaces

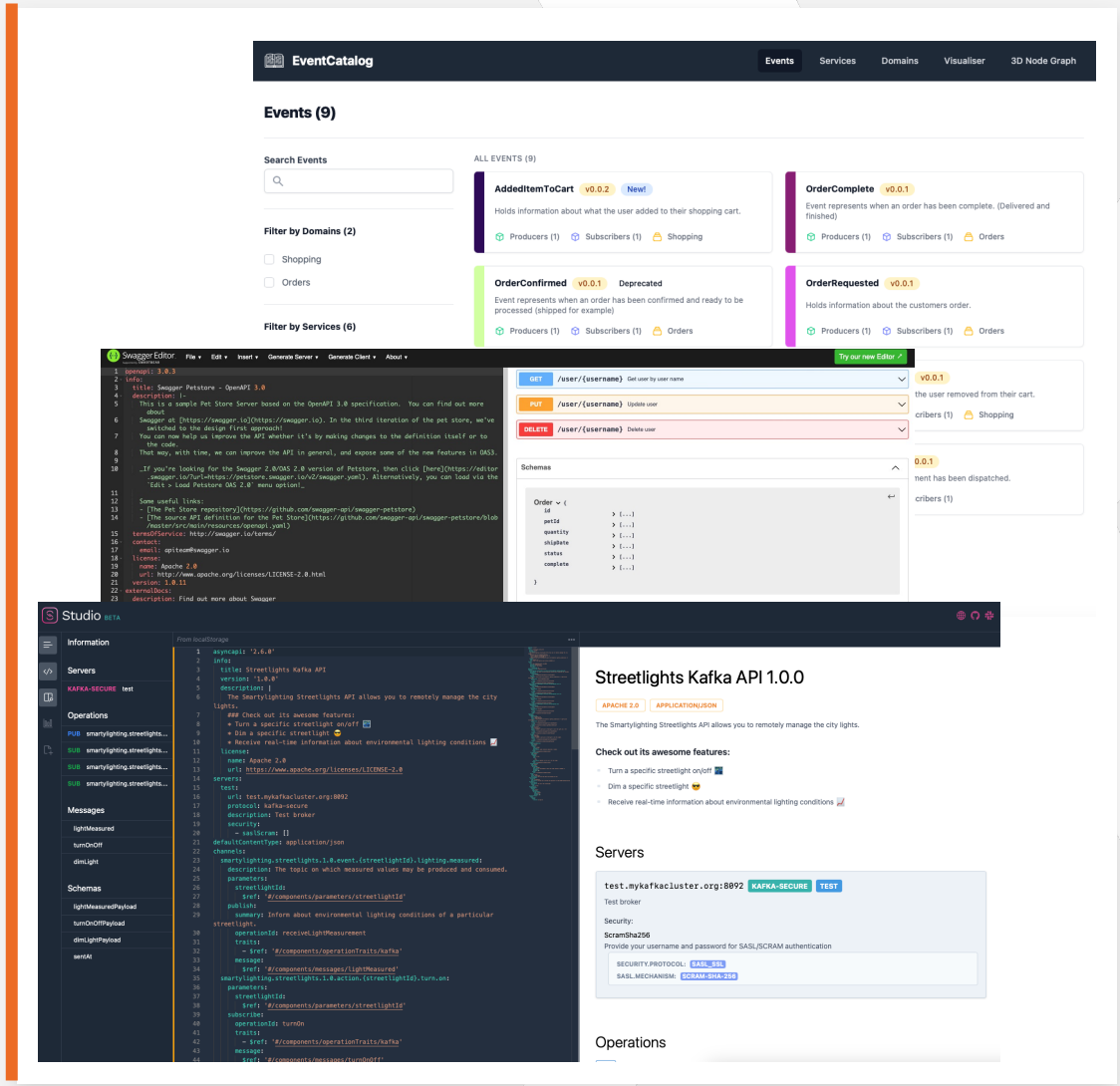
- All requests to reach any API of your solution should have one manageable entry point!
- Solution of choice
  - Azure API Management Services
- Various advantages
  - Analyze usage
  - Providing different sets of functionalities to different consumers
  - Securing your solution

Components overview of an EventSourcing system

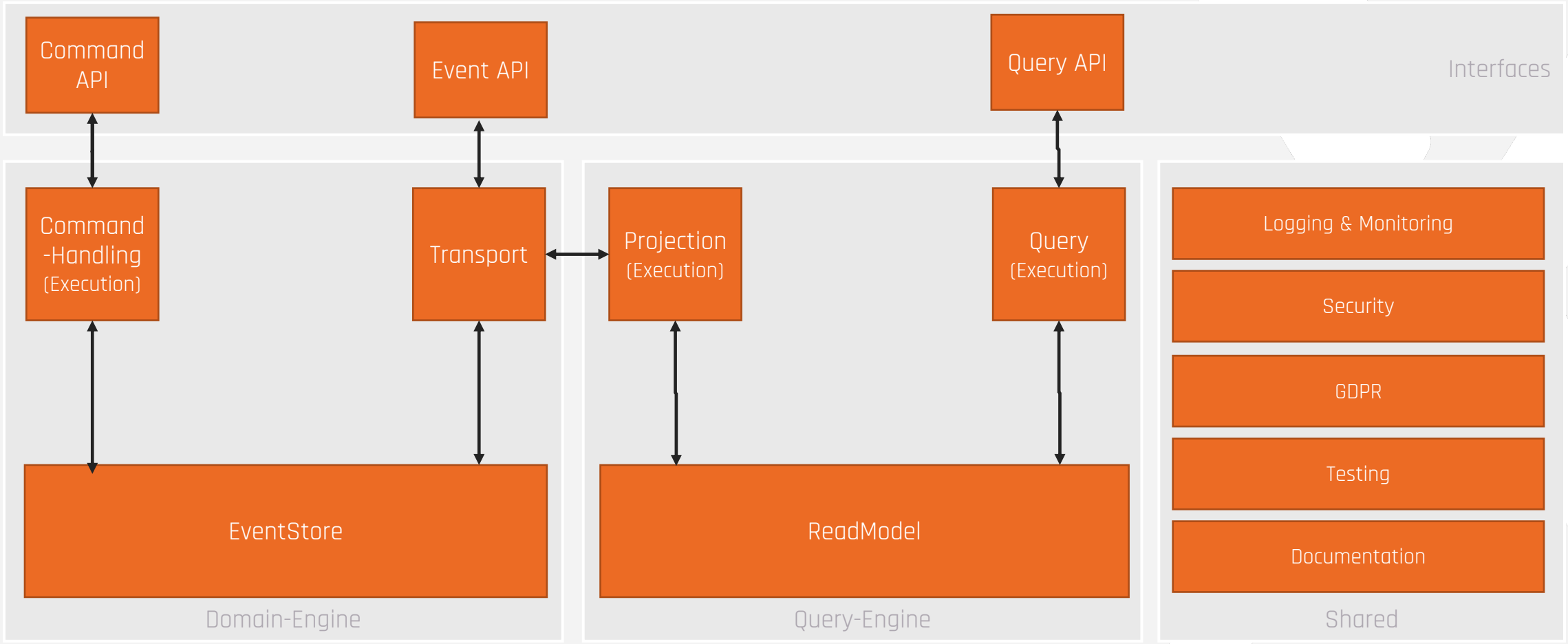


Documentation

- 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



Components overview of an EventSourcing system



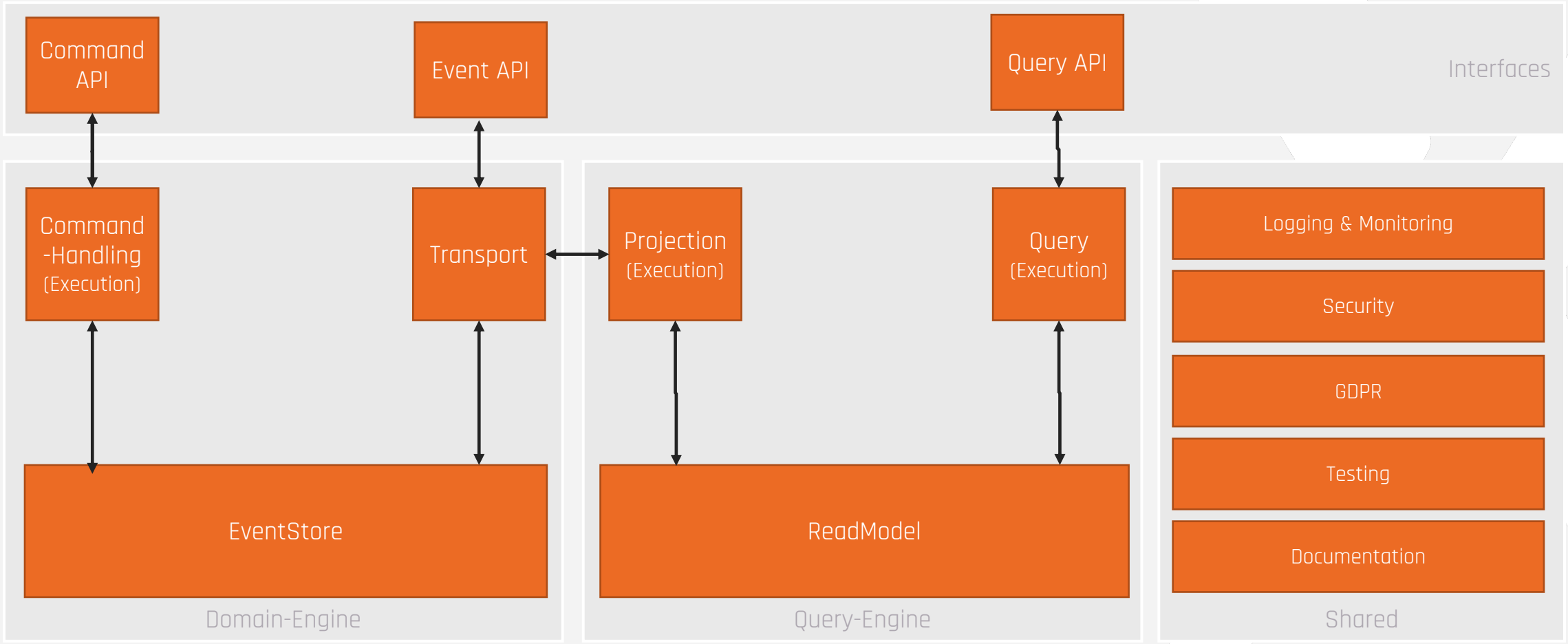
Testing & Debugging

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 independ test environments for each run!
- Go BDD -> SpecFlow as solution! Early, execute frequently.
- Do CDCT, every single time!

Components overview of an EventSourcing system

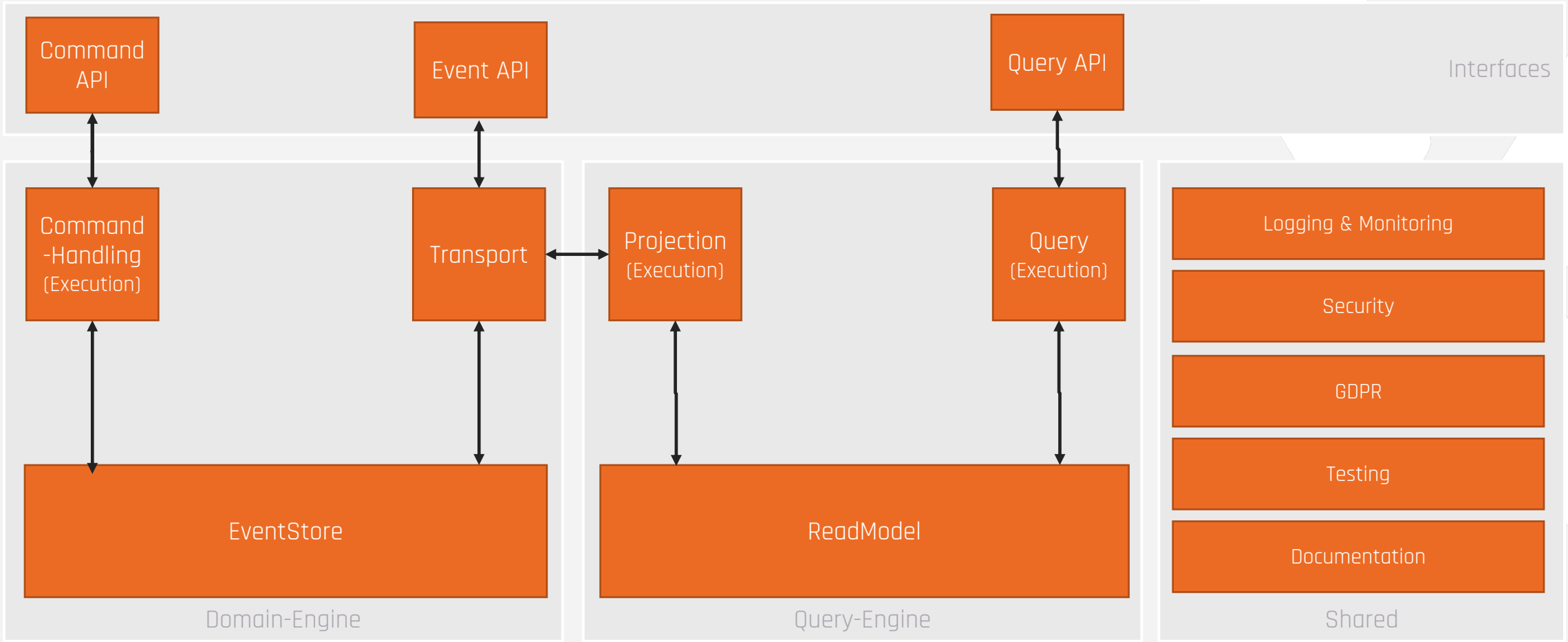


Security

- 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!

Components overview of an EventSourcing system



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!

"Be passionate and bold. Always keep learning. You stop doing useful things if you don't learn."

Satya Nadella

Let's connect

Robin Konrad

Enterprise Architect

Solution Architect

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

rkonrad@xpirit.com

https://robinkonrad.de

@robin_konrad_

@robinkonrad

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

Let's connect!