SafeBunny

A .NET RabbitMQ Library

Andrei Mihaila

Dr. Florin Olariu

Agenda

- Problems in distributed systems
- Classic solutions
- Why SafeBunny
- Architecture of SafeBunny
- Solving concurrency problems
- Future work

Problems in distributed systems

- Load distribution and the Infrastructure Layer

Problems in distributed systems

- Load distribution and the Infrastructure Layer
  - Non deterministic network
  - Unreliable dependencies
  - Concurrency bugs quickly become epidemic
  - A retry at the top translates to a lot more further down
  - Not understanding the depths of the infrastructure

Classic Solutions

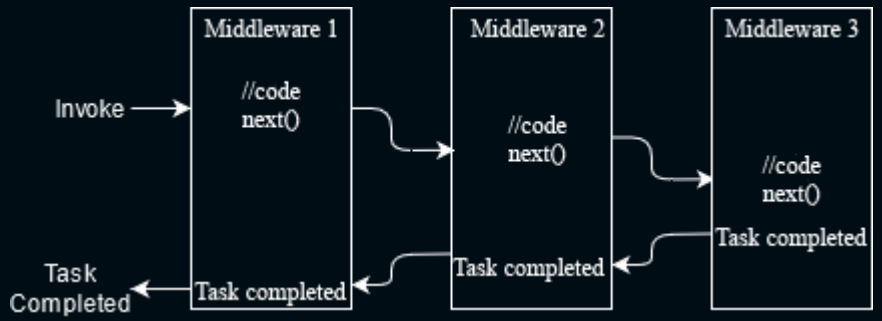
- Idempotency tokens
- Relying on ACID
- Domains idempotent by design
- Long running processes

SafeBunny

- .NET Library abstracting RabbitMQ
- Domain friendly APIs
- Automatic topology creation
- Flexibility and Extensibility
- Retries and deduplication
- Thread safety

SafeBunny - Architecture

- Fully modular
- Components can be added anywhere
- Pipeline can be short-circuited
- All logic is built in modules



Why this library? – Early stage

Application

- Easy and fast integration
- Easily testable
- Quick understanding

SafeBunny

- Low parameter startup
- Vertical architecture mindset
- RabbitMQ abstracted away
- Automatic topology

Why this library? – Setup

```
internal sealed class InvoiceModelMessageHandler : IMessageHandler<InvoiceModel>
{
    public Task HandleAsync(IProcessingContext<InvoiceModel> context)
    {
        // code
        return Task.CompletedTask;
    }
}
```

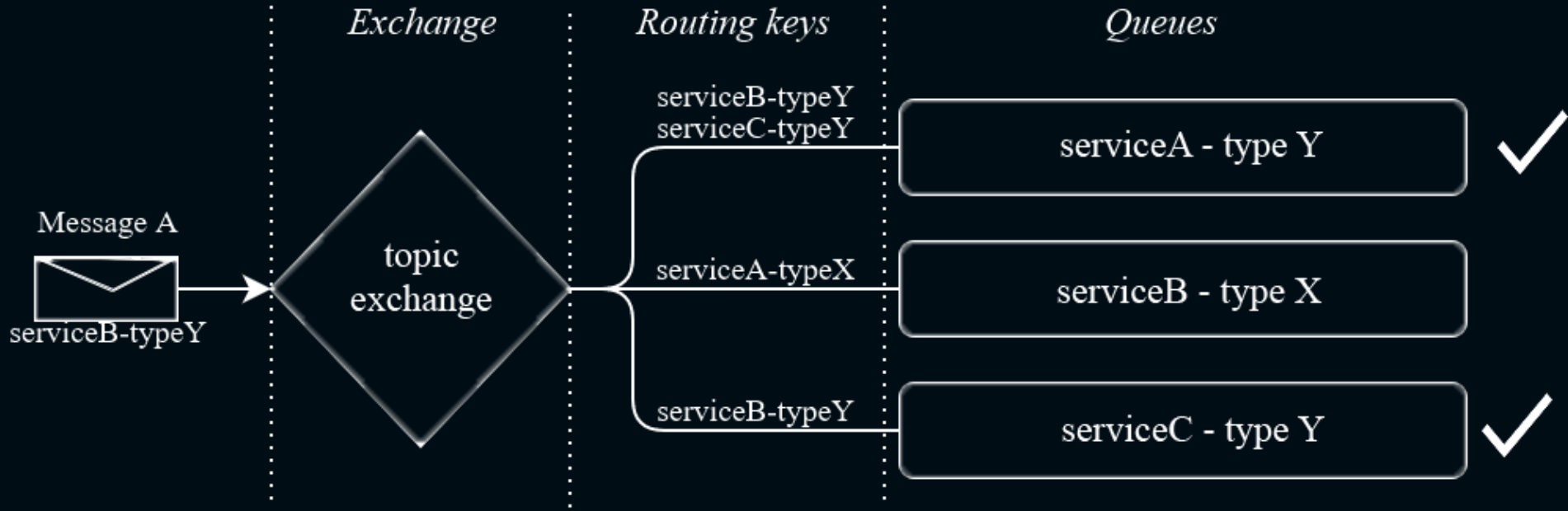
```
host.Services.UseSafeBunnySubscriptions(subscriptions =>
    {
        subscriptions.FromNode("invoicing-service")
            .Consume<InvoiceModel>(10, RetryStrategy.From(5, TimeSpan.FromMilliseconds(100)), RetryStrategy.Default);
    }
);
```

Maximum concurrency

In memory retries

Infrastructure delayed retries

Why this library? – Automatic routing



Why this library? – Middle stage

Application

- Cost saving optimizations
- Quick expansion
- Multi instance debugging

SafeBunny

- Extensible
- Optimized and benchmarked
- Built for multi instance

Why this library? – Extensibility

```
internal sealed class BeforePublishingMiddleware : ISafeBunnyMiddleware<IPublishingContext>
{
    public Task InvokeAsync(IPublishingContext context, Func<Task> next)
    {
        return next();
    }
}
```

```
internal sealed class AfterPublishingMiddleware : ISafeBunnyMiddleware<IPublishingContext>
{
    public Task InvokeAsync(IPublishingContext context, Func<Task> next)
    {
        return next();
    }
}
```

```
host.Services.UseSafeBunnyPrePublisher<BeforePublishingMiddleware>();
host.Services.UseSafeBunnyPostPublisher<AfterPublishingMiddleware>();
```

Why this library? – Optimizations

Pipeline optimizations

Method	Mean	Error	StdDev	Ratio	Gen 0	Gen 1	Gen 2	Allocated
Scoped	2.623 µs	0.0167 µs	0.0156 µs	1.00	0.5035	-	-	2 KB
ScopedActivator	3.603 µs	0.0266 µs	0.0236 µs	1.37	0.4463	-	-	2 KB
Singleton	2.194 µs	0.0235 µs	0.0220 µs	0.84	0.3700	-	-	2 KB

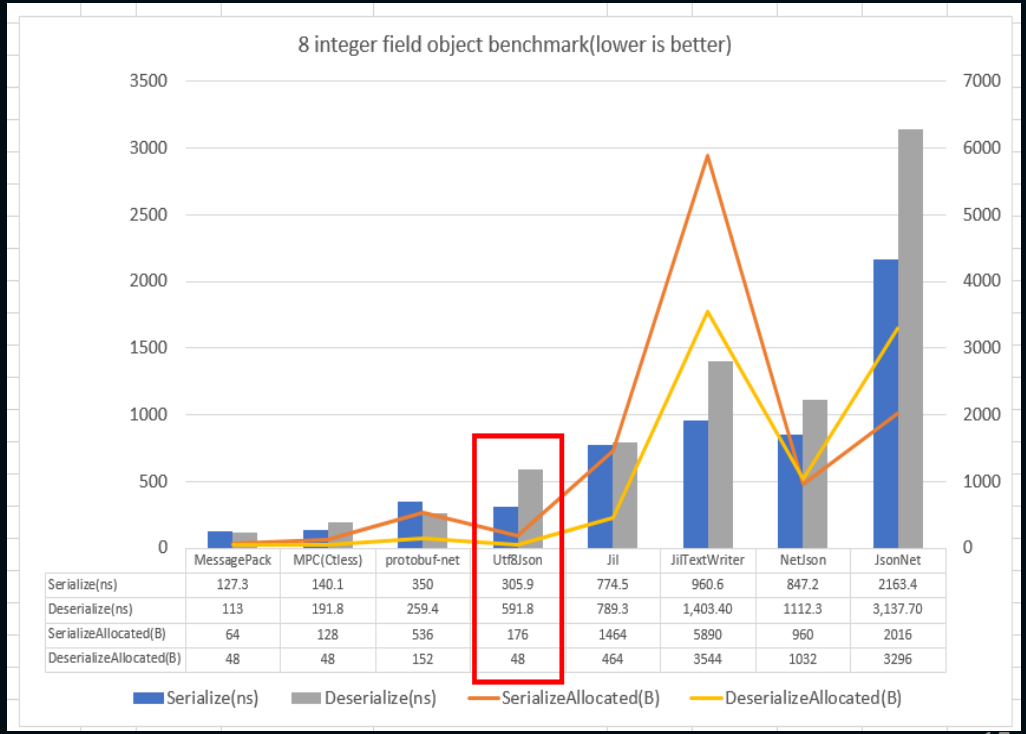
Why this library? – Optimizations

Reflection optimizations

Method	Mean	Error	StdDev	Ratio	Gen 0	Gen 1	Gen 2	Allocated
RawReflection	6.492 µs	0.0299 µs	0.0265 µs	1.00	0.7782	-	-	3 KB
CachedReflection	5.242 µs	0.0249 µs	0.0233 µs	0.81	0.6638	-	-	3 KB

Why this library? – Optimizations

Fast serialization



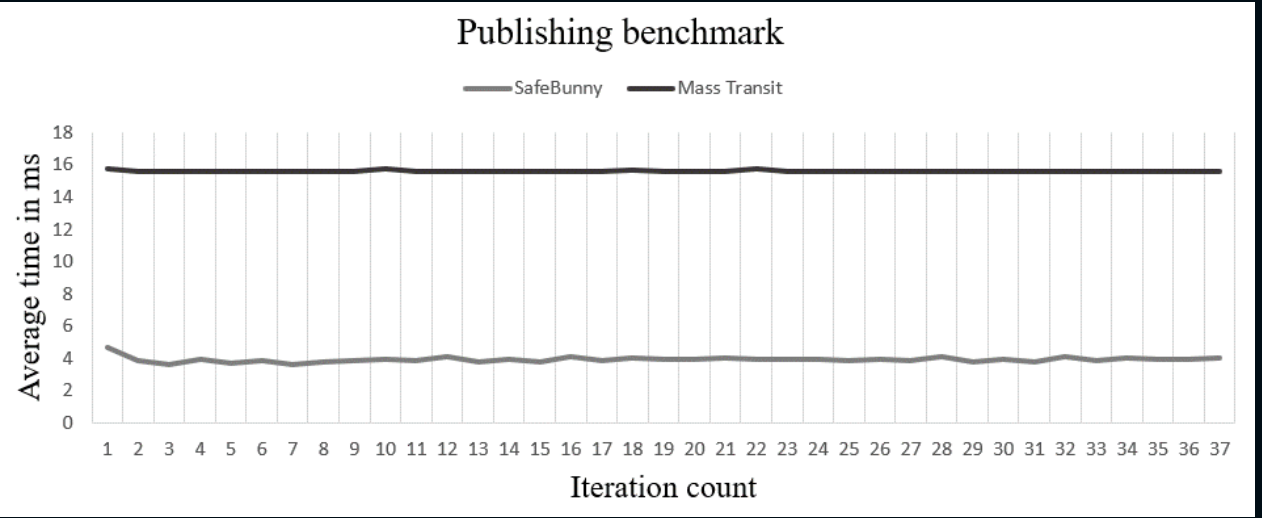
	MessagePack	MPC(Ctless)	protobuf-net	Utf8Json	Jil	JilTextWriter	NetJson	JsonNet
Serialize(ns)	127.3	140.1	350	305.9	774.5	960.6	847.2	2163.4
Deserialize(ns)	113	191.8	259.4	591.8	789.3	1,403.40	1112.3	3,137.70
SerializeAllocated(B)	64	128	536	176	1464	5890	960	2016
DeserializeAllocated(B)	48	48	152	48	464	3544	1032	3296

Why this library? – Performance

Publishing benchmark

8 + 16 + 16 runs
128 iterations each

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
SafeBunny_Publish	3.931 ms	0.0924 ms	0.0907 ms	-	-	-	4 KB
MassTransit_Publish	15.595 ms	0.0198 ms	0.0165 ms	-	-	-	25 KB

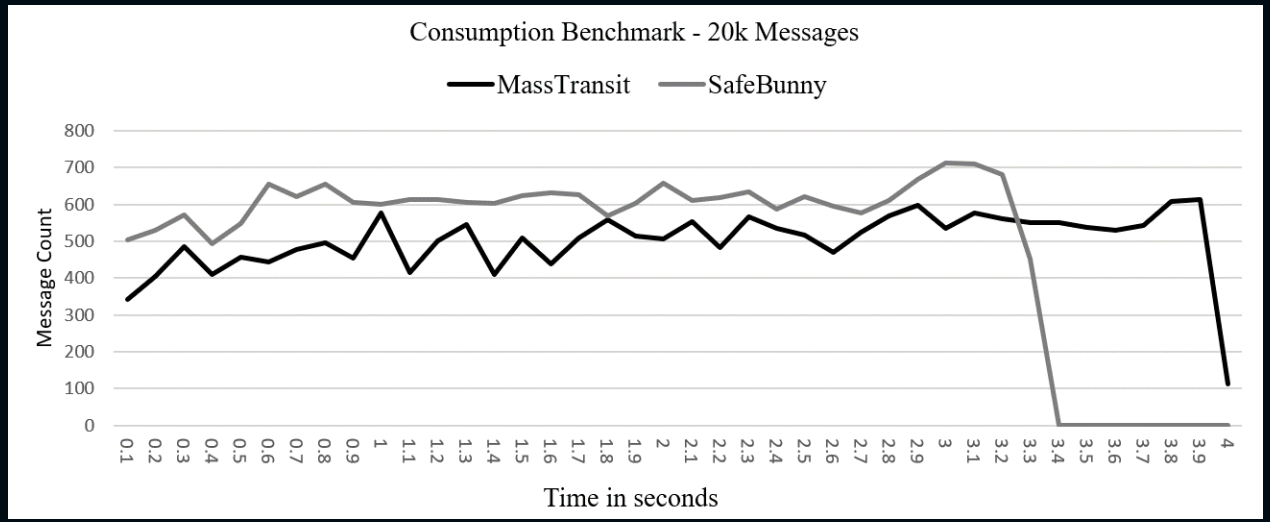


3 Node RabbitMQ Cluster – 4 cores, 16 GB RAM

Why this library? – Performance

Consumption benchmark

20k messages consumed



3 Node RabbitMQ Cluster – 4 cores, 16 GB RAM

Why this library? – End stage

Application

- Concurrency bugs
- Ghost messages
- High load

SafeBunny

- Two retry mechanisms
- Automatic deduplication
- Built-in continuation
- Per type circuit breaker

Why this library? – Deduplication

- Continuation of messages
  - Immutable states
- Deterministic message ids

Why this library? – Deduplication

- Deterministic message ids
  - Correlation - *db19dadf-39d7-4529-836d-cb89e97bfb48*
  - Marker: *27* - *0000000-0000-0000-0000-0000000000ab*
  - Result - *db19dadf-39d7-4529-836d-cb89e97bfbab*

Why this library? – Deduplication

- Distributed deduplication

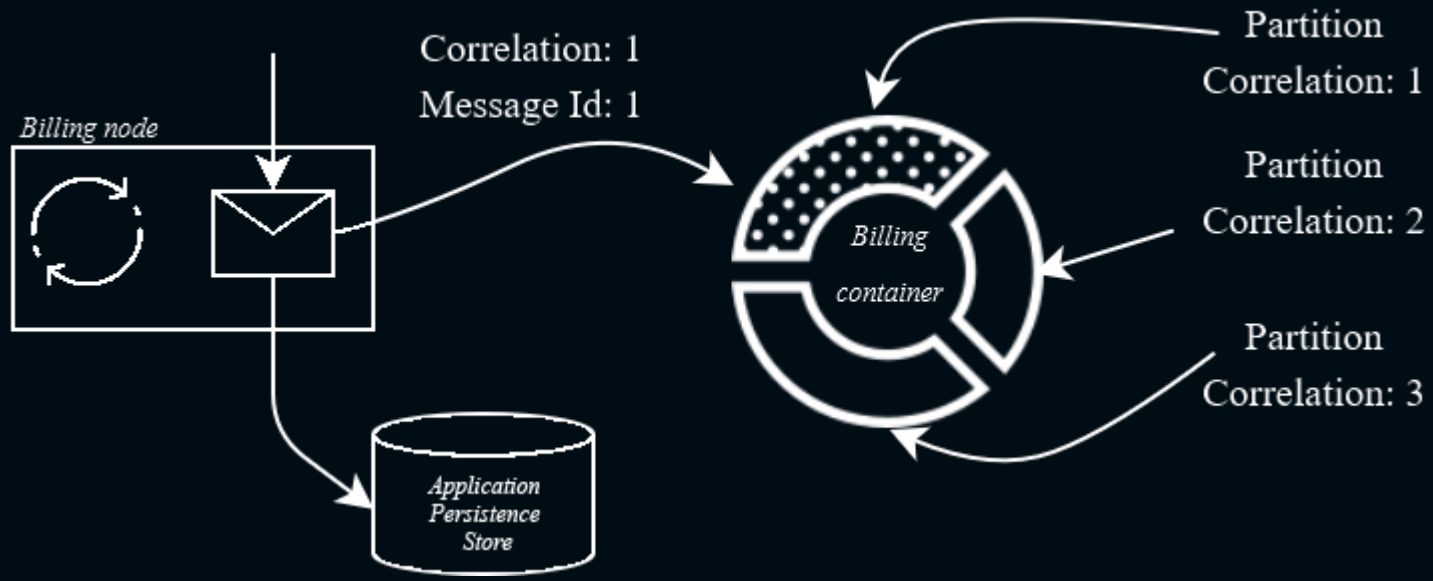
Why this library? – Deduplication

- Distributed deduplication
  - Deduplicate a message across all instances of a service
  - Do not be bound by the performance of ACID databases
  - Only interested in the transactions performed by the current service

Why this library? – Deduplication

- Cosmos DB
  - Stores data in containers -> Mapped to a service
  - Partitioned by design -> Correlation id
  - Transactions are partition **scoped**

Why this library? – Deduplication



Why this library? – Deduplication

```
internal sealed class InvoiceModelMessageHandler : IMessageHandler<InvoiceModel>
{
    public async Task HandleAsync(IProcessingContext<InvoiceModel> context)
    {
        await context.TransactAsync(() => SaveToDatabase(context.Message));
        await context.ContinueAsync(new InvoiceSavedModel());
    }

    public Task SaveToDatabase(InvoiceModel model)
    {
        return Task.CompletedTask;
    }
}
```

Action time

Demo Demo Demo Demo Demo Demo

Future work

- **In-memory transport**
- **Real time encryption**
- **More transaction store options**
- **Integration in another language**

Questions