Integration patterns using NServiceBus & Azure Functions

Dibran Mulder

cloud republic

$> whoami

Dibran Mulder

Azure Solution Architect

Cito, NS, Allego, AkzoNobel

@dibranmulder

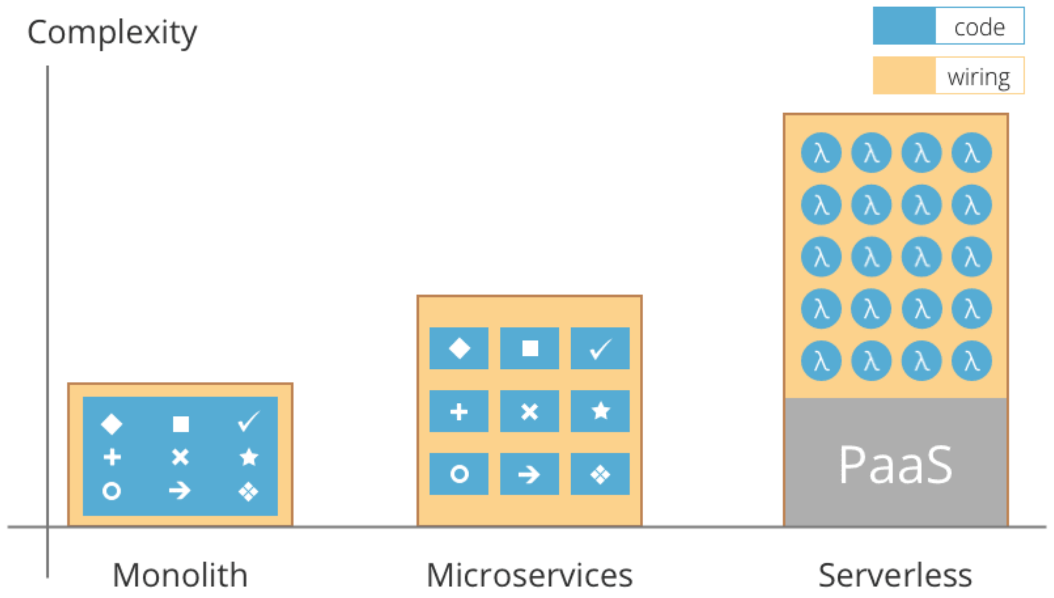
Co-Host

www.devtalks.nl

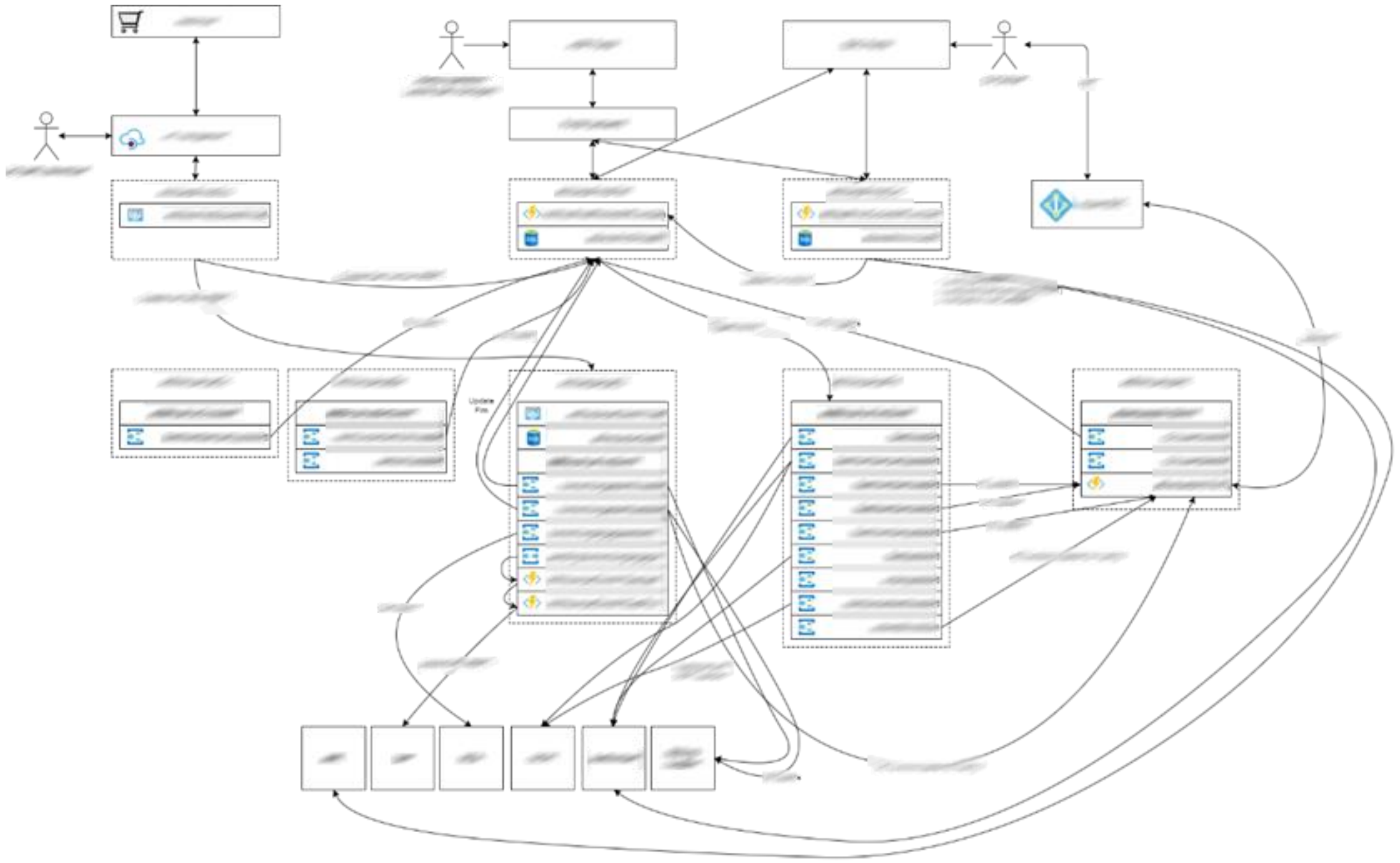
@devtalkspodcast

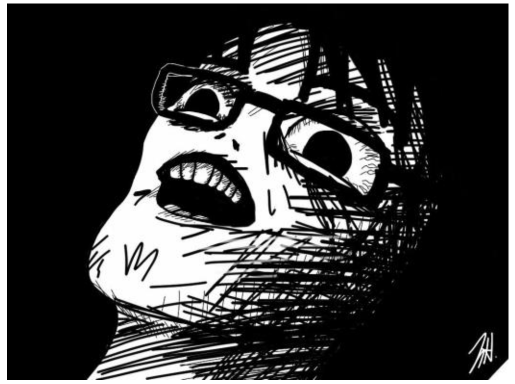
Communcation/Interfaces in different application architectures

- Monolith
  - In-application interfaces
  - DLL's, Com-Interops
- Service Oriented Architecture
  - WCF
  - SOAP
- Microservices
  - REST
  - gRPC
  - Messaging
- Serverless
  - Every piece of functionality has an interface
  - Http, Storage, Service Bus triggers and outputs



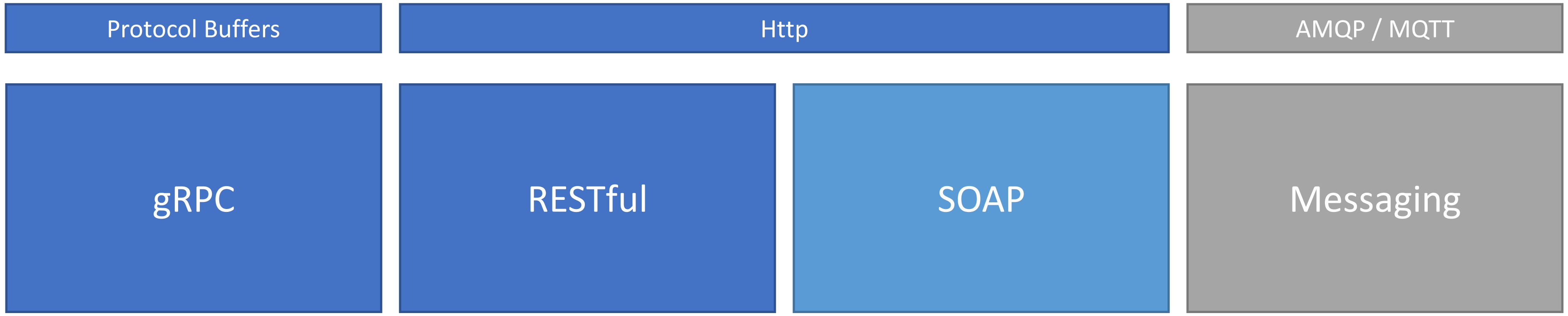
Serverless microservices wiring can quickly become a mess





cloud republic

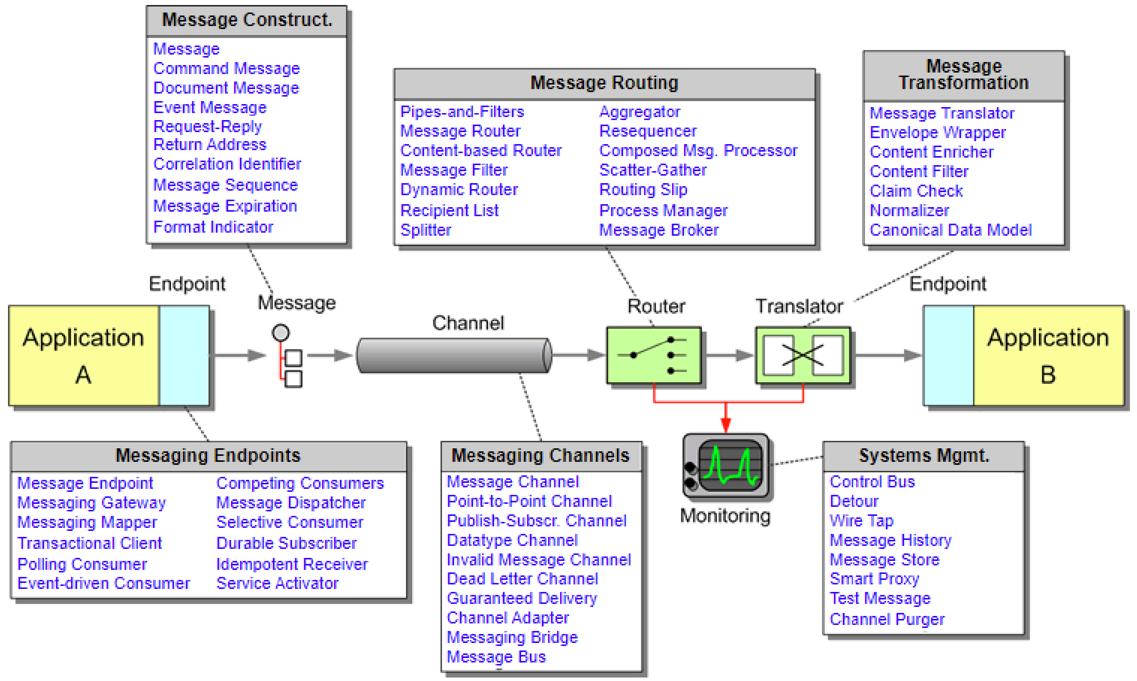
Communication Transport Mechanisms

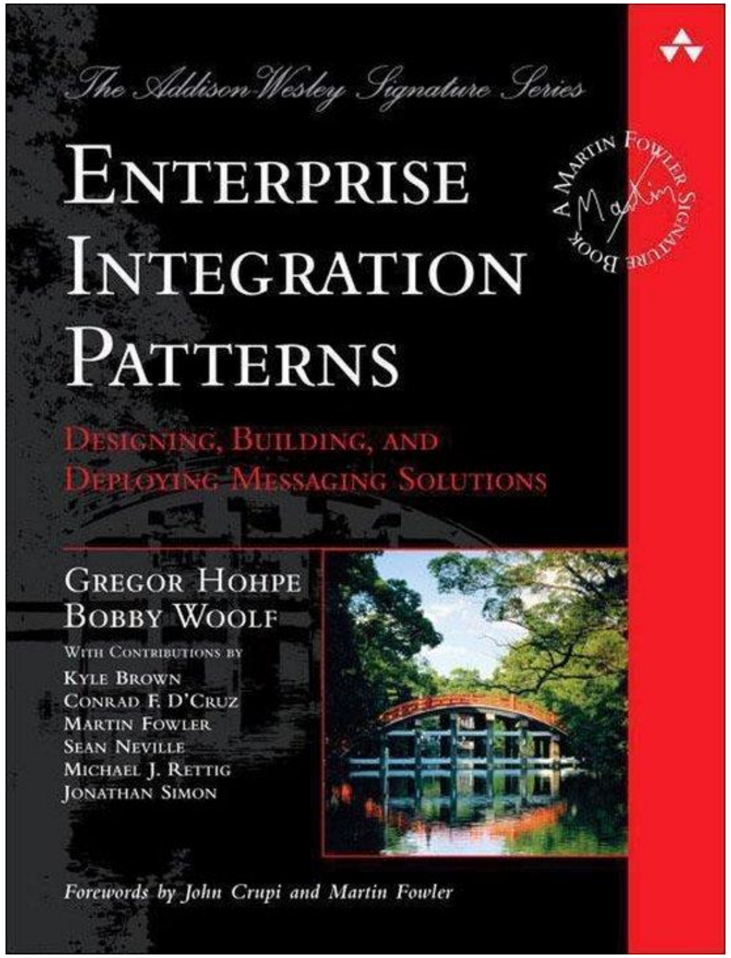


Synchronous

Asynchronous

Enterprise Integration Patterns



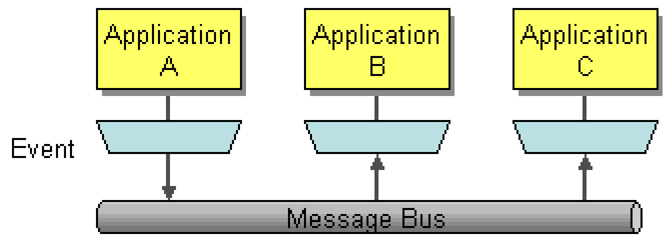

The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
A Martin Fowler Signature Book
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler

Messaging 101

cloud republic

Messaging in a nutshell

- Message Bus / Service Bus is used to transport messages
  - Reliable
  - Asynchronous / Decoupled
- Messages are data records that are transmitted through a message channel
  - Serializable (JSON/XML)
  - Events/Commands
- Messaging is asynchronous per definition
- Messaging changes a developer's mindset
  - Don’t always expect an immediate response
  - What to do if a message can’t be processed



Asynchronous messaging vs REST API’s

- Asynchronous messaging
  - Non-blocking, does not require both systems to be up and ready at the same time.
  - Reduces the release dependencies from a Microservices architecture significantly.
  - Messaging interfaces outside a controlled domain introduce security challenges.
- RESTful API’s
  - Tight coupling, one-to-one communication
  - Blocking, the system has to wait for the response. Useful in scenario’s where for instance user have to get feedback.
  - Error handling, retry logic for when the other systems is down. Increases blocking issue.
    - Potential loss of data

Messaging systems

- ApacheMQ
- RabbitMQ
- AWS
  - Amazon SQS
  - Amazon MQ
- Azure
  - Azure Service Bus
  - Azure Storage Queues
  - EventHub
  - EventGrid
- Traditional Enterprise Service Buses
  - IBM WebSphere
  - MuleSoft
  - FuseESB Redhat

System Types

- Queues
  - Decoupled Fifo storage
- MessageBrokers
  - Validation
  - Transformation
  - Routing
- Enterprise Service Buses
  - Validation
  - Routing
  - Transformation
  - Monitoring
  - Workflows
  - Etc…

cloud republic

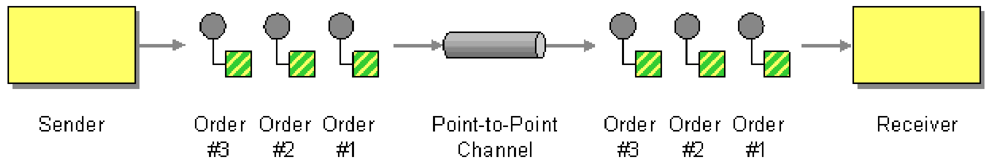
Messaging Channels

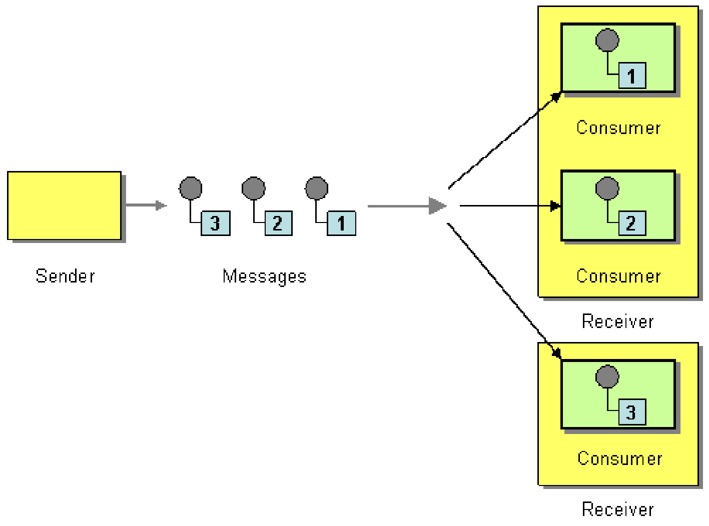
Point-to-Point Channel

- Implemented by for instance an Azure Service Bus Queue

- Messages are Commands or Events
- Message sequence / FIFO
- Load-leveling
  - Decoupling gives the other system time to scale up.

- Competing consumer pattern
  - Scalable workloads
  - Stateless





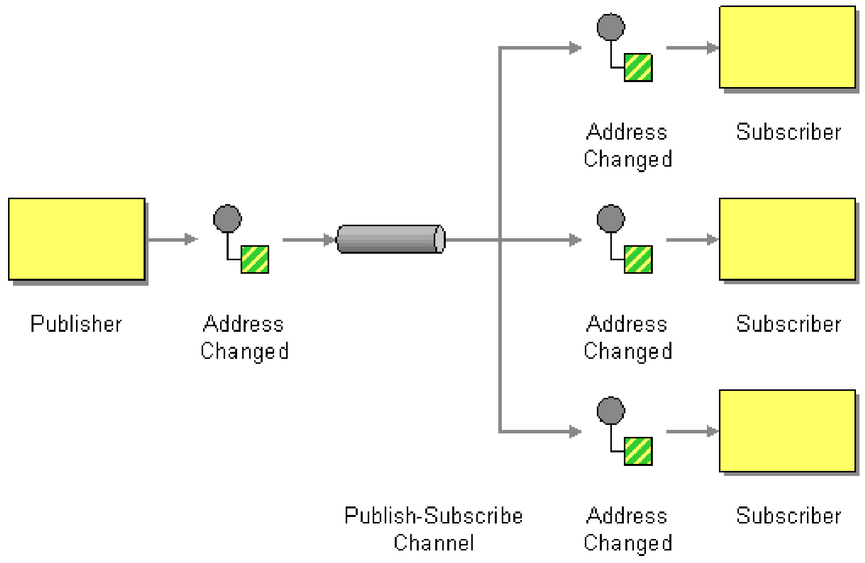
cloud republic

Publish/Subscribe Channel

- Implemented by for instance an Azure Service Bus Topic

- Offers one-to-many communication
    - Copies are delivered to every subscriber
    - Subscribers can consume the message at will
    - Subscribers can optionally filter messages

- Messages are often Events instead of Commands

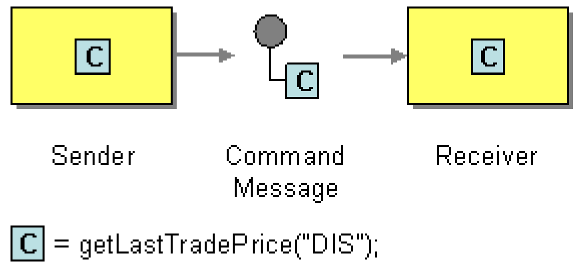


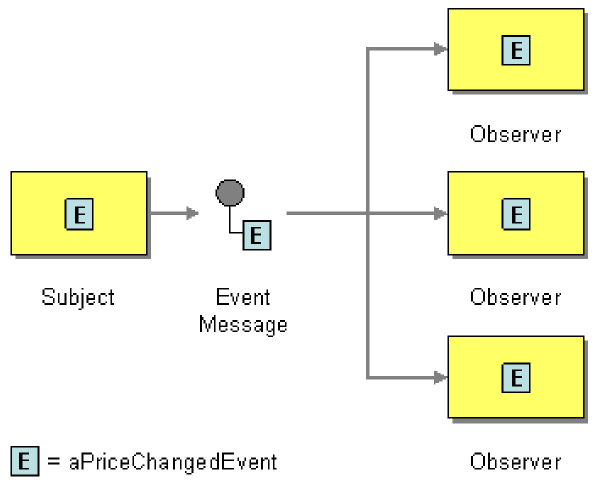
cloud republic

Messaging Patterns

Commands vs Events

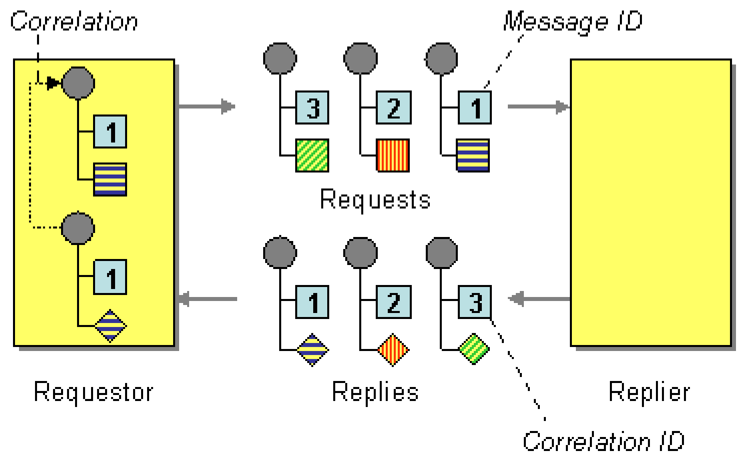
- Commands are actions to another systems
  - Are often used in a point-to-point channel
  - Sometimes require a response (request /reply pattern)
  - Examples, getLastTracePrice, updatePriceForProduct
- Events are notifications to other systems
  - Are often used to notify or update other systems
  - Events can’t be replied
  - Examples: priceUpdatedForProduct, orderProcessed





Request / Reply pattern

- Requestor — An application that performs a business task by sending a request and waiting for a reply.
- Replier — Another application that receives the request, fulfills it, then sends the reply. It gets the request ID from the request and stores it as the correlation ID in the reply.
- Request — A Message sent from the requestor to the replier containing a request ID.
- Reply — A Message sent from the replier to the requestor containing a correlation ID.
- Request ID — A token in the request that uniquely identifies the request.
- Correlation ID — A token in the reply that has the same value as the request ID in the request.



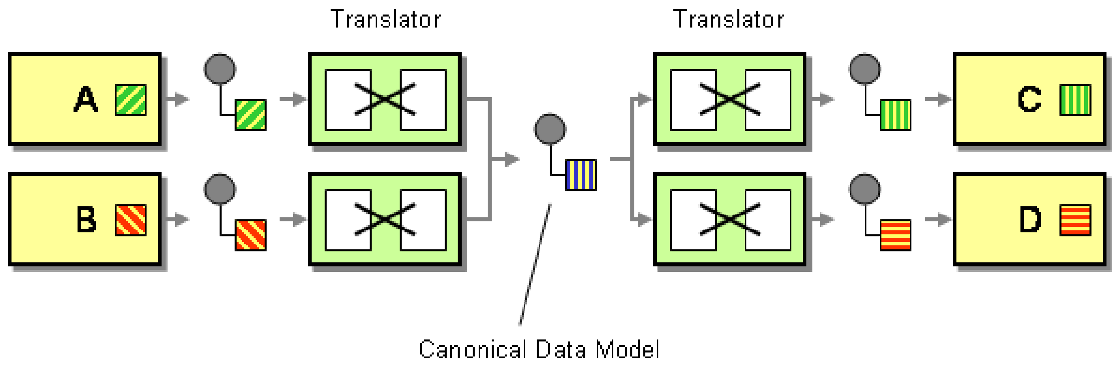
- Requestor could wait and use timeouts.
- Requestor could, if its an API, use websockets to update the user.

Canonical Data Model

- The *Canonical Data Model* provides an additional level of indirection between application's individual data formats. If a new application is added to the integration solution only transformation between the *Canonical Data Model* has to created, independent from the number of applications that already participate.

- XML/JSON definition
  - Version the messages

- Translators
  - .NET Assemblies / NuGet packages
  - NPM TypeScript Type definitions



Messaging in a Serverless Architecture

cloud republic

Azure Functions and Messaging

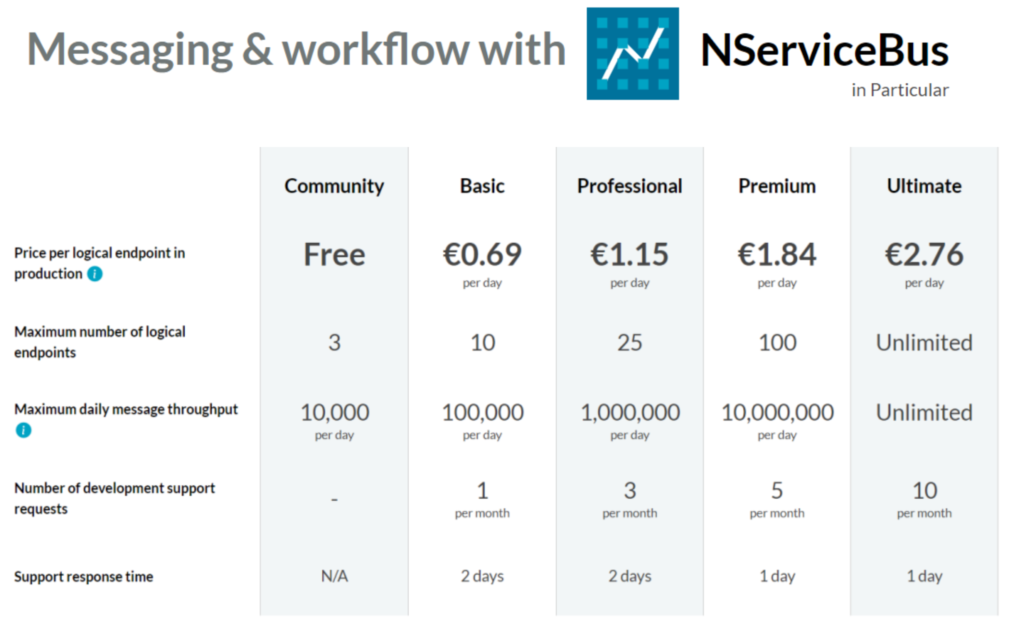
- Why its a good fit
  - Decoupling
    - Enables decoupled communication which enables a Severless Architecture.
  - Messaging improves resiliency
    - Enables retries, Dead letter queues
  - Messaging enables scalable workloads
    - Azure Functions uses the Service Bus message count as a metric to scale up instances
  - Messaging is pay-per-use

- Serverless Architectures require lots of wiring
  - Service Bus messaging for core messaging
  - Trigger on service bus messages.
  - Use Azure Function Bindings to reduce wiring to other interfaces, such as storage, databases, EventGrid, etc.

What is NServiceBus

- .NET Implementation of the Enterprise Integration Patterns
- Additional tools:
  - ServiceInsight visualization of messages, sagas, performance, etc. Looks a lot like Azure Service Bus Explorer.
  - ServicePulse, monitoring, health, error handling

- Disclaimer
  - Community Free Tier
  - Paid professional use
  - I don’t benefit from NServiceBus
  - Particular is not an early adopter
    - Azure Functions Nuget package is not v1 yet.
    - ServiceControl, ServiceInsight, ServicePulse are based on Windows services.

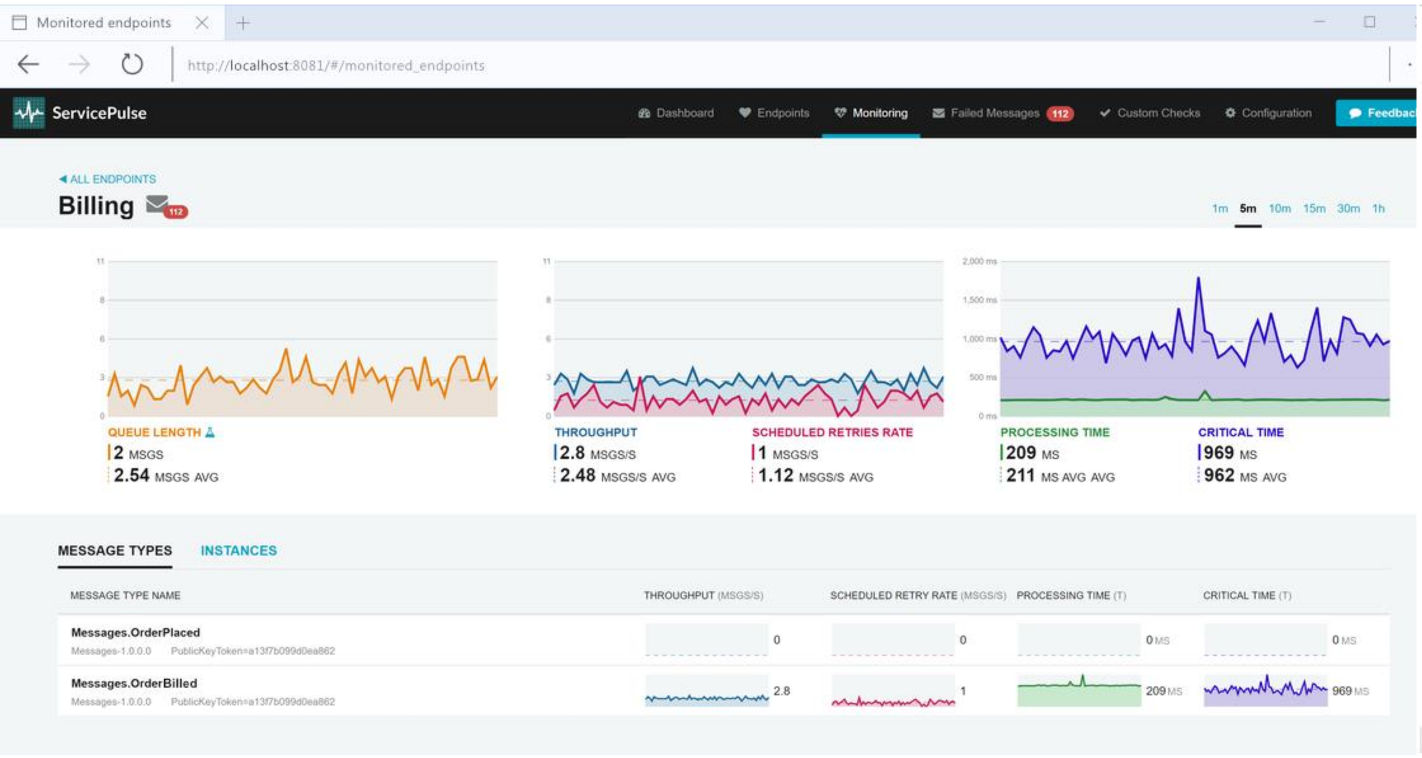


Messaging & workflow with NServiceBus in Particular

	Community	Basic	Professional	Premium	Ultimate
Price per logical endpoint in production	Free	€0.69 per day	€1.15 per day	€1.84 per day	€2.76 per day
Maximum number of logical endpoints	3	10	25	100	Unlimited
Maximum daily message throughput	10,000 per day	100,000 per day	1,000,000 per day	10,000,000 per day	Unlimited
Number of development support requests	-	1 per month	3 per month	5 per month	10 per month
Support response time	N/A	2 days	2 days	1 day	1 day

cloud republic

NServiceBus



QUEUE LENGTH
2 MSGS
2.54 MSGS AVG

THROUGHPUT
2.8 MSGS/S
2.48 MSGS/S AVG

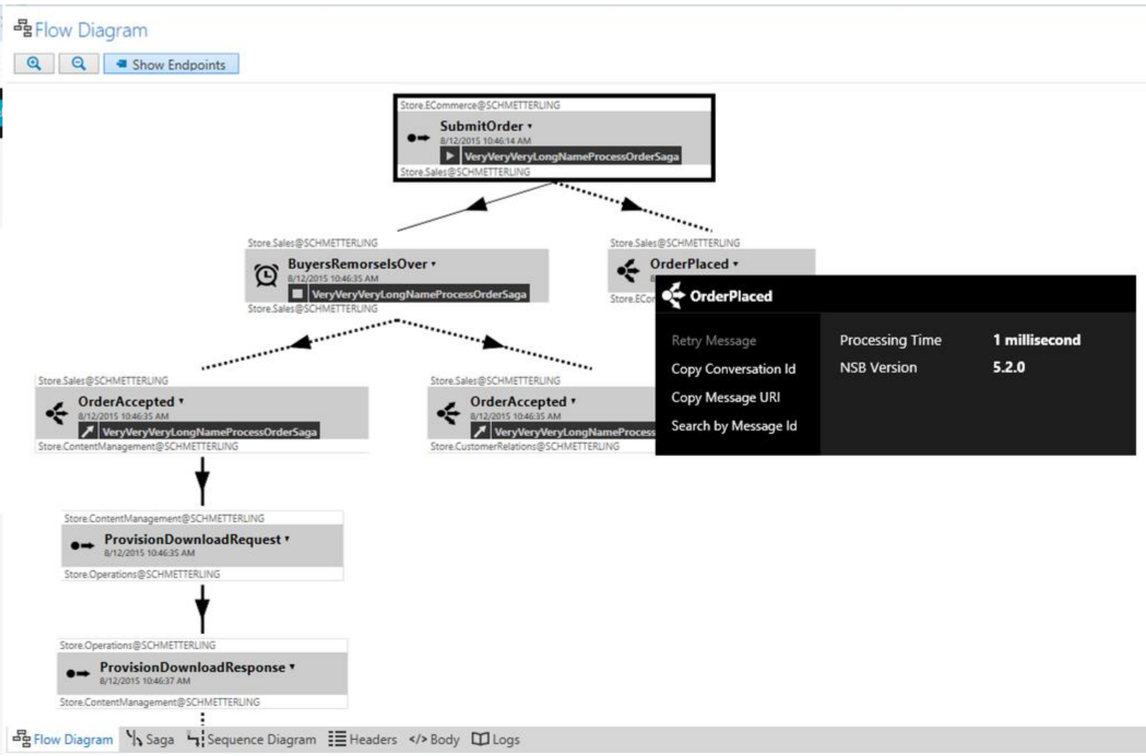
SCHEDULED RETRIES RATE
1 MSGS/S
1.12 MSGS/S AVG

PROCESSING TIME
209 MS
211 MS AVG AVG

CRITICAL TIME
969 MS
962 MS AVG

MESSAGE TYPES INSTANCES

MESSAGE TYPE NAME	THROUGHPUT (MSGS/S)	SCHEDULED RETRY RATE (MSGS/S)	PROCESSING TIME (T)	CRITICAL TIME (T)
Messages.OrderPlaced Messages-1.0.0.0 PublicKeyToken=a13f7b099d0ea862	0	0	0 MS	0 MS
Messages.OrderBilled Messages-1.0.0.0 PublicKeyToken=a13f7b099d0ea862	2.8	1	209 MS	969 MS

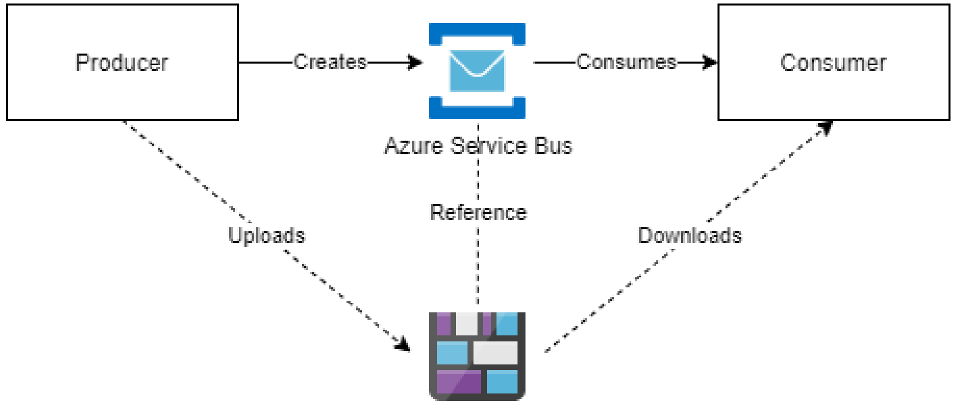

Flow Diagram
Show Endpoints
Store.ECommerce@SCHMETTERLING
SubmitOrder
8/12/2015 10:46:14 AM
VeryVeryVeryLongNameProcessOrderSaga
Store.Sales@SCHMETTERLING
Store.Sales@SCHMETTERLING
BuyersRemorseIsOver
8/12/2015 10:46:35 AM
VeryVeryVeryLongNameProcessOrderSaga
Store.Sales@SCHMETTERLING
Store.Sales@SCHMETTERLING
OrderPlaced
Store.ECor
OrderPlaced
Retry Message
Copy Conversation Id
Copy Message URI
Search by Message Id
Processing Time
1 millisecond
NSB Version
5.2.0
Store.Sales@SCHMETTERLING
OrderAccepted
8/12/2015 10:46:35 AM
VeryVeryVeryLongNameProcessOrderSaga
Store.ContentManagement@SCHMETTERLING
Store.Sales@SCHMETTERLING
OrderAccepted
8/12/2015 10:46:35 AM
VeryVeryVeryLongNameProcess
Store.CustomerRelations@SCHMETTERLING
Store.ContentManagement@SCHMETTERLING
ProvisionDownloadRequest
8/12/2015 10:46:35 AM
Store.Operations@SCHMETTERLING
Store.Operations@SCHMETTERLING
ProvisionDownloadResponse
8/12/2015 10:46:37 AM
Store.ContentManagement@SCHMETTERLING
Flow Diagram
Saga
Sequence Diagram
Headers
</> Body
Logs

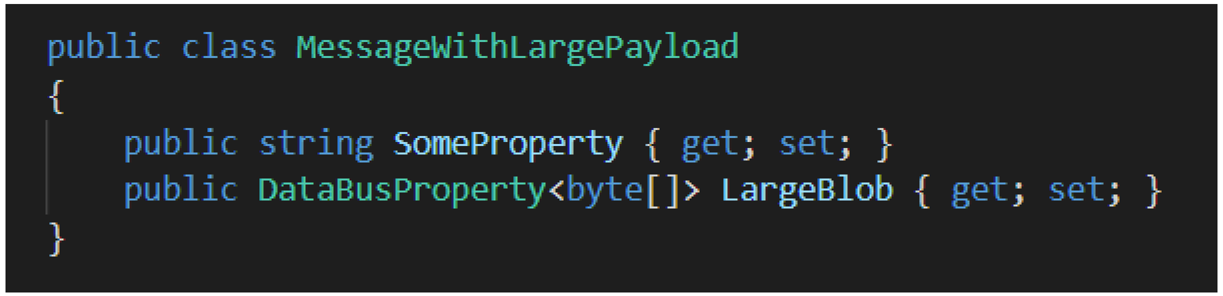
cloud republic

NServiceBus additional patterns - DataBus

- Convenient way of sending large messages or messages with files/documents as payloads.

- Library handles plumbing of storage connection, uploading and downloading of message.
  - Azure Storage and Windows File Share support



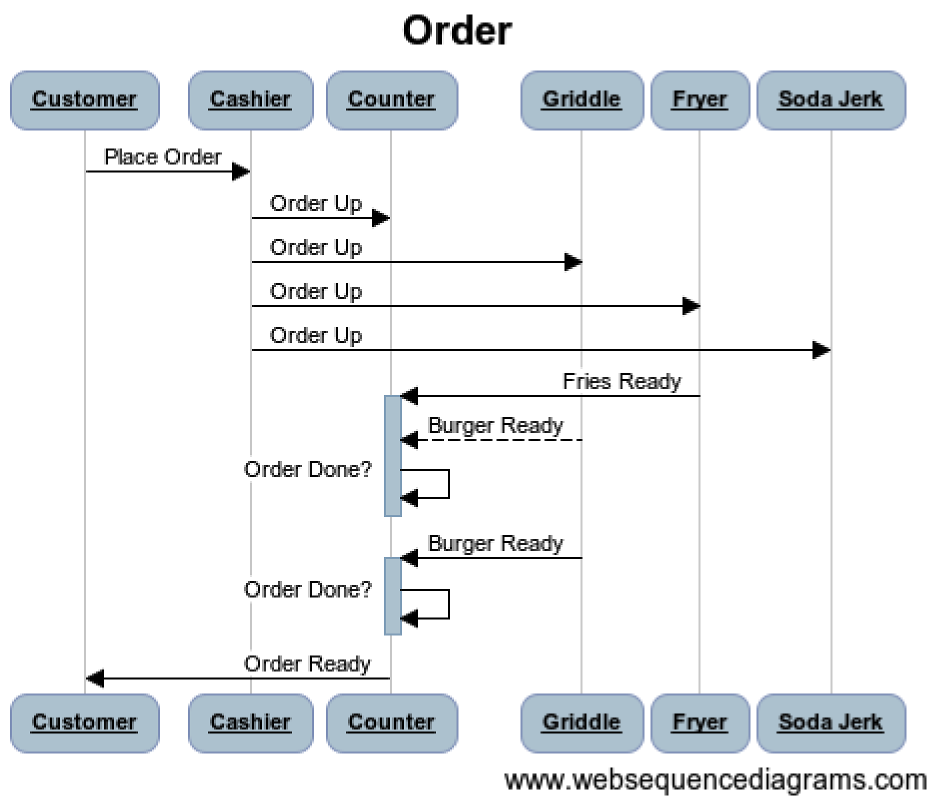


```
public class MessageWithLargePayload
{
    public string SomeProperty { get; set; }
    public DataBusProperty<byte[]> LargeBlob { get; set; }
}
```

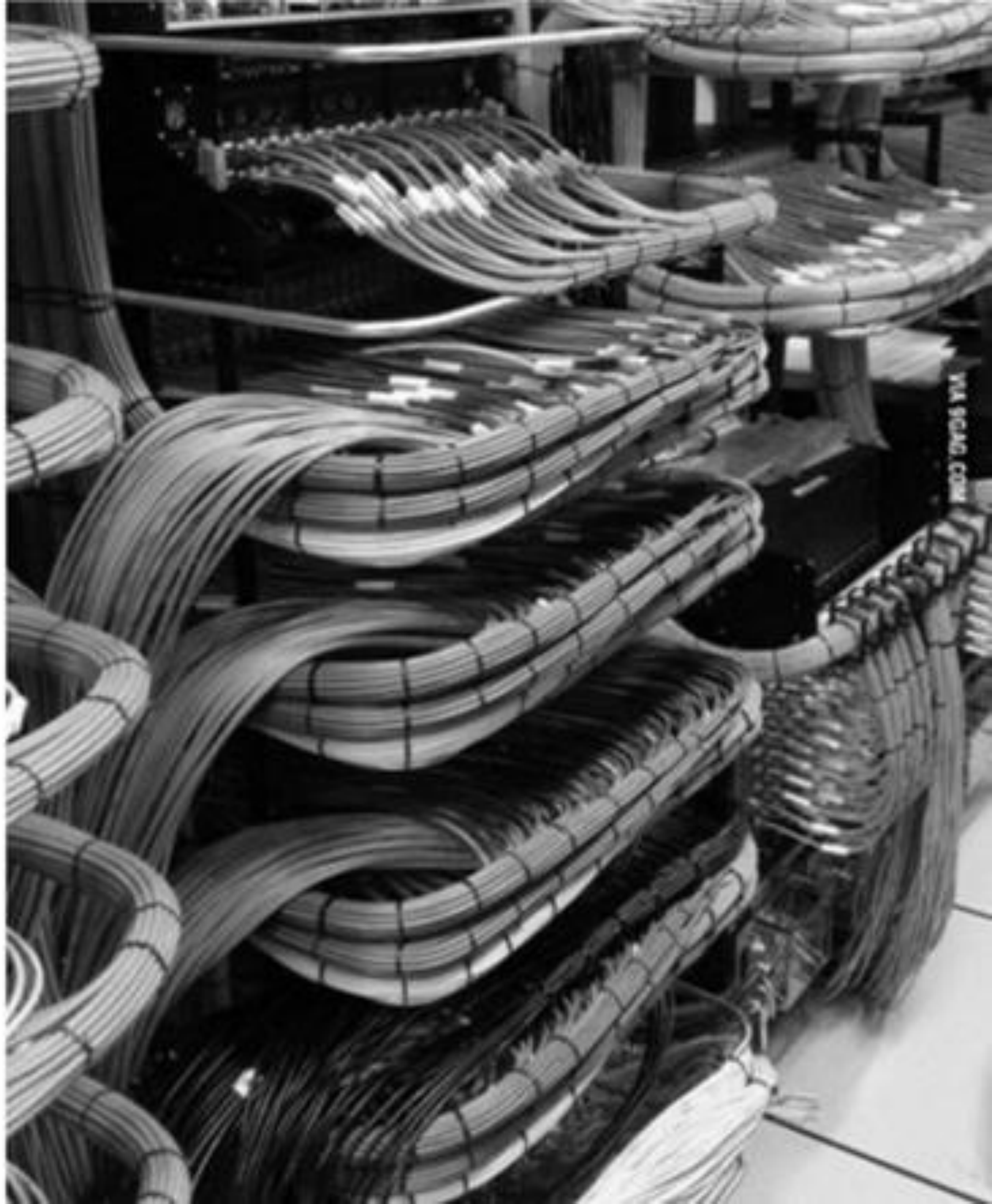
cloud republic

NServiceBus additional patterns – Saga's

- Workflow consisting out of several messages being handled
  - Is started by specific messages
  - Handles certain messages
- Somewhat comparable to Azure Durable Functions / Azure Durable Entities
  - State is stored in persistence of choice
  - Orchestration is handled via Service bus messages.
- NServiceBus Saga persistence
  - SQL Server, MySql, PostgreSql, Azure Table Storage, MongoDb, RavenDb and more.



NServiceBus helps to get from chaos to control



Conclusion

+ +

=

Resilient
Scalable
Cost effective
Maintainable

Serverless Architecture

- Less infrastructure
- Application life cycle
- Pay-per-use
- Scalable
- Bindings/Triggers

Messaging

- Decoupled
- Reliable
- Scalable

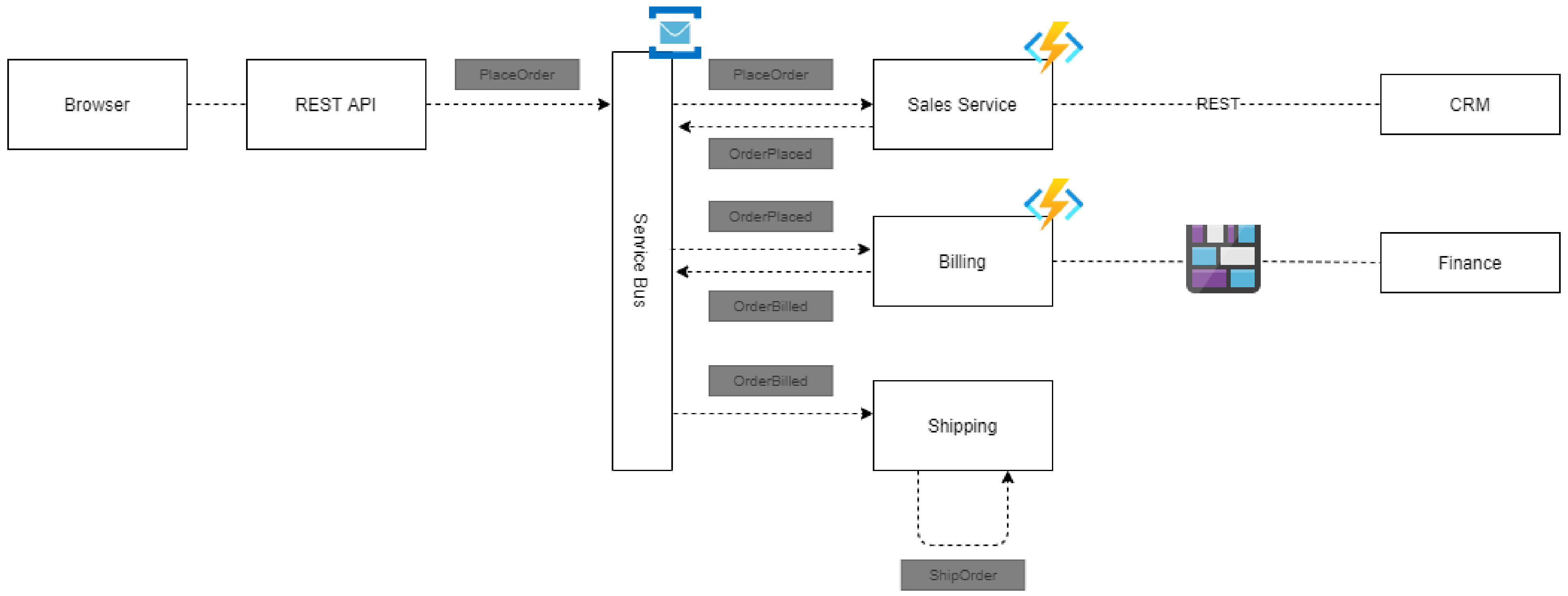
NServiceBus

- Patterns
- Best practices
- Tooling

Demo Time

Demo

- Commands, Events, Saga’s, Versioning



https://github.com/DibranMulder/NServiceBus-Serverless-Demo

cloud republic

Questions?