Parallel and Distributed Programming

Hello!

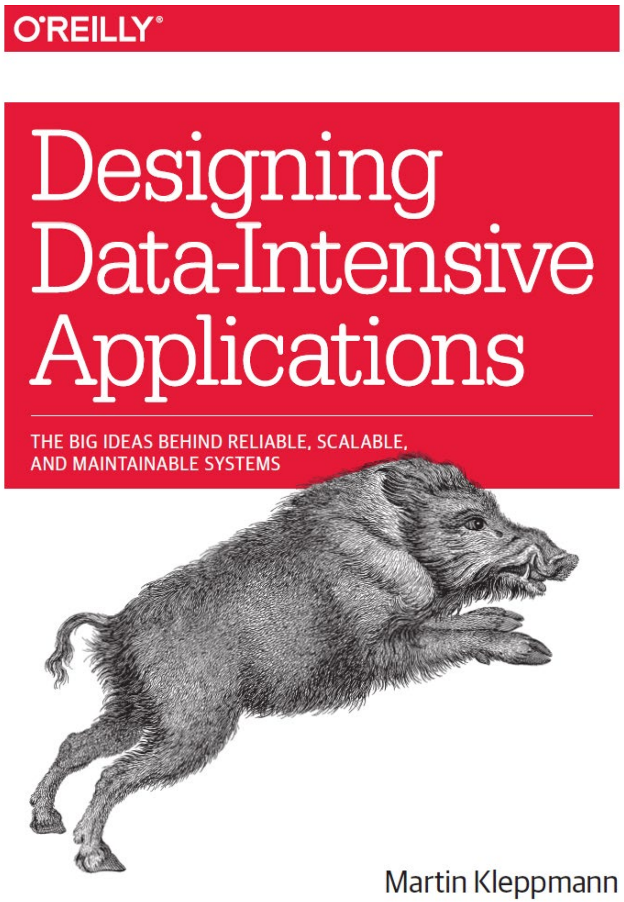
I am Diego Bonura

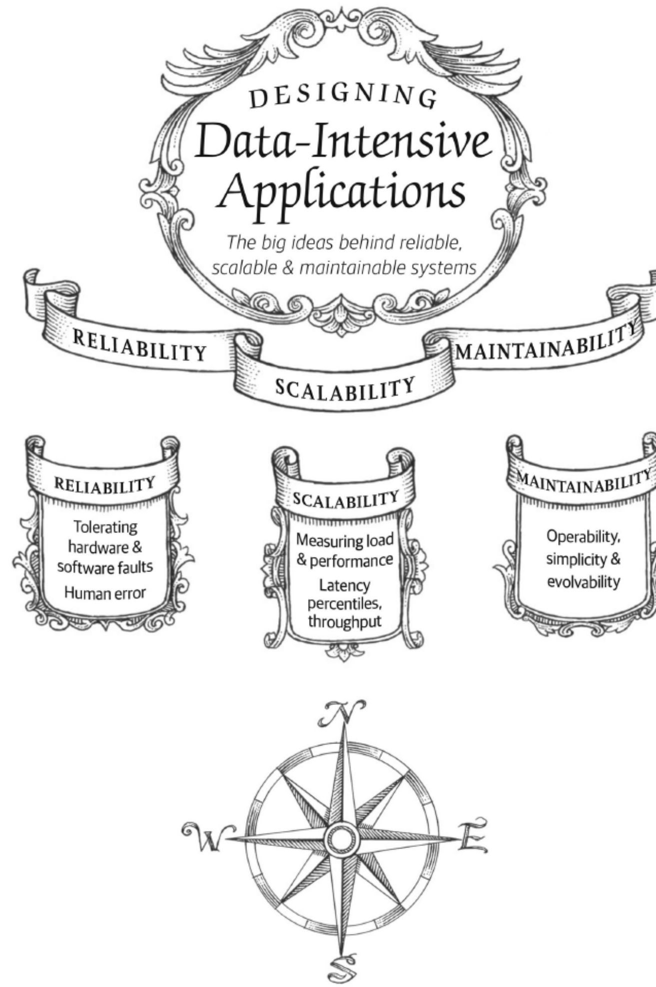
Mi occupo di:

- Frontend
- Backend
- Mobile
- IoT
- R&D

diego@bonura.dev

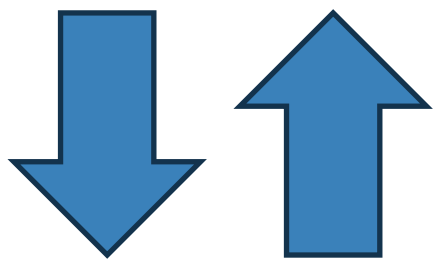
https://medium.com/@diegobonura


O'REILLY
Designing Data-Intensive Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS
Martin Kleppmann


DESIGNING
Data-Intensive Applications
The big ideas behind reliable, scalable & maintainable systems
RELIABILITY
SCALABILITY
MAINTAINABILITY
RELIABILITY
Tolerating hardware & software faults
Human error
SCALABILITY
Measuring load & performance
Latency percentiles, throughput
MAINTAINABILITY
Operability, simplicity & evolvability
N
W
E
S

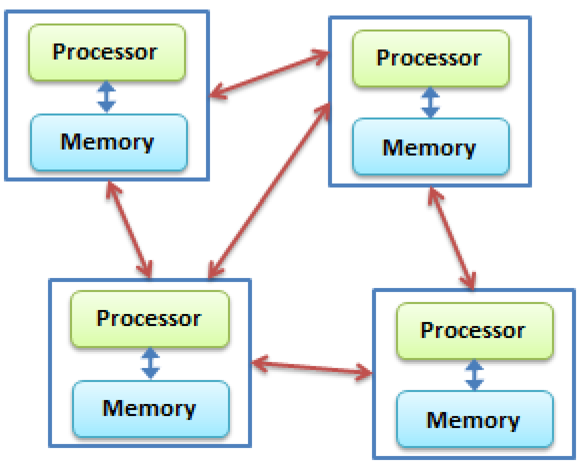
“

Distribuited programming is complex

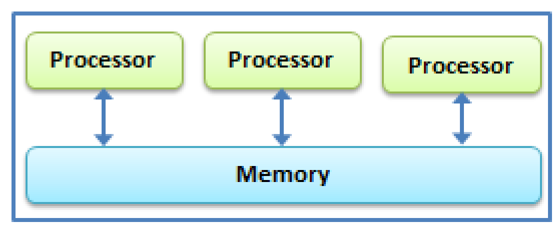


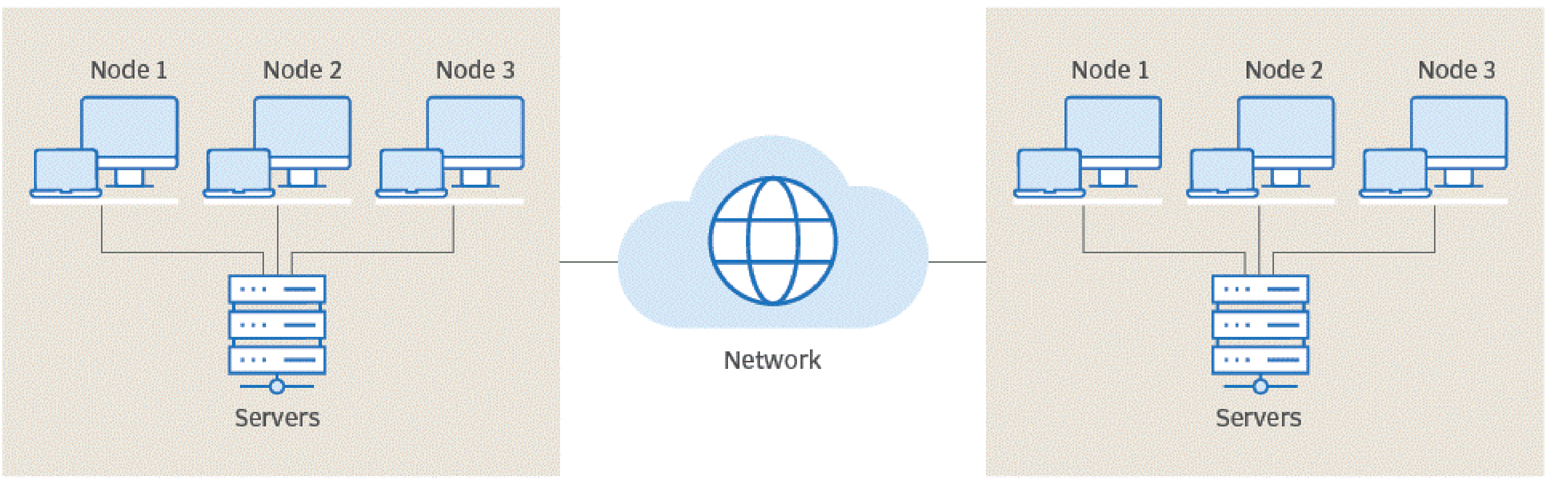
Use only on complex applications

Distributed Computing



Parallel Computing





Node 1
Node 2
Node 3
Servers
Network
Node 1
Node 2
Node 3
Servers

Why?

- **Performance**
  - Maintains System Performance During High Demand Periods
  - Adapts to the Increase/Decrease Workloads and User Demands
- **Scalability**
  - Boosts Performance and Utilization through Collaboration
- **Resilience**
  - Ensures System Continuity in the Face of Failures
- **Redundancy**
  - Enhances User Experience with Geographically Distributed Systems

https://youtu.be/CZ3wIuvmHeM?si=eHIQEqZkHpZWhHDm&t=604

How?

Main types:

- Cluster Computing
  - https://www.mongodb.com/basics/clusters
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/high-availability.html
- Grid computing
  - https://en.wikipedia.org/wiki/Great_Internet_Mersenne_Prime_Search
  - https://en.wikipedia.org/wiki/SETI@home
- Cloud computing
  - https://www.linkedin.com/pulse/how-cloud-computing-made-netflix-possible-keimo-edwards/
  - https://cloudacademy.com/blog/aws-reinvent-netflix/
- Peer-2-Peer
  - Torrent
  - Bitcoin

Example of complex system?

Two of Twitter's main operations are:

Post tweet

- A user can publish a new message to their followers (4.6k requests/sec on average, over 12k requests/sec at peak).

Home timeline

- A user can view tweets posted by the people they follow (300k requests/sec)….
- ….

Continue to book «Designing Data-Intensive Applications» page 11

Main agenda

- Object oriented programming (message passing)
- Async programming
- In-process / out-of-process programming
- Distributed programming
  - Message brokers
  - Actor Model
  - Serialization
  - Transaction
  - Saga
  - Idempotent operations
  - Stream processing
  - Event sourcing
- Deploy a distributed application
- Infrastructure as code
- Update and maintain
- Observability

How to start?

docker

https://visualstudio.microsoft.com/it/vs/community/

or

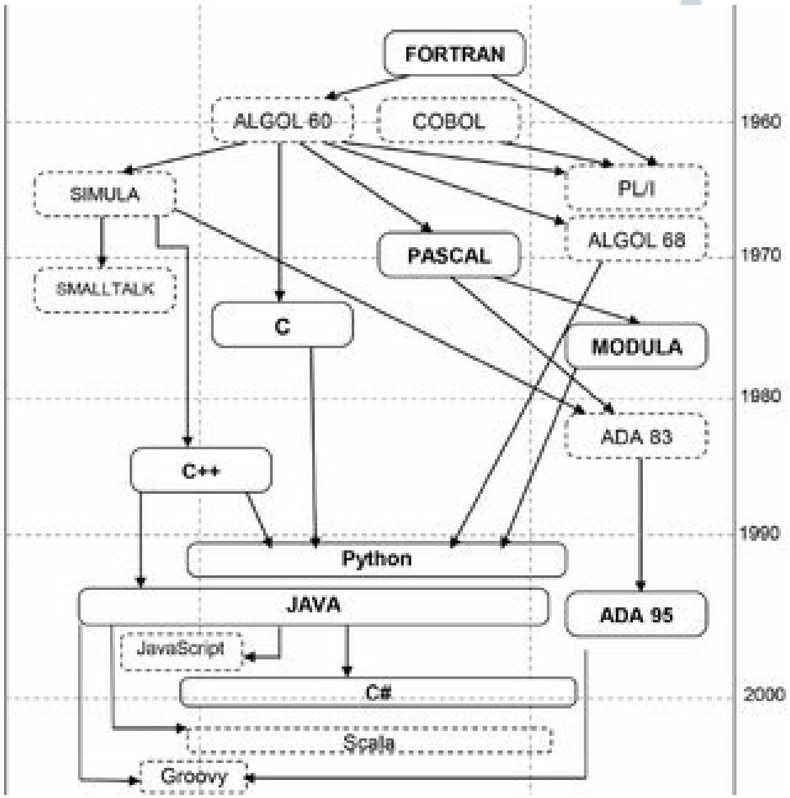
https://code.visualstudio.com/

https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csdevkit

How to start?

https://github.com/meriturva/Parallel-and-Distributed-Programming

Message Passing



Smaltalk: A *message* is simply a method call on an object. Smalltalk messages are perfectly synchronous (the caller waits for the callee to return a value), and not terribly different then function/method calls in other languages.

https://www.researchgate.net/publication/260447599_An_Evaluation_Framework_and_Comparative_Analysis_of_the_Widely_Used_First_Programming_Languages

Message Passing

Message passing is a technique for invoking behavior

```
public class Producer
{
    public void Start()
    {
        var consumer = new Consumer();
        int i = 0;
        while (true)
        {
            var result = consumer.Elaborate(i, i);
            Console.WriteLine($"Counter: {i} with result: {result}");
            i++;
        }
    }
}
```

Example project: 01 MessagePassing

https://en.wikipedia.org/wiki/Message_passing

Async programming

Code run in the background while other code is executing.

```
public class Producer
{
    public async Task StartAsync()
    {
        var consumer = new Consumer();
        int i = 0;
        while (true)
        {
            var result = await consumer.ElaborateAsync(i, i);
            Console.WriteLine($"Counter: {i} with result: {result}");
            i++;
        }
    }
}
```

Example project: 02 AsyncAwait

On the C# side of things, the compiler transforms your code into a state machine that keeps track of things like yielding execution when an await is reached and resuming execution when a background job has finished.

https://devblogs.microsoft.com/dotnet/how-async-await-really-works/#async/await-under-the-covers

https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios

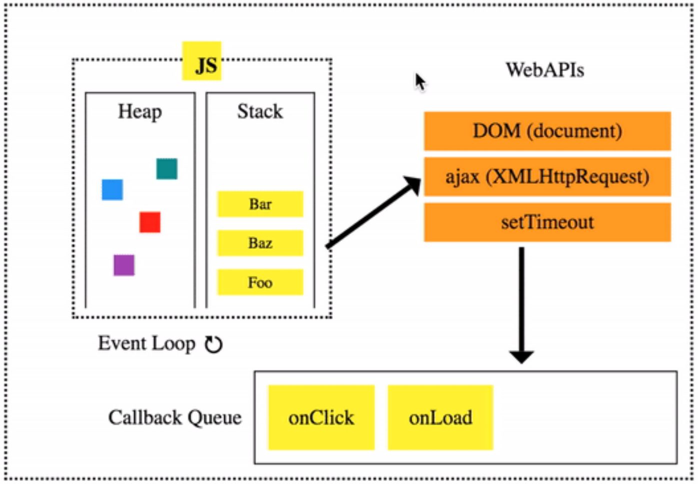
Async programming (on single thread)

JavaScript is a single-threaded language!

```
async function doWork()
{
    console.log("frist");
    await wait(1000);
    console.log("second");
}

doWork();
```

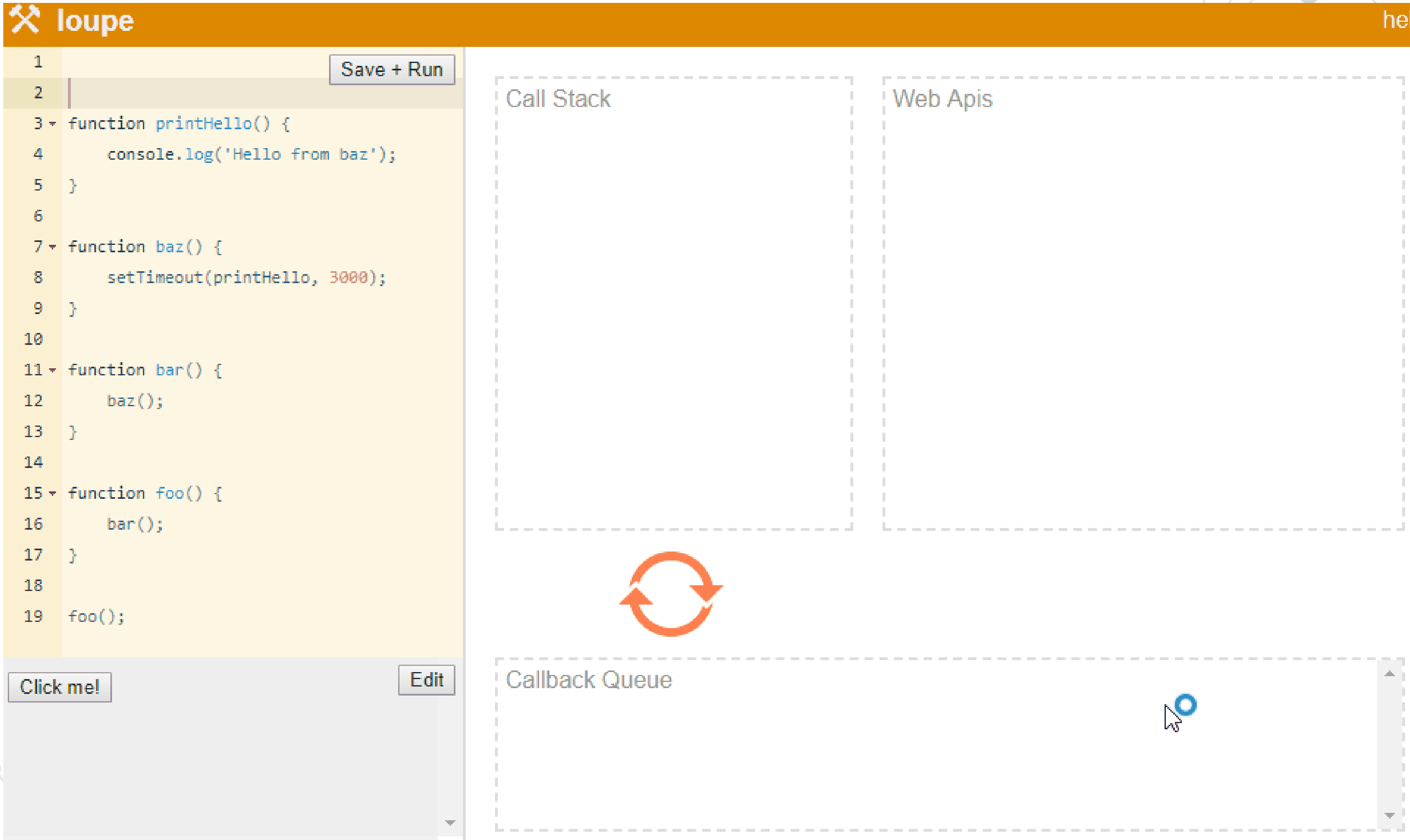
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Event Loop



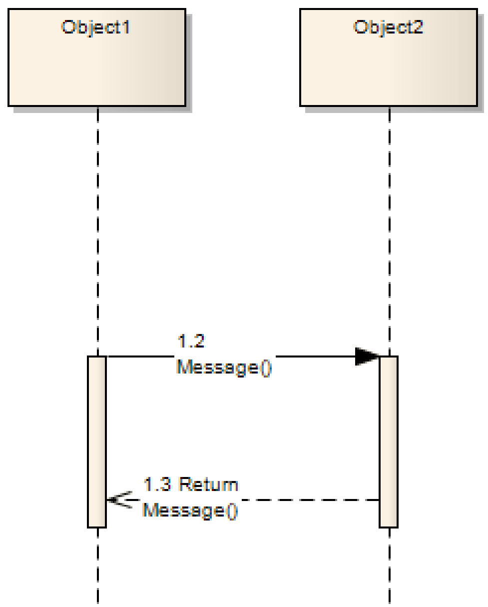
https://www.youtube.com/watch?v=8aGhZQkoFbQ

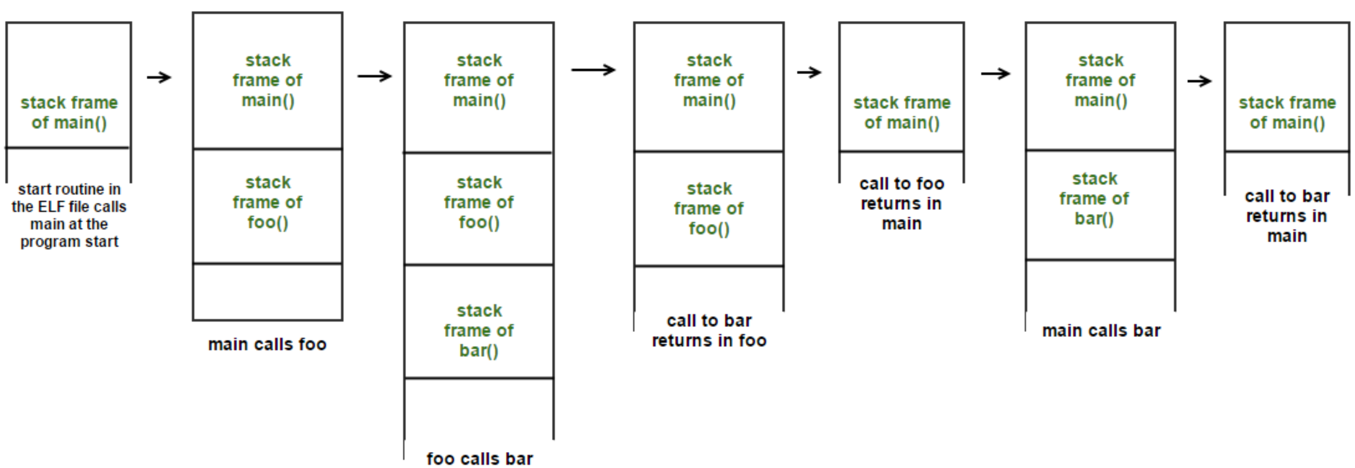
Javascript – Callback and Promise



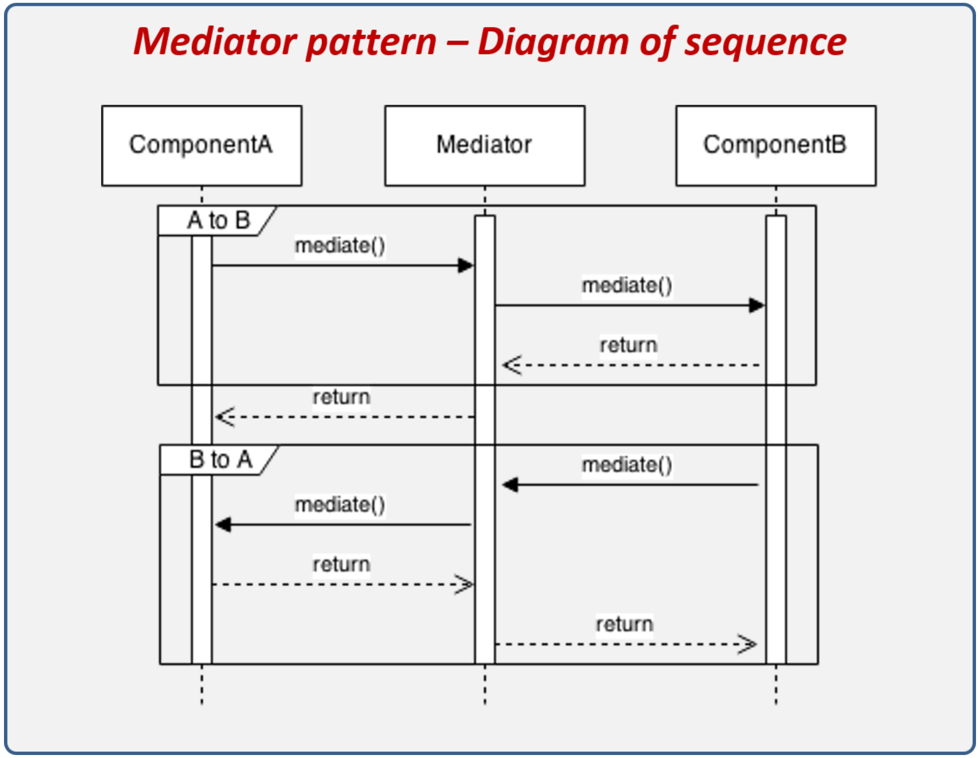
https://latentflip.com/loupe/

In-process / sync





In-process / sync with mediator pattern



Objects no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing coupling.

https://en.wikipedia.org/wiki/Mediator_pattern

In-process / sync with mediator pattern

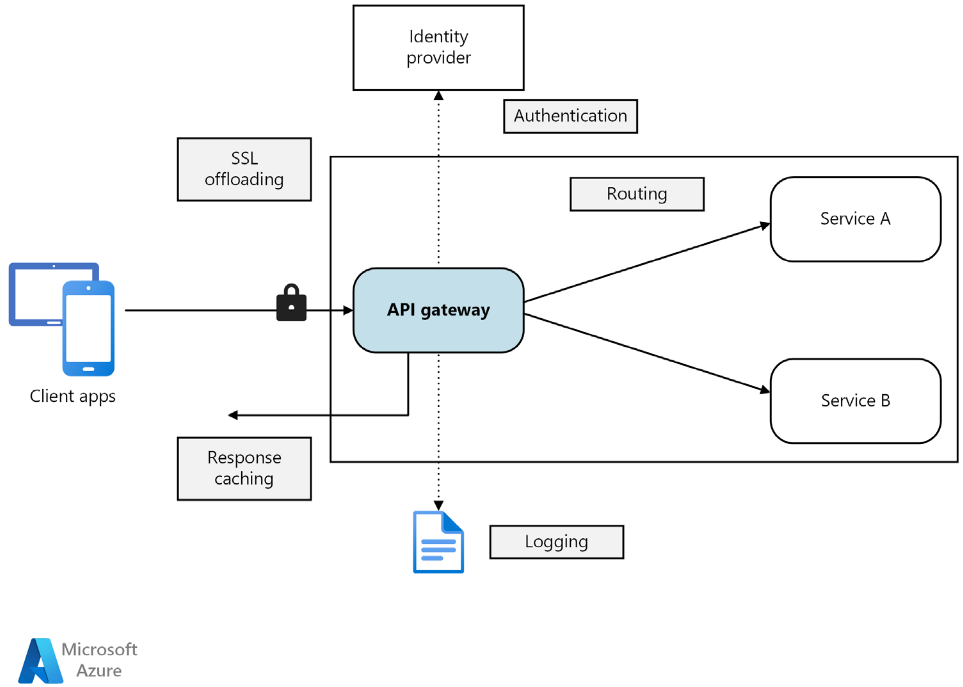
```
namespace Events.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly IPublisher _publisher;

        public OrderController(IPublisher publisher)
        {
            _publisher = publisher;
        }

        [HttpGet]
        public async Task NewOrder()
        {
            var @event = new NewOrderEvent();
            await _publisher.Publish(@event);
        }
    }
}
```

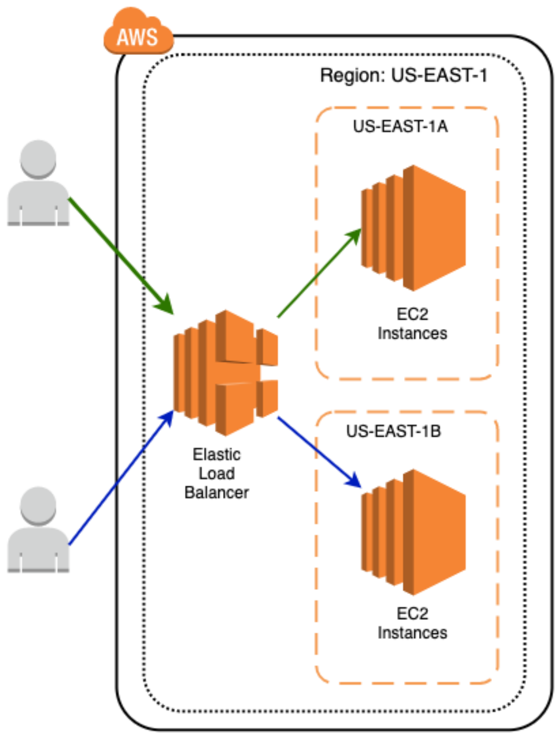
Example project: 03 EventsInProcessByMediator

In-process / sync with mediator pattern

Performance
Scalability
Resilience
Redundancy

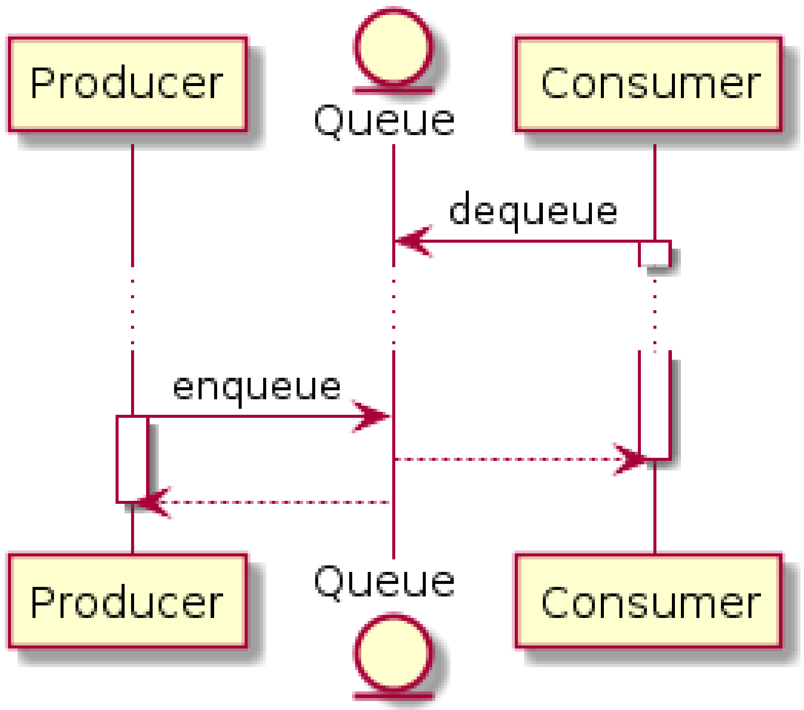
?



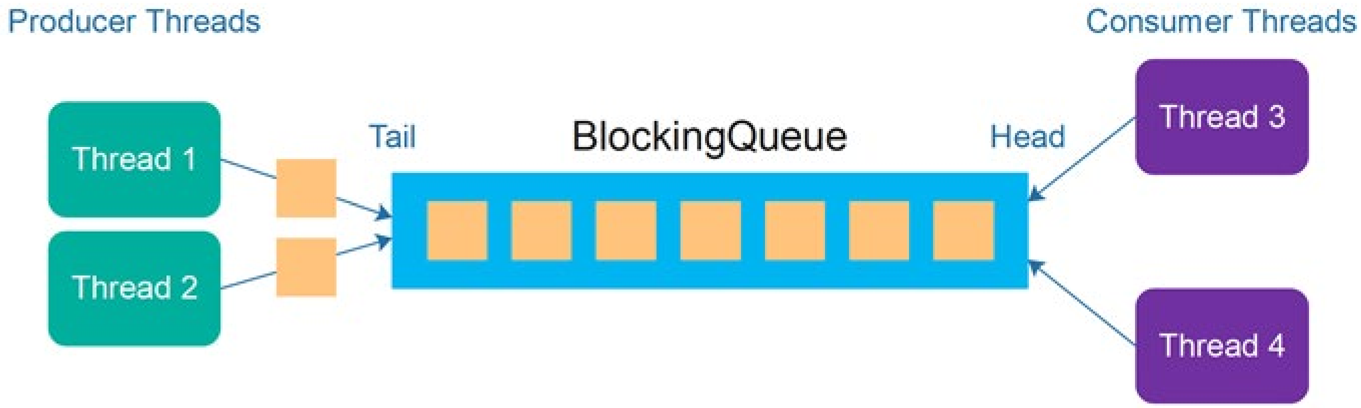


Continue to book «Designing Data-Intensive Applications» page 13

Out of process / async



Out of process / async with producer/consumer



Queue Producer

```
namespace EventsOutOfProcessByChannel.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly ChannelWriter<NewOrderEvent> _channelWriter;

        public OrderController(ChannelWriter<NewOrderEvent> channelWriter)
        {
            _channelWriter = channelWriter;
        }

        [HttpGet]
        public async Task NewOrder()
        {
            // Produce a new event and sent to channel
            var @event = new NewOrderEvent();
            await _channelWriter.WriteAsync(@event);
        }
    }
}
```

C# Channels are an implementation of the producer/consumer programming model.

https://learn.microsoft.com/en-us/dotnet/core/extensions/channels

Example project: 04 EventsOutOfProcessByChannel

Queue Consumer

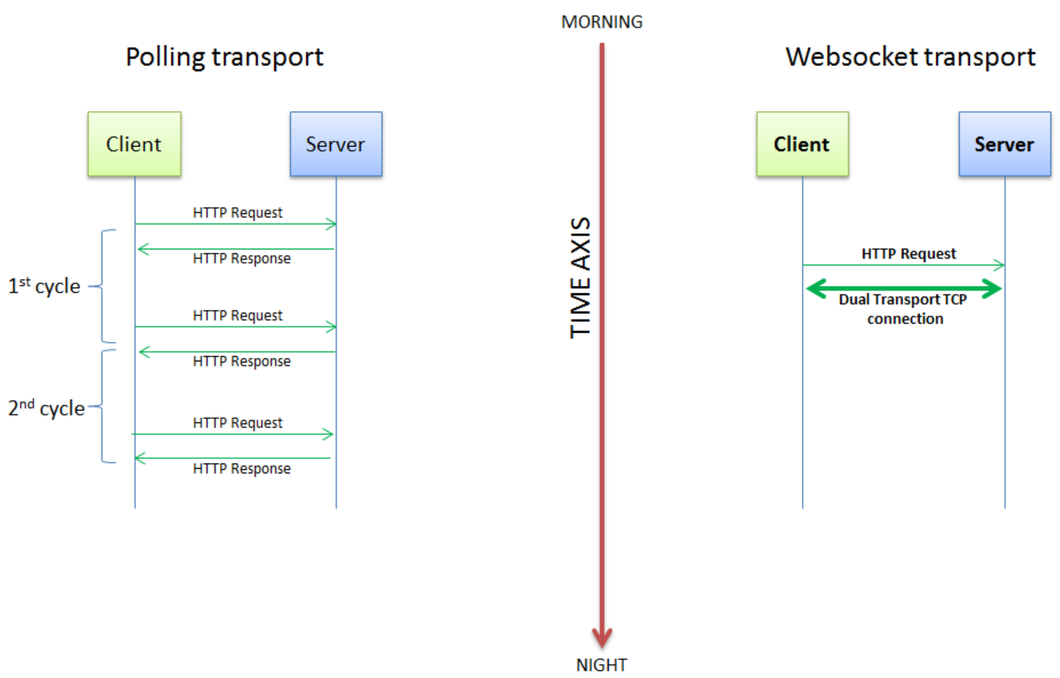
```
namespace EventsOutOfProcessByChannel
{
    public class Consumer
    {
        public static async ValueTask ConsumeWithWhileAsync(ChannelReader<NewOrderEvent> reader)
        {
            while (true)
            {
                var @event = await reader.ReadAsync();
                // Simulate some work
                Console.WriteLine($"Event elaborating {@event.Created}");
                Thread.Sleep(5000);
                Console.WriteLine($"Event comsumed {@event.Created}");
            }
        }
    }
}
```

C# Channels are an implementation of the producer/consumer conceptual programming model.

https://learn.microsoft.com/en-us/dotnet/core/extensions/channels

Example project: 04 EventsOutOfProcessByChannel

Queue Consumer – user feedback – polling vs websocket

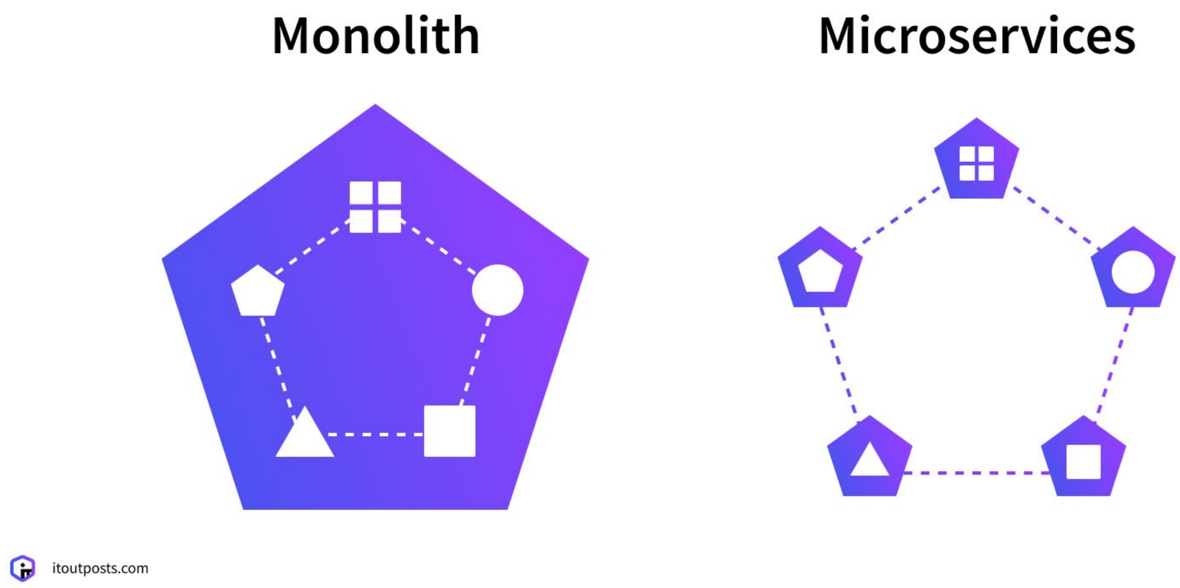


ASSERT (Night >= Morning)

Up-Scaling?
Down-Scaling?
Failure and reconnection from clients?

https://mashhurs.wordpress.com/2016/09/30/polling-vs-websocket-transport/

https://dev.to/kevburnsjr/websockets-vs-long-polling-3a0o



In a monolithic application running on a single process, components invoke one another using language-level method or function calls.

A microservices-based application is a distributed system running on multiple processes or services, usually even across multiple servers or hosts

https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

Out of-process / sync with microservice

```
namespace MicroserviceA.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly HttpClient _client;

        public OrderController(HttpClient client)
        {
            _client = client;
        }

        [HttpGet]
        public async Task<long> NewOrder()
        {
            Console.WriteLine("Sending request to MicroserviceB");
            var paymentResult = await _client.GetFromJsonAsync<long>("https://localhost:7165/payment");
            Console.WriteLine($"Sent request MicroserviceB with result {paymentResult}");

            …
        }
    }
}
```

Example project: 05 MicroserviceA/B

Out of-process / sync with microservice

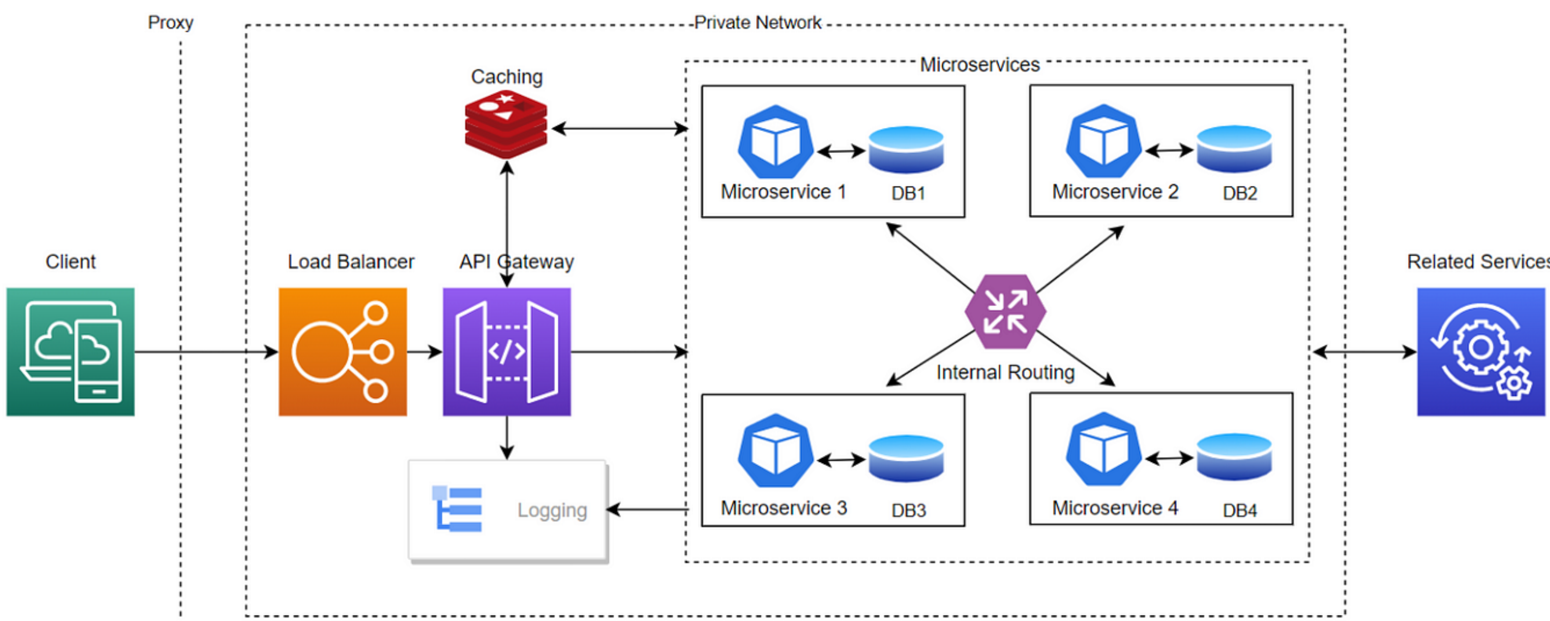
```
namespace MicroserviceB.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PaymentController : ControllerBase
    {
        [HttpGet]
        public long Get()
        {
            Console.WriteLine("Elaborating request");
            var result = Random.Shared.Next(0, 100);
            Thread.Sleep(1000);
            Console.WriteLine($"Elaborated request with result: {result}");
            return result;
        }
    }
}
```

Example project: 05 MicroserviceA/B

Out of-process / sync with microservice

Performance
Scalability
Resilience
Redundancy

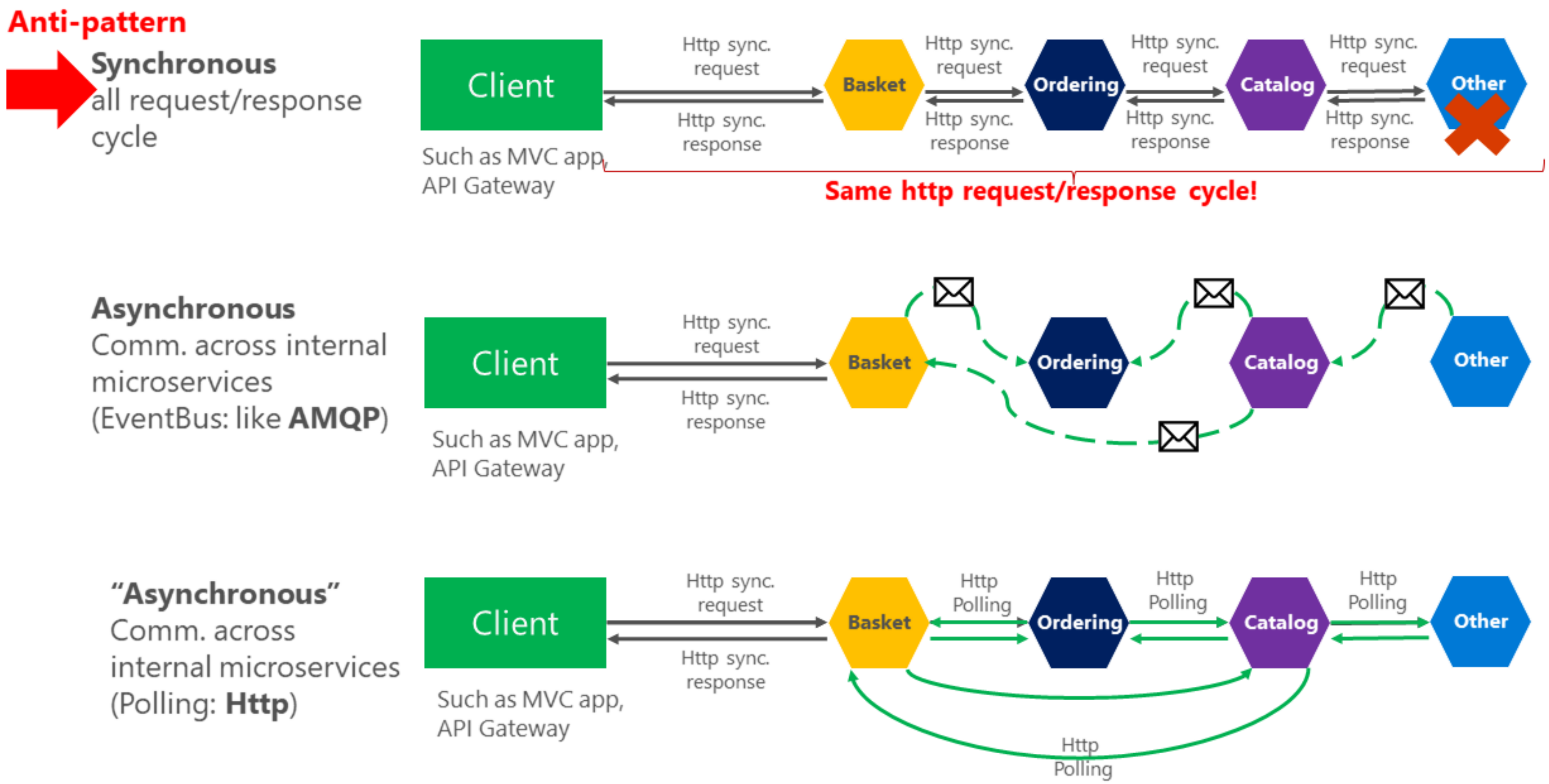
?



https://medium.com/@beuttam/building-scalable-microservices-with-proxy-load-balancer-api-gateway-private-network-services-f25c73cc8e02

Communication types

Synchronous vs. async communication across microservices



Out of-process / async with microservice - producer

```
namespace EventsOutOfProcessByDB.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly EventBusContext _eventBusContext;

        public OrderController(EventBusContext eventBusContext)
        {
            _eventBusContext = eventBusContext;
        }

        [HttpGet]
        public async Task NewOrder()
        {
            // Produce a new event and sent to channel
            var @event = new NewOrderEvent();
            @event.UserEmail = "diego@bonura.dev";

            var content = JsonSerializer.Serialize(@event, @event.GetType());
            var typeName = @event.GetType().FullName!;

            var message = new Message()
            {
                Type = typeName,
                Content = content
            };

            _eventBusContext.Add(message);
            await _eventBusContext.SaveChangesAsync();
        }
    }
}
```

Example project: 06 EventsOutOfProcessByDatabaseConsumer

Out of-process / async with microservice - consumer

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    while (true)
    {
        var messageToElaborate = _eventBusContext.Set<Message>().Where(m => m.ProcessedOn == null).OrderBy(m
=> m.OccurredOn).FirstOrDefault();
        if (messageToElaborate != null)
        {
            var type = AppDomain.CurrentDomain.GetAssemblies().Where(a => !a.IsDynamic).SelectMany(a =>
a.GetTypes()).FirstOrDefault(t => t.FullName == messageToElaborate.Type);
            var domainEvent = (INotification)JsonSerializer.Deserialize(messageToElaborate.Content, type);

            await _publisher.Publish(domainEvent);

            messageToElaborate.ProcessedOn = DateTime.Now;
            await _eventBusContext.SaveChangesAsync();
        }

        await Task.Delay(1000);
    }
}
```

Example project: 06 EventsOutOfProcessByDatabaseConsumer

Out of-process / async with microservice consumer

Performance
Scalability
Resilience
Redundancy

?

Is it easy to add new consumers to increase performance?

we need to introduce a row lock (on db side) or optimistic concurrency control (occ)

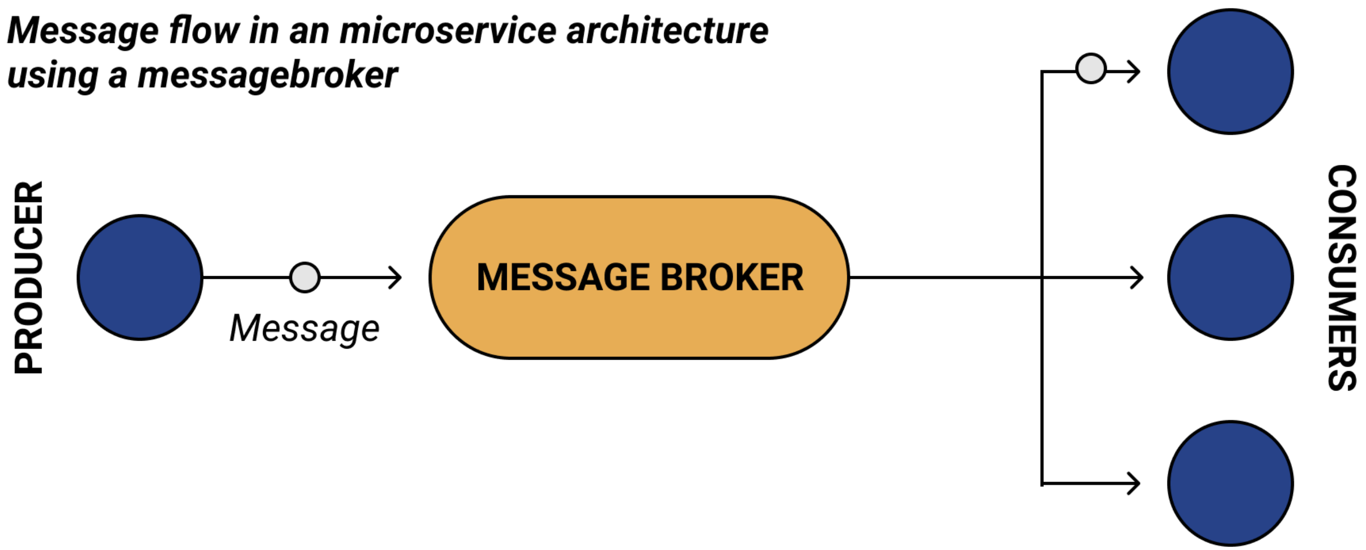
https://medium.com/@beuttam/building-scalable-microservices-with-proxy-load-balancer-api-gateway-private-network-services-f25c73cc8e02

Message broker

an intermediary for messaging

Message broker

Message flow in an microservice architecture using a messagebroker



Message broker

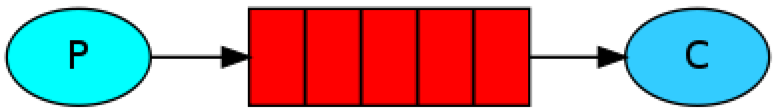
Message brokers

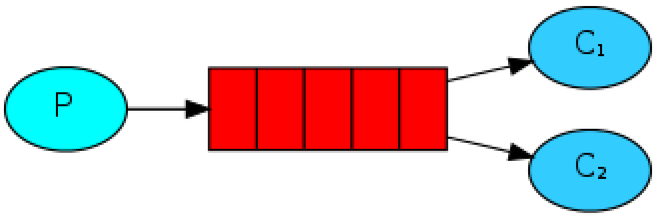
- can validate, store, route, and deliver messages to the appropriate destinations.
- act as intermediaries between other applications, allowing senders to issue messages without knowing where the recipients are located, whether or not they are active, or how many there are.
- simplifies the separation of processes and services within systems.

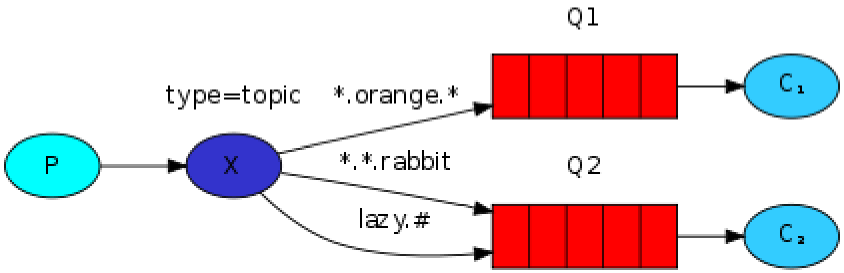
Protocols

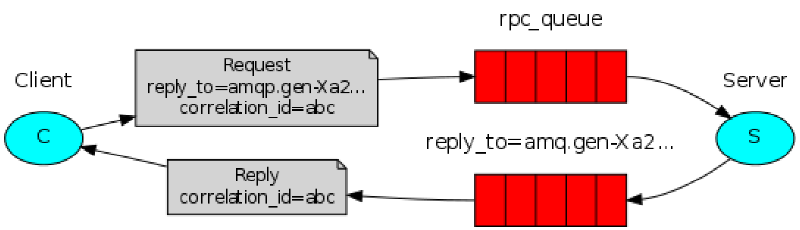
- AMQP: The Advanced Message Queuing Protocol (RabbitMQ/ Azure Service Bus / Amazon MQ / Apache ActiveMQ)
- Kafka: binary protocol over TCP
- MQTT: Lightweight and Efficient for IoT Messages (Mosquitto)

RabbitMQ

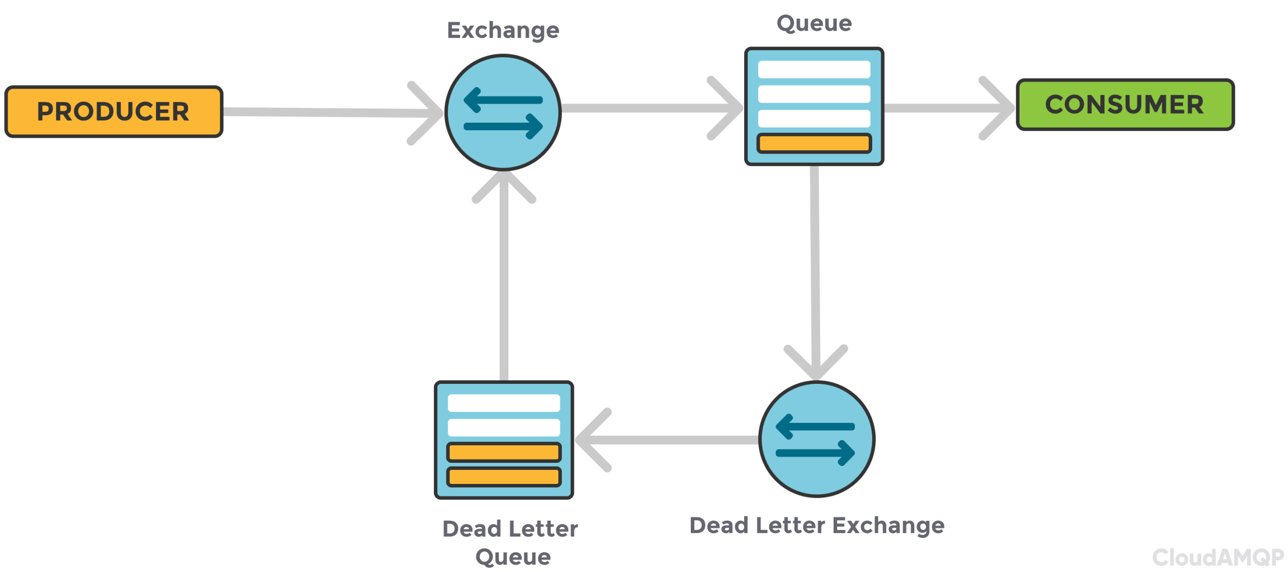

P
C


P
C_1
C_2


Q1
type=topic
.orange.
P
X
..rabbit
Q2
lazy.#
C_1
C_2


rpc_queue
Client
Request
reply_to=amqp.gen-Xa2...
correlation_id=abc
Server
C
S
reply_to=amq.gen-Xa2...
Reply
correlation_id=abc

RabbitMQ



Exchange
Queue
PRODUCER
CONSUMER
Dead Letter Queue
Dead Letter Exchange
CloudAMQP

RabbitMQ - Producer

```
public class EventBusRabbitMQ : IEventBus
{
    public void Publish(IEvent @event)
    {
        var factory = new ConnectionFactory { HostName = "localhost" };
        using var connection = factory.CreateConnection();
        using var channel = connection.CreateModel();

        channel.QueueDeclare(queue: "task_queue",
                             durable: true,
                             exclusive: false,
                             autoDelete: false,
                             arguments: null);

        string message = JsonSerializer.Serialize(@event, typeof(NewOrderEvent));
        var body = Encoding.UTF8.GetBytes(message);

        var properties = channel.CreateBasicProperties();
        properties.Persistent = true;

        channel.BasicPublish(exchange: string.Empty,
                             routingKey: "task_queue",
                             basicProperties: properties,
                             body: body);

    }
}
```

RabbitMQ - Consumer

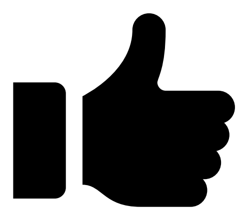
```
var factory = new ConnectionFactory { HostName = "localhost" };
using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

channel.QueueDeclare(queue: "task_queue",
                        durable: true,
                        exclusive: false,
                        autoDelete: false,
                        arguments: null);

channel.BasicQos(prefetchSize: 0, prefetchCount: 1, global: false);
var messageConsumer = new EventingBasicConsumer(channel);

messageConsumer.Received += async (model, ea) =>
{
    byte[] body = ea.Body.ToArray();
    var @event = (NewOrderEvent)JsonSerializer.Deserialize(body, typeof(NewOrderEvent));
    Console.WriteLine($"Received from {@event.UserEmail}");

    await Task.Delay(100);

    channel.BasicAck(deliveryTag: ea.DeliveryTag, multiple: false);
};

channel.BasicConsume(queue: "task_queue",
                     autoAck: false,
                     consumer: messageConsumer);

Console.ReadLine();
```

Distribute application with message broker

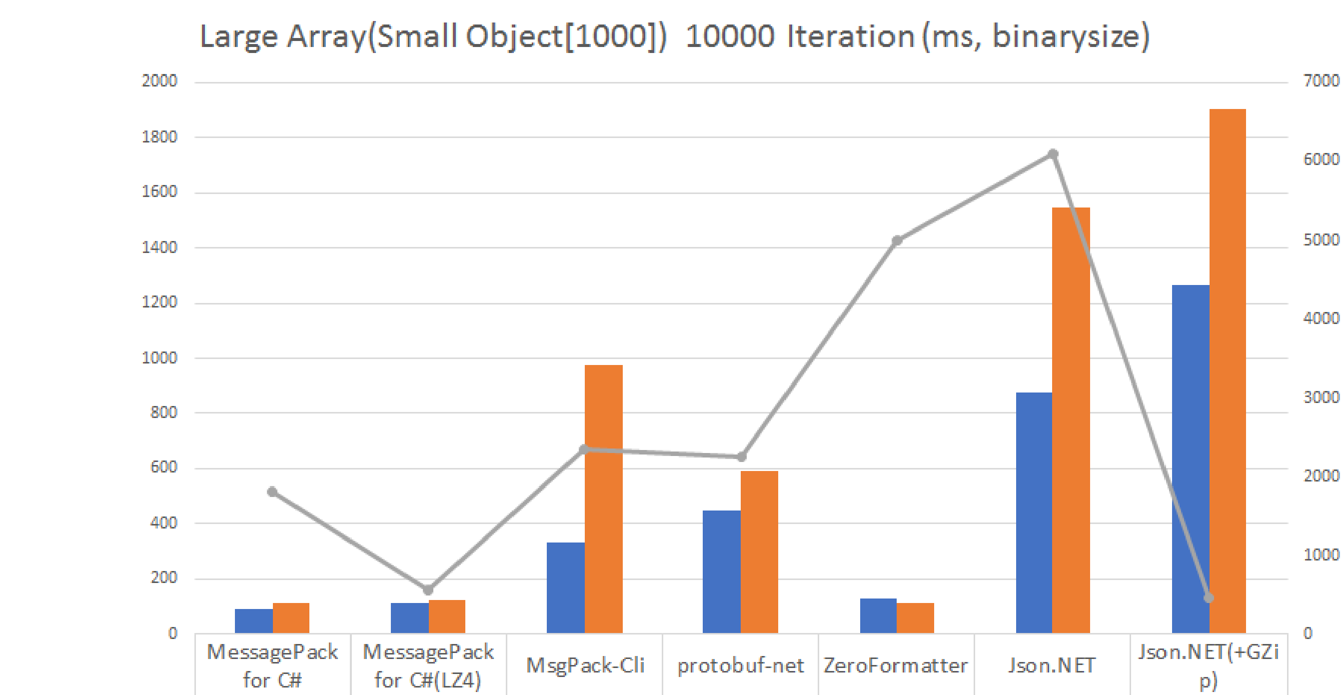
Performance
Scalability
Resilience
Redundancy

?

Is it easy to add new consumers to increase performance?



Serialization performance



	MessagePack for C#	MessagePack for C#(LZ4)	MsgPack-Cli	protobuf-net	ZeroFormatter	Json.NET	Json.NET(+GZip)
Serialize	90.5203	112.2836	333.7555	448.4978	127.6575	875.7939	1265.6311
Deserialize	114.9669	124.6713	976.4129	592.2848	115.8211	1546.8205	1902.2232
BinarySize	1803	562	2347	2248	5004	6096	458

https://github.com/neuecc/Utf8Json

https://github.com/MessagePack-CSharp/MessagePack-CSharp

Serialization performance

Json

Overview				Messages			Message rates		
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
task_queue	classic	D	running	1,835	0	1,835	36/s		

+/-

Add a new queue

Protobuf

Overview				Messages			Message rates		
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
task_queue	classic	D	running	237	0	237	52/s		

+/-

Add a new queue

Generate Ids on distributed application

We need to generate Id on the client before inserting a new row into the database:

Possibilities:

- GUID generated on client (too big – not sortable)
- Sql server – single table (Single point of failure – Not scalable)
- Specific services as *snowflake* and *zookeeper (Scalable but another service to mantain)*
- *Sequence on db and cache chunks*

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasSequence<int>("BlogIdSequence")
        .IncrementsBy(100); //10 is default

    modelBuilder.Entity<Blog>()
        .Property(b => b.Name)
        .IsUnicode(false)
        .HasMaxLength(20);

    modelBuilder.Entity<Blog>()
        .Property(b => b.BlogId)
        .UseHiLo("BlogIdSequence");
}
```

https://medium.com/@sandeep4.verma/system-design-distributed-global-unique-id-generation-d6a440cc8e5

https://medium.com/@jitenderkmr/demystifying-snowflake-ids-a-unique-identifier-in-distributed-computing-72796a827c9d

https://phanikumaryadavilli.medium.com/generating-distributed-uuids-using-zookeeper-a02cabfda0e9

Distributed application with a framework

Masstransit

Easily build reliable distributed applications

MassTransit provides a developer-focused, modern platform for creating distributed applications without complexity.

- First class testing support
- Write once, then deploy using RabbitMQ, Azure Service Bus, and Amazon SQS
- Observability via Open Telemetry (OTEL)
- Fully-supported, widely-adopted, a complete end-to-end solution

Masstransit - Producer

```
public class OrderController : ControllerBase
{
    private readonly IBus _bus;

    public OrderController(IBus bus)
    {
        _bus = bus;
    }

    [HttpGet]
    public async Task NewOrderAsync()
    {
        // Produce a new event and sent to channel
        var @event = new NewOrderEvent();
        @event.UserEmail = "diego@bonura.dev";

        await _bus.Publish(@event);
    }
}
```

Masstransit - Consumer

```
namespace DistributedAppWithMassTransitConsumer
{
    public class MessageConsumer : IConsumer<NewOrderEvent>
    {
        readonly ILogger<MessageConsumer> _logger;

        public MessageConsumer(ILogger<MessageConsumer> logger)
        {
            _logger = logger;
        }

        public Task Consume(ConsumeContext<NewOrderEvent> context)
        {
            _logger.LogInformation("Received ordine from: {email}", context.Message.UserEmail);

            return Task.CompletedTask;
        }
    }
}
```

Applications go wrong

The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair.

—Douglas Adams, *Mostly Harmless* (1992)

Applications go wrong

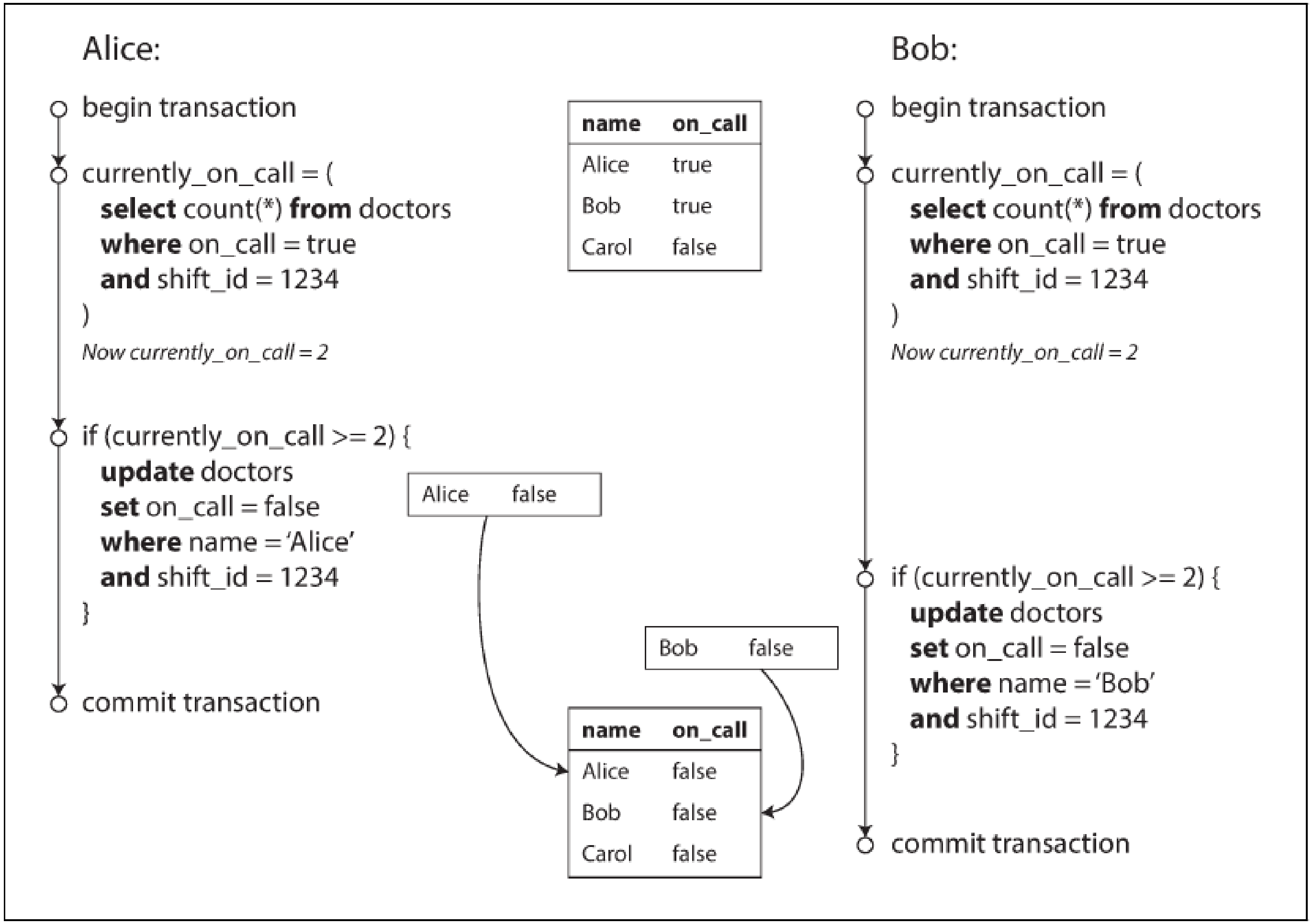
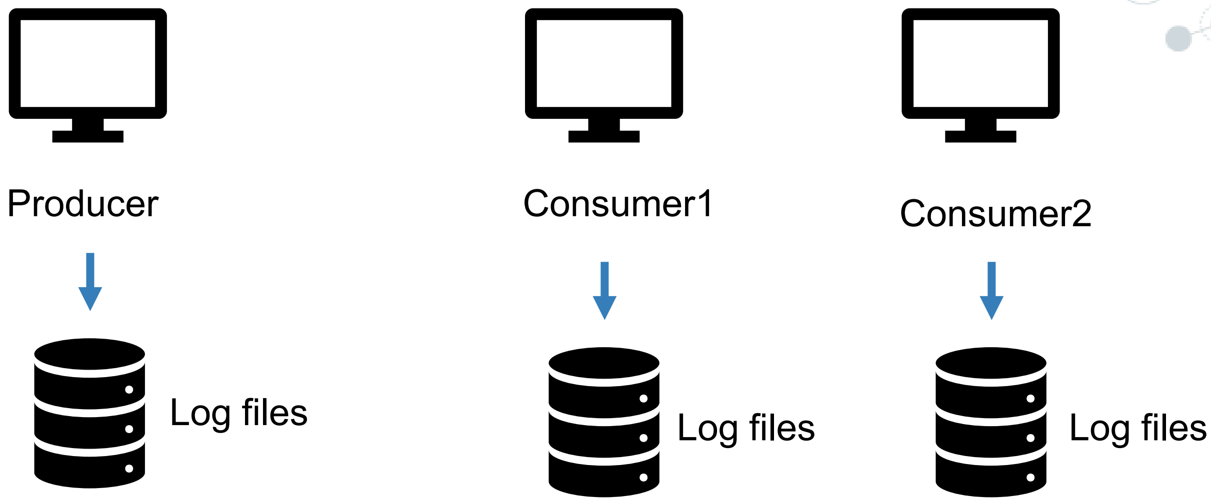


Figure 7-8. Example of write skew causing an application bug.

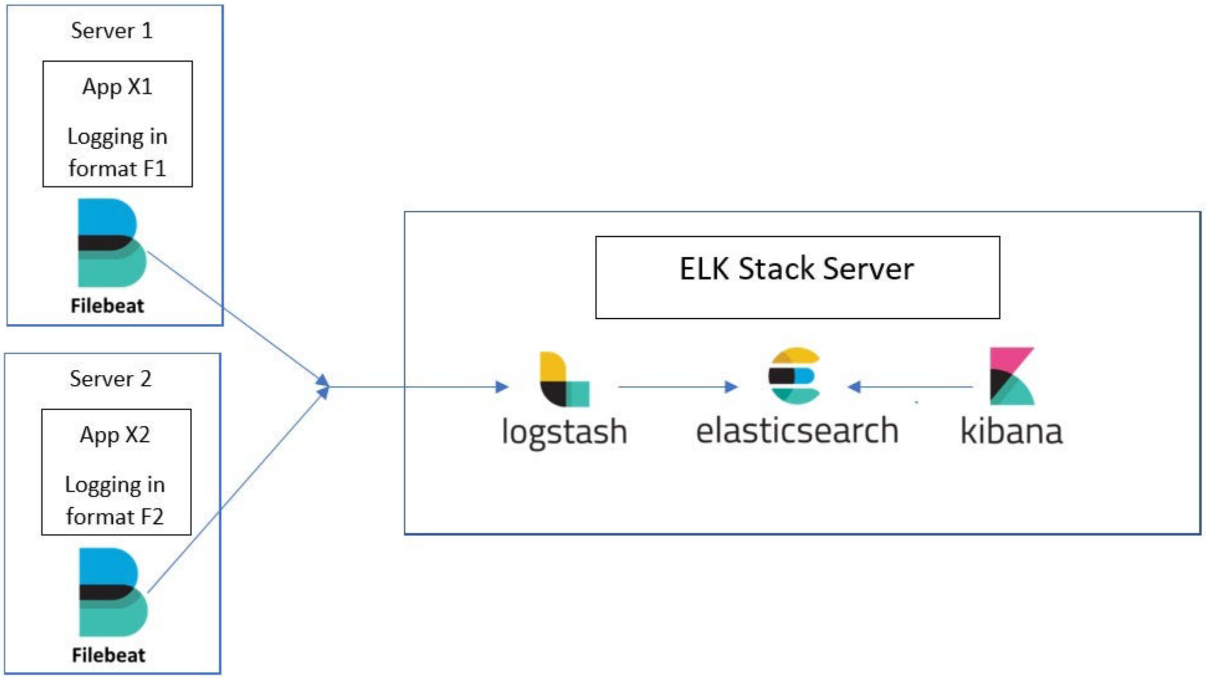
Page 246 of Design Data-Intensive Applications

Logging on distributed application

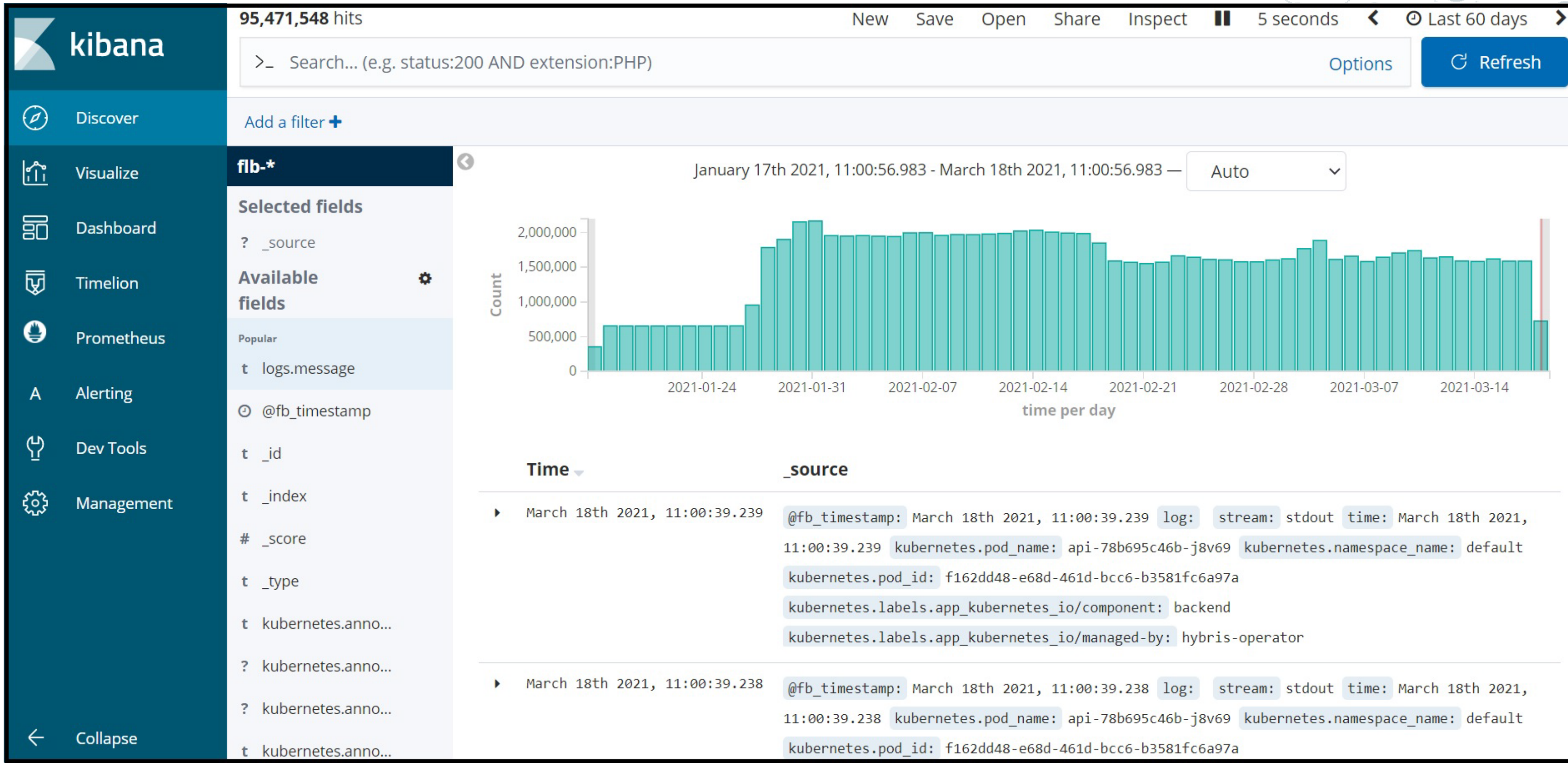


How to get information when things go wrong?

Callect logs in one place



Call logs in one place



ELK Stack

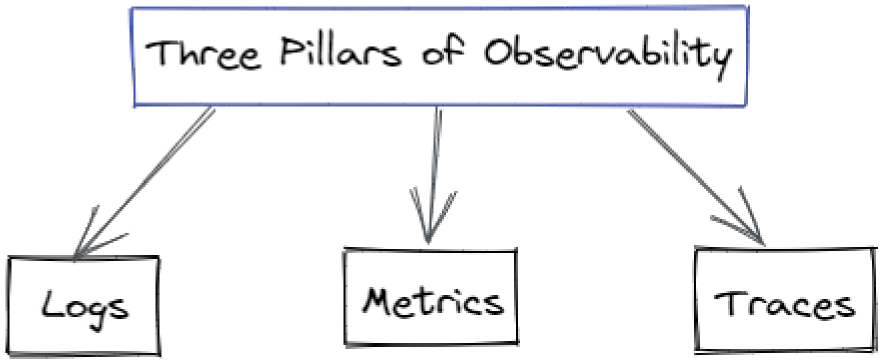
Seq

DATADOG

Observability

On distributed application logs monitoring could be difficult

Main concepts of observability



Logs in the technology and development field give a written record of happenings within a system, similar to the captain's log on a ship.

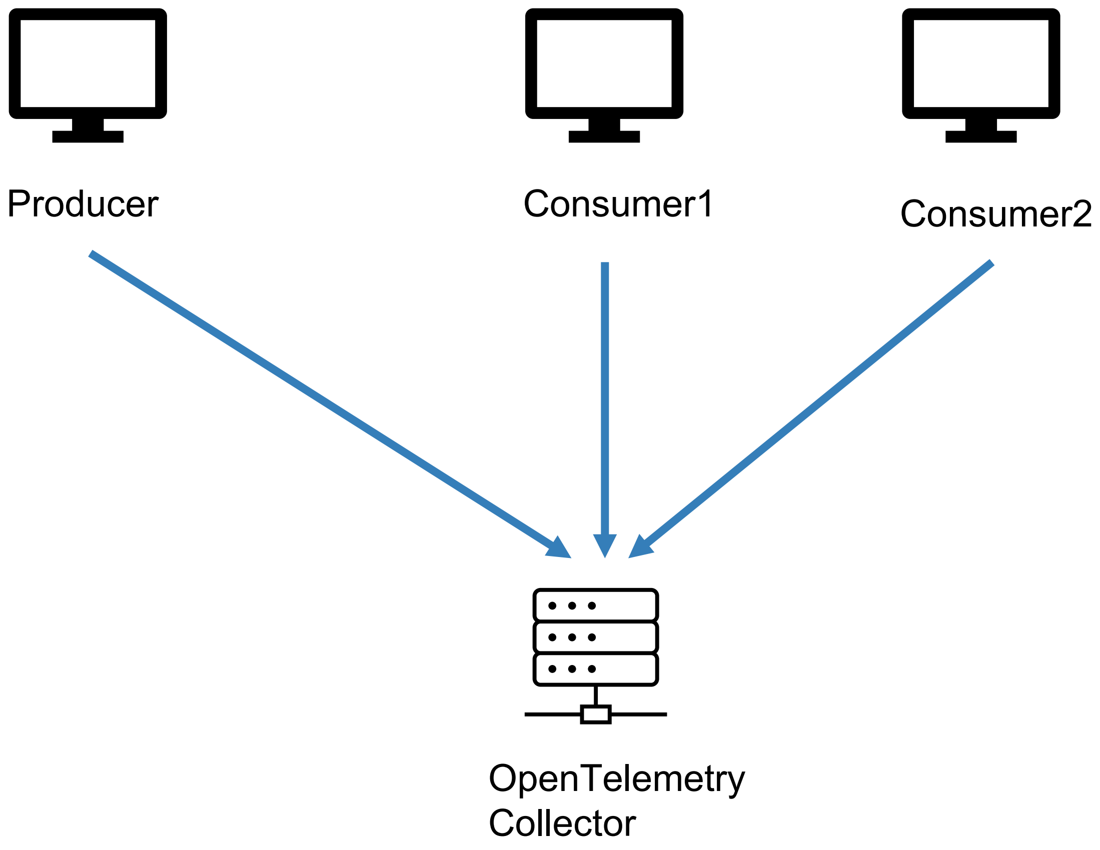
Metrics are a set of values that are tracked over time.

A **trace** is a means to track a user request from the user interface all the way through the system and back to the user when they receive confirmation that their request has been completed. As part of the trace, every operation executed in response to the request is recorded.

Observability standard

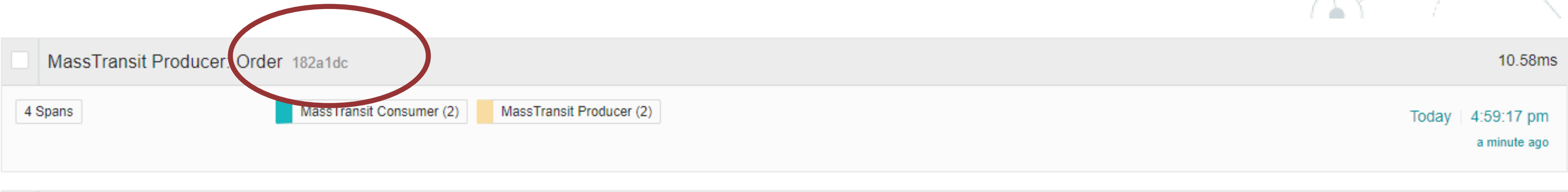
OpenTelemetry is an open-source CNCF (Cloud Native Computing Foundation) project formed from the merger of the OpenCensus and OpenTracing projects. It provides a collection of tools, APIs, and SDKs for capturing metrics, distributed traces and logs from applications.

OpenTelemetry on distributed application

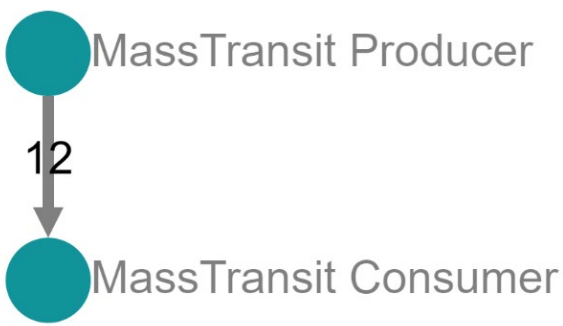


Example

Trace:



Metric:



Distributed lock

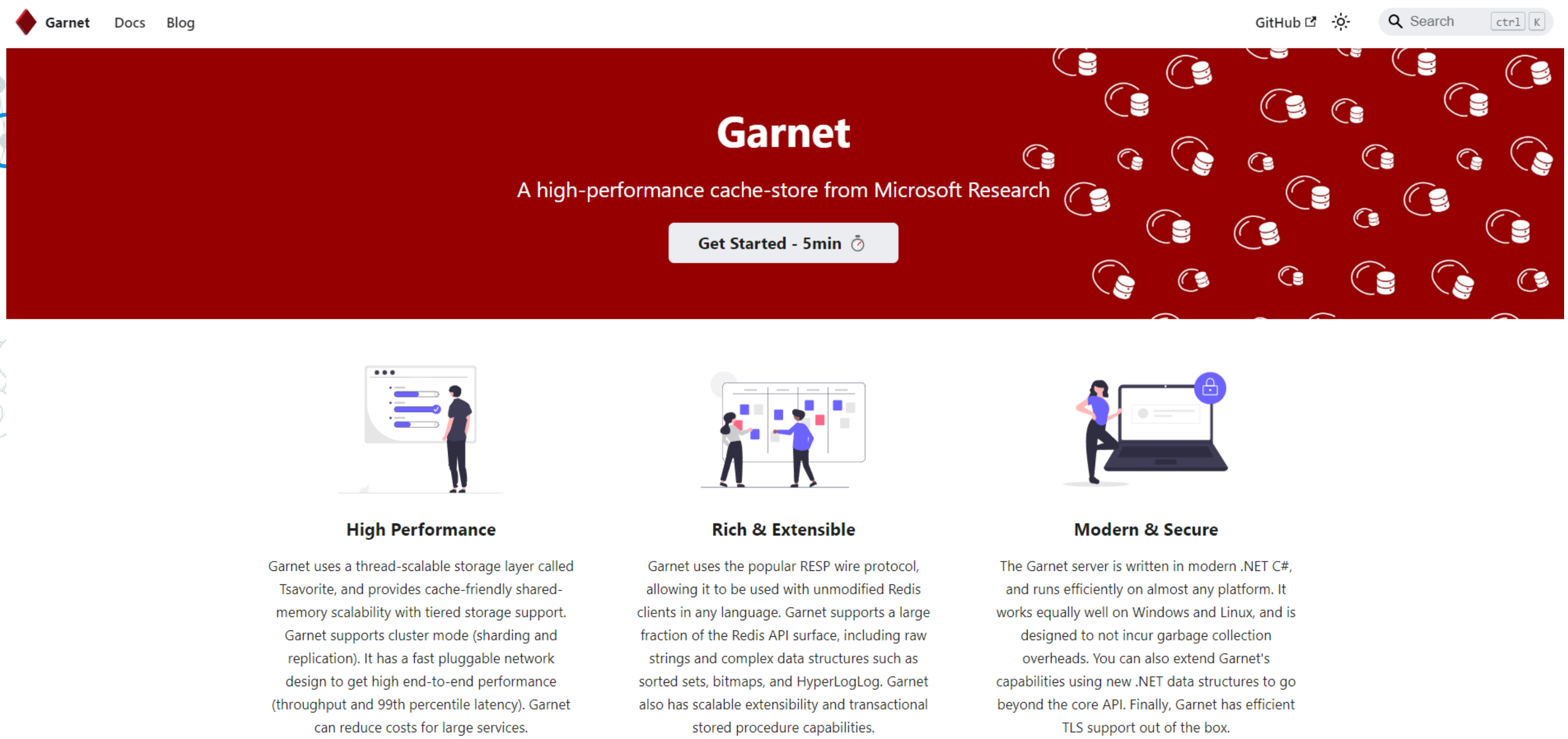
Distributed locks are a very useful primitive in many environments where different processes must operate with shared resources in a mutually exclusive way.

Redis

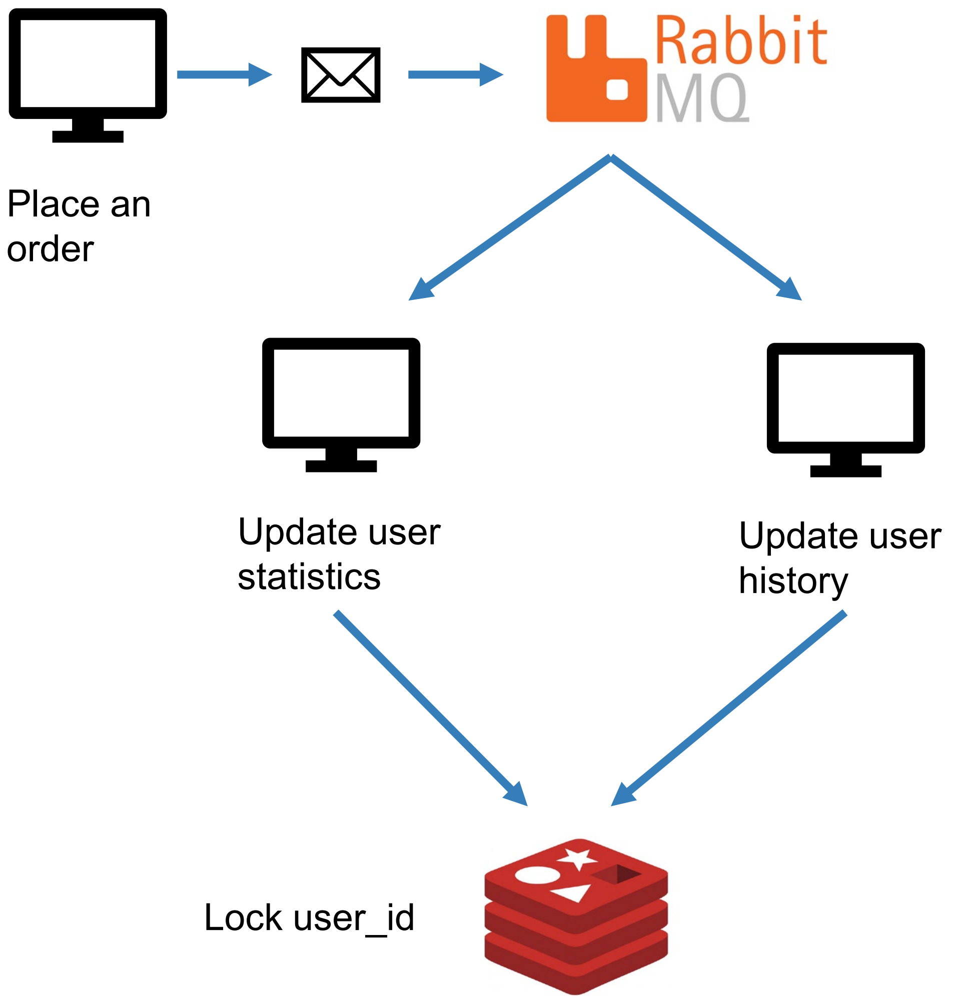
The open source, in-memory data store used by millions of developers as a database, cache, streaming engine, and message broker.

Created by: Salvatore Sanfilippo

https://redis.io/



https://microsoft.github.io/garnet/

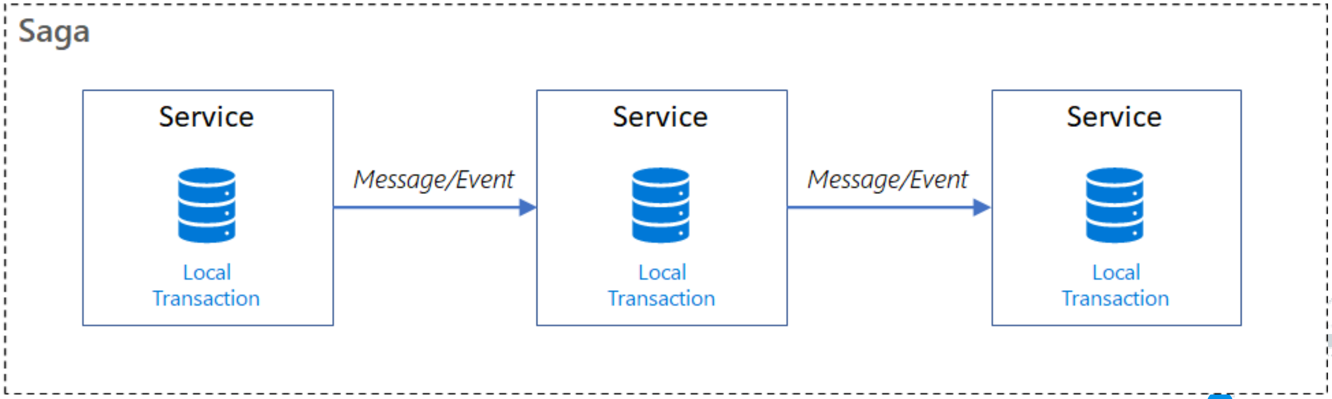

Rabbit MQ
Place an order
Update user statistics
Update user history
Lock user_id

Redis lock

```
static async Task Main(string[] args)
{
    var endPoints = new List<RedLockEndPoint> { new DnsEndPoint("localhost", 6379) };
    var redlockFactory = RedLockFactory.Create(endPoints);

    var resource = "my-order-id";
    var expiry = TimeSpan.FromSeconds(30);

    await using (var redLock = await redlockFactory.CreateLockAsync(resource, expiry))
    {
        // make sure we got the lock
        if (redLock.IsAcquired)
        {
            // do stuff
        }
    }
}
```

Saga

When you have to orchestrate events!



Saga: consistency models

Immediate consistency: once a write operation (e.g., updating a piece of data) is completed, any subsequent read operation (e.g., retrieving that data) will reflect the updated value.

- expensive in terms of performance
- not ideal in all distributed systems

ACID (atomicity, consistency, isolation, durability).

Eventual consistency: may be a period of time during which different nodes or replicas in the system have different versions of the data.

- commonly used in systems like NoSQL databases

BASE (basically-available, soft-state, eventual consistency)

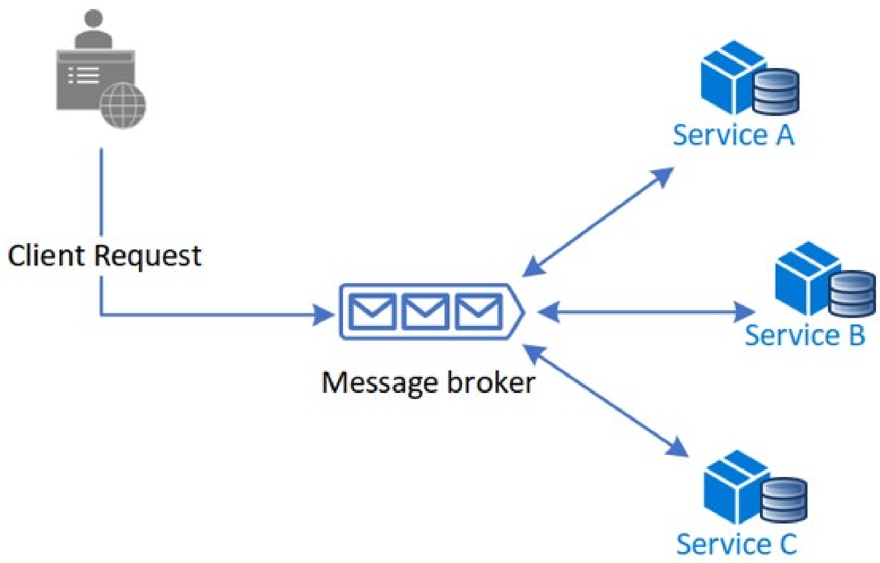
Saga: trade off

https://priyalwalpita.medium.com/steering-clear-of-distributed-monolith-traps-in-your-journey-to-effective-microservices-86671be0b604

https://www.youtube.com/watch?v=p2GlRToY5HI

Saga approaches: choreography and orchestration

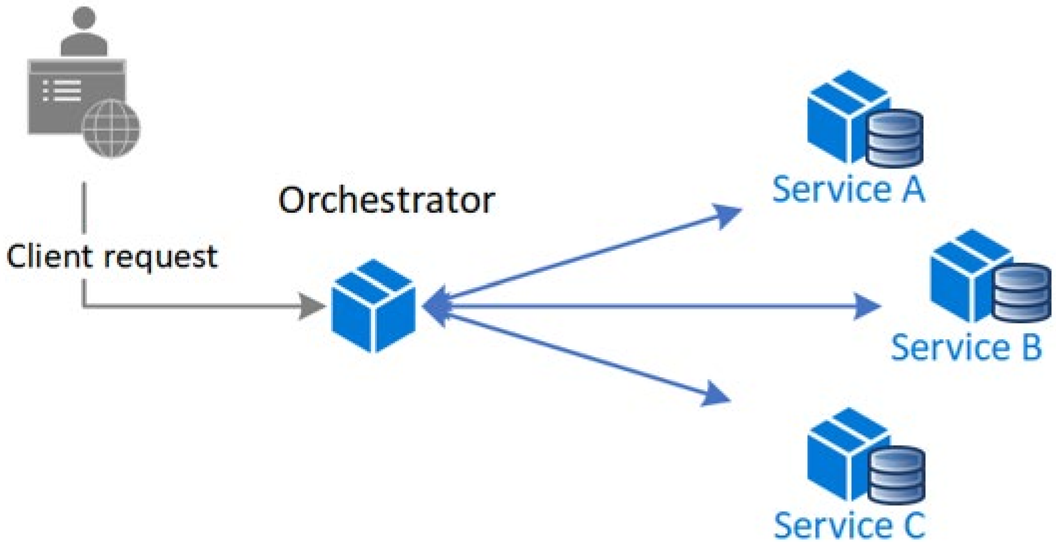
Choreography: without a centralized point of control



https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga

Saga approaches: choreography and orchestration

Orchestration: centralized controller tells participants what to execute



https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga

Saga with MassTransit

```
public OrderStateMachine()
{
    InstanceState(x => x.CurrentState);

    Event(() => NewOrderEvent, x => x.CorrelateById(context => context.Message.OrderId));
    Event(() => OrderProcessed, x => x.CorrelateById(context => context.Message.OrderId));
    Event(() => OrderCancelled, x => x.CorrelateById(context => context.Message.OrderId));

    Initially(
        When(NewOrderEvent)
            .Then(context =>
            {
                context.Saga.ProcessingId = Guid.NewGuid();
            })
            .Publish(context => new ProcessOrder(context.Saga.CorrelationId))
            .TransitionTo(Pending)
            .Then(context => Console.Out.WriteLineAsync($"From New to Pending: {context.Saga.CorrelationId}"))
    );

    During(Pending,
        When(OrderProcessed)
            .TransitionTo(Accepted)
            .Then(context => Console.Out.WriteLineAsync($"From Pending to Accepted: {context.Saga.CorrelationId}"))
            .Finalize(),
        When(OrderCancelled)
            .TransitionTo(Cancelled)
            .Then(context => Console.Out.WriteLineAsync($"From Pending to Faulted: {context.Saga.CorrelationId} for reason:
{context.Message.Reason}"))
            .Finalize()
        );

    SetCompletedWhenFinalized();
}
```

Saga choreography

MassTransit elaborates saga and creates few queue and exchanges on RabbitMq

Exchanges

All exchanges (13)

Pagination

Page 1 of 1 - Filter: Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/	(AMQP default)	direct	D			
/	Message	fanout	D			
/	OrderState	fanout	D			
/	SagaWithMasstransitShared:NewOrderEvent	fanout	D	0.00/s	0.00/s	
/	SagaWithMasstransitShared:OrderCancelled	fanout	D	0.00/s	0.00/s	
/	SagaWithMasstransitShared:OrderProcessed	fanout	D	0.00/s	0.00/s	
/	SagaWithMasstransitShared:ProcessOrder	fanout	D	0.00/s	0.00/s	
/	amq.direct	direct	D			
/	amq.fanout	fanout	D			
/	amq.headers	headers	D			
/	amq.match	headers	D			
/	amq.rabbitmq.trace	topic	D I			
/	amq.topic	topic	D			

Actor model

Instead of calling methods, actors send messages to each other!

https://doc.akka.io/docs/akka/current/typed/guide/actors-intro.html

https://learn.microsoft.com/en-us/dotnet/orleans/overview

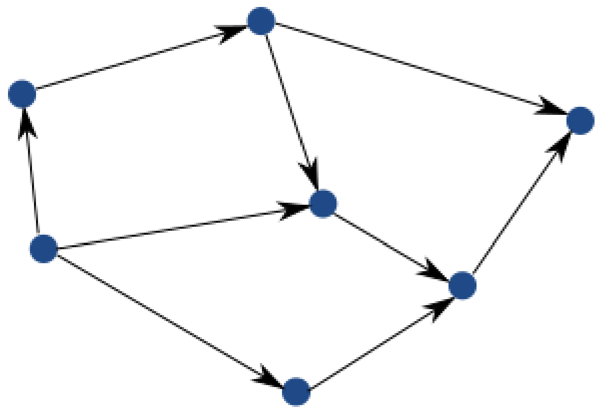
Actor model

The Actor Model: A Paradigm for Concurrent and Distributed Computing

The actor model is a programming model in which each actor is a lightweight, concurrent, immutable object that encapsulates a piece of state and corresponding behavior. Actors communicate exclusively with each other using asynchronous messages.

Actor model

When we have a Producer and Consumer we usually send message to a queue

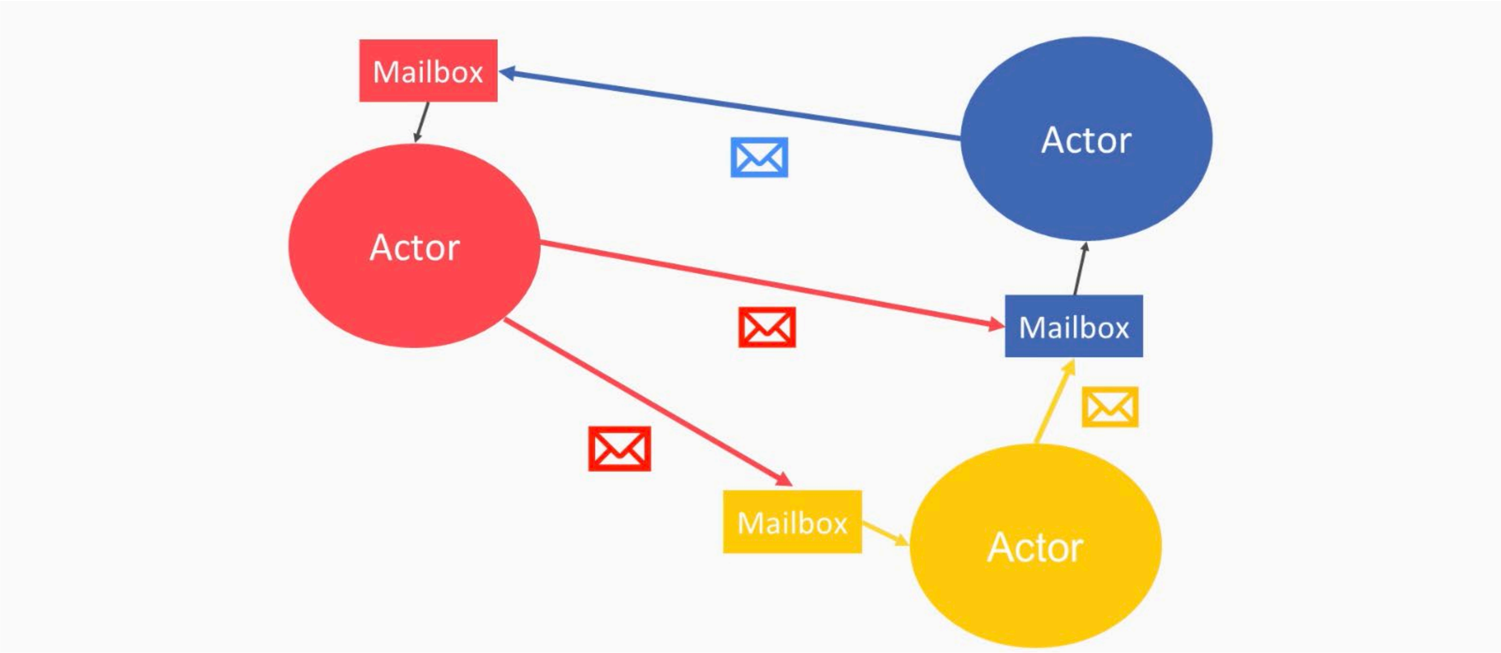


Actors interacting with each other by sending messages to each other

On actor model, we can implement Producer and Consumer as actor.

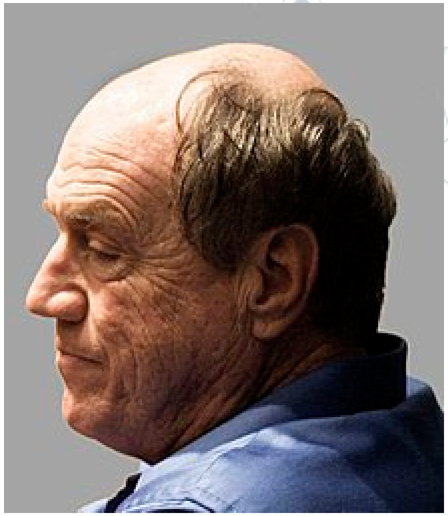
In Producer, we just get the actor reference of Consumer actor to send messages to Consumer's mailbox.

Actor model



Actor model: History 1973

The Actor Model is a mathematical theory of computation that treats "Actors" as the universal conceptual primitives of concurrent digital computation.



Carl Hewitt

The actor model was inspired by physics

Actors is based on "behavior" as opposed to the "class" concept of object-oriented programming.

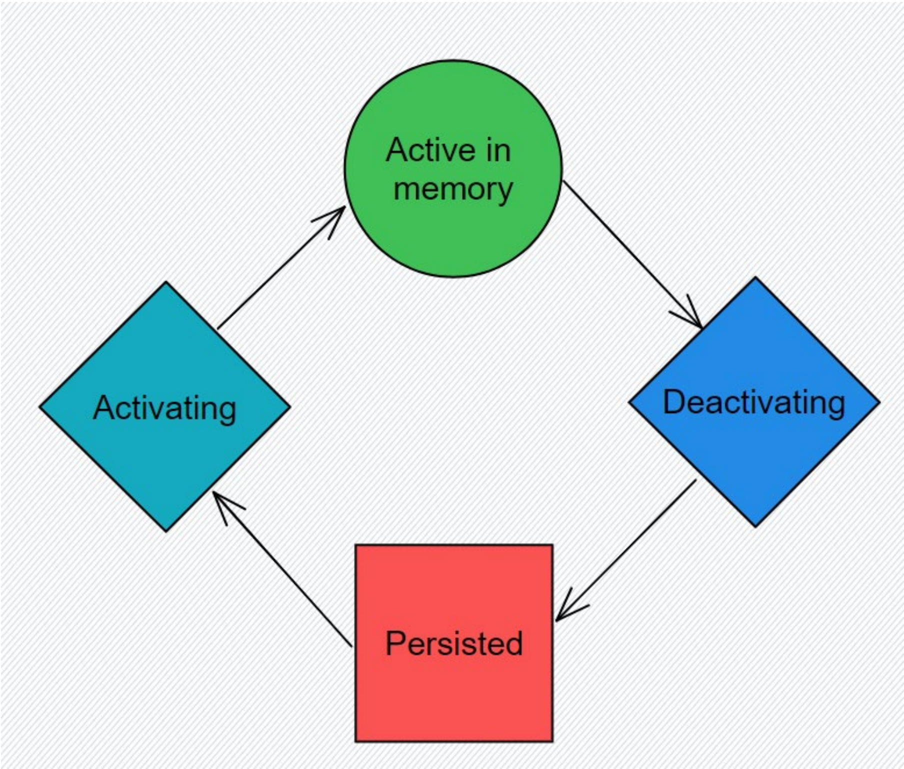
https://en.wikipedia.org/wiki/Actor_model

Actor model

Main principles:

1. **Isolation**: Actors are independent, with their own state and behavior.
2. **Single thread**: Actors process requests one at time
3. **Messaging**: Actors interact by exchanging asynchronous messages.
4. **Location Transparency**: Actors' locations are abstracted, enabling distribution.

Actor model: life cycle



Actor model: implementations

Java / c#

c#

https://akka.io/

https://getakka.net/

https://learn.microsoft.com/en-us/dotnet/orleans/overview

Actor model implementations on Orleans Microsoft research (2010)

https://www.microsoft.com/en-us/research/project/orleans-virtual-actors/

Orleans invented the Virtual Actor abstraction

Actors are purely logical entities that always exist, virtually. An actor cannot be explicitly created nor destroyed, and its virtual existence is unaffected by the failure of a server that executes it. Since actors always exist, they are always addressable.

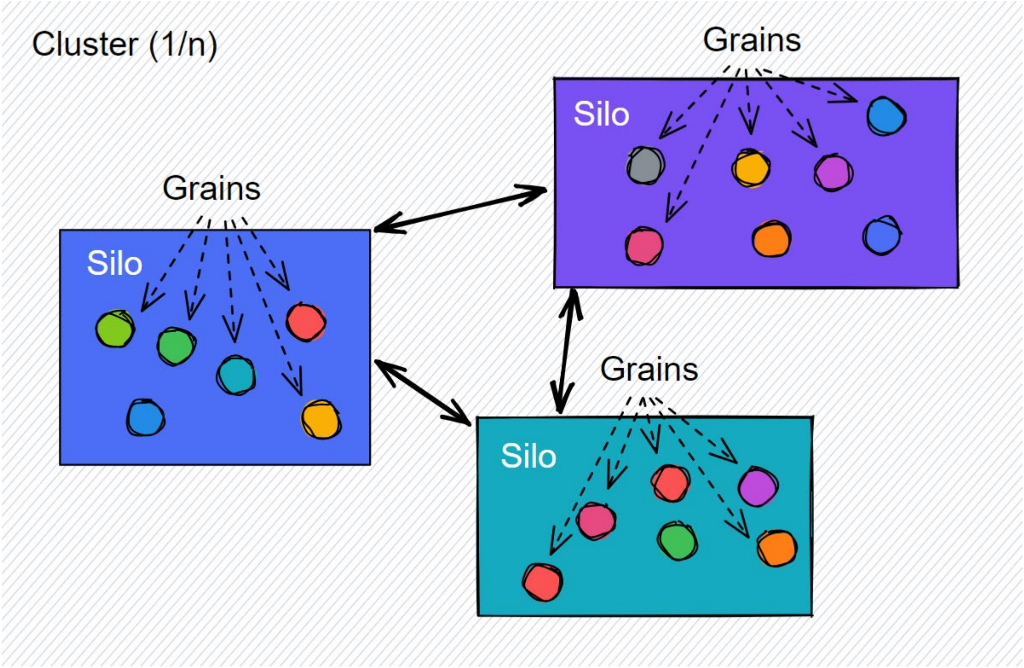
Actor model implementations on Orleans - Grain

1. **Grain**: grains are implementation of a virtual actor.
2. **Interfaces**: grains define interfaces.
3. **Grain:** has always an identity (string, number, guid)
4. **Persistence**: grains could volatile or persisted
5. **Lifecycle**: grains could be terminated to free computer resources

https://learn.microsoft.com/en-us/dotnet/orleans/overview#what-are-grains

Actor model implementations on Orleans - Silo

A silo hosts one or more grains



You can have any number of clusters, each cluster has one or more silos, and each silo has one or more grains

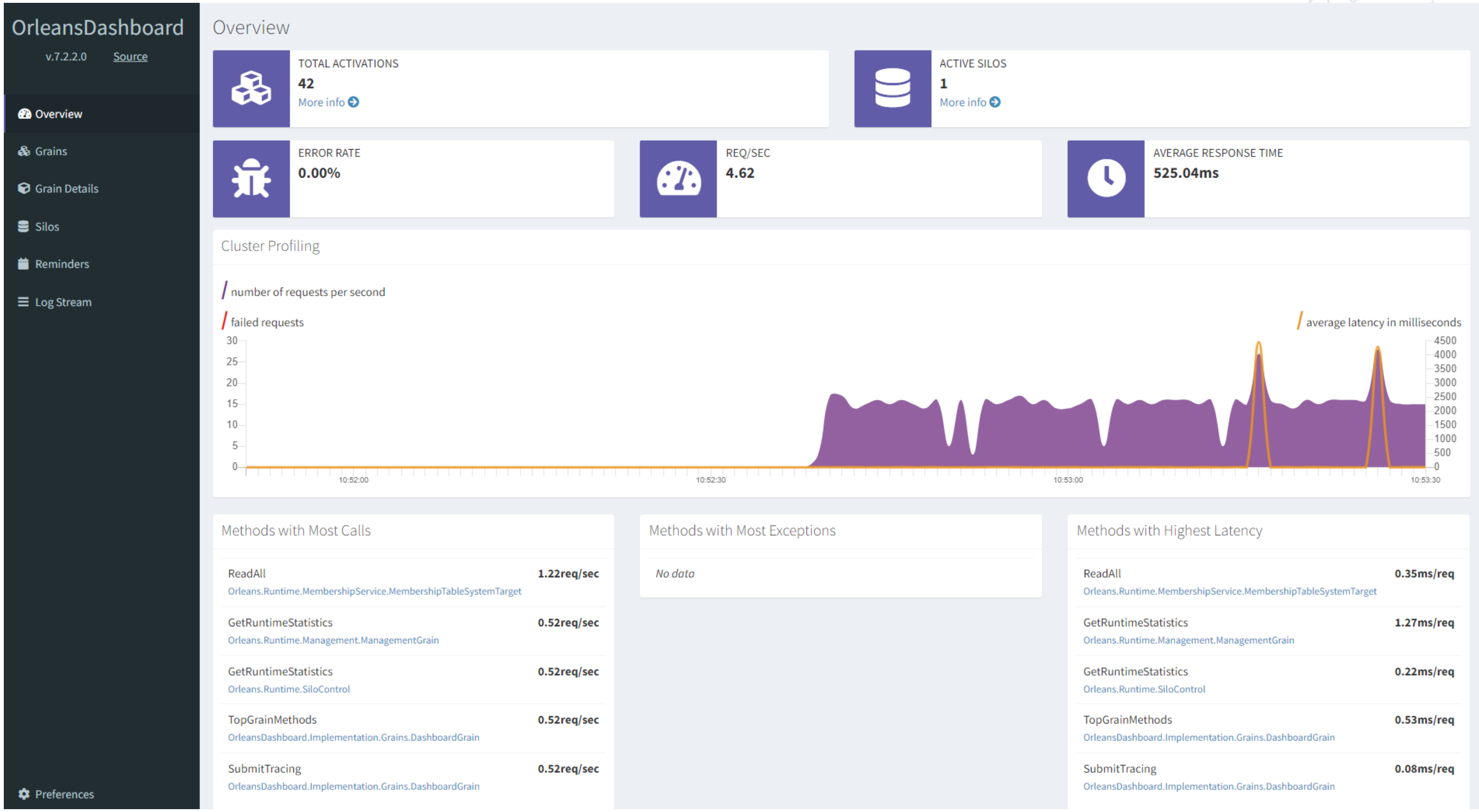
https://learn.microsoft.com/en-us/dotnet/orleans/overview#what-are-silos

Actor model implementations on Orleans - Silo

1. Host grains
2. Responsible to activate and deactivate grains
3. Typically: 1 silo per container/node
4. Could be embedded into main application or in separate container/node
5. Clustering silos is easy

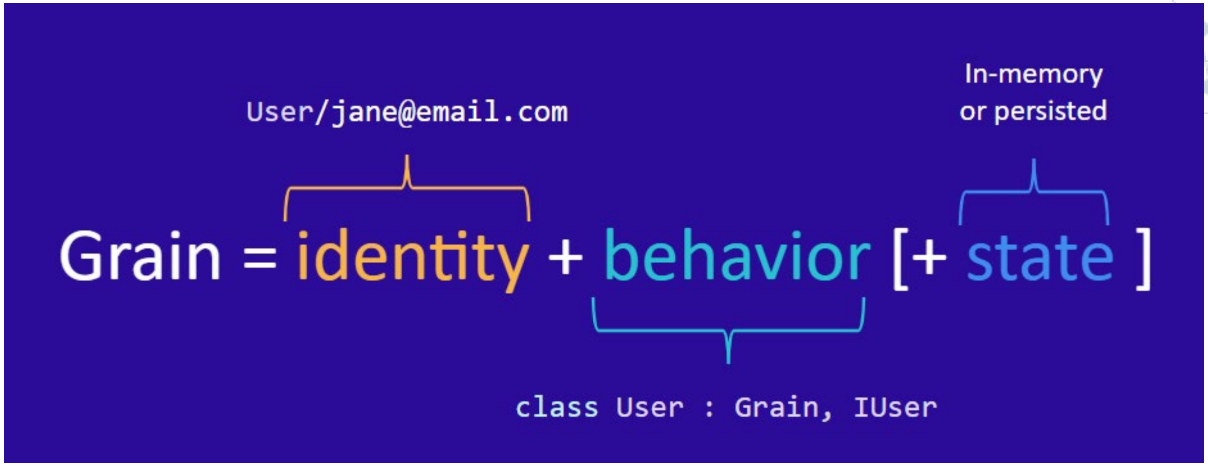
Actor model implementations on Orleans - Dashboard

https://github.com/OrleansContrib/OrleansDashboard



http://localhost:8080

Actor model implementations on Orleans – Calling actors



You can start an actor using grainFactory:

```
_grainFactory.GetGrain<IGrainA>("my-id");
```

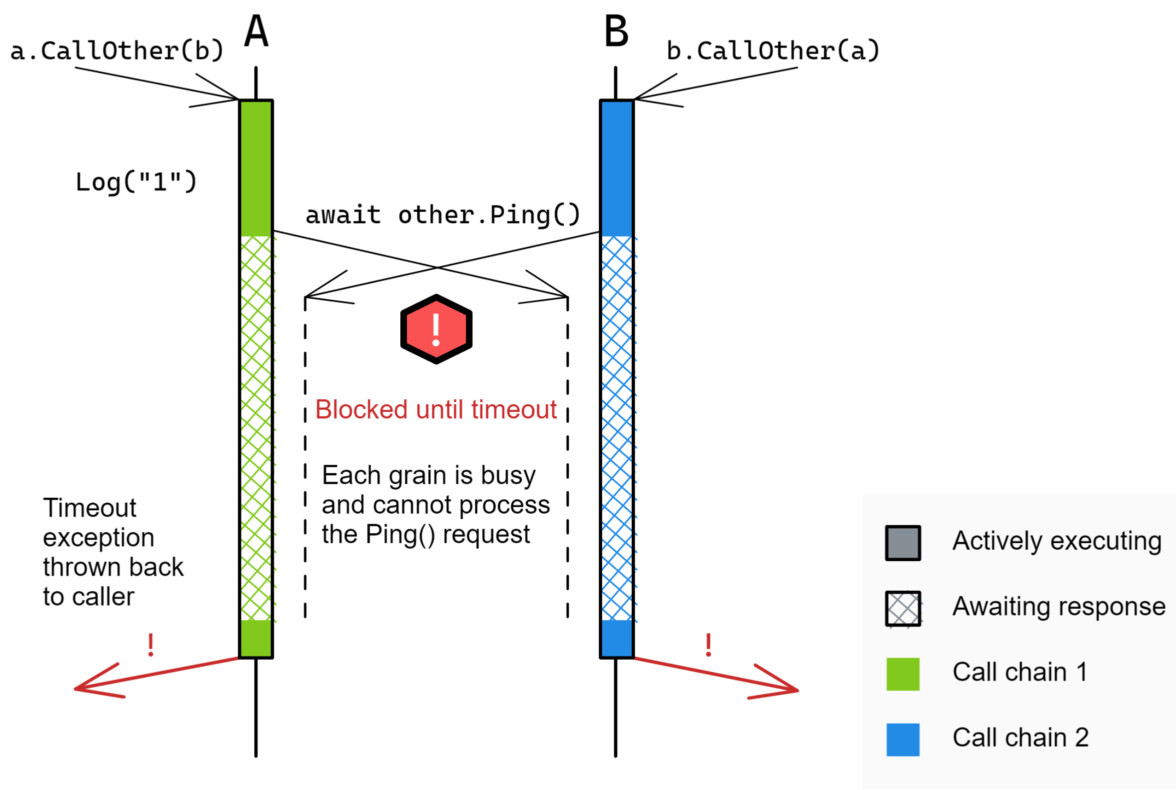
Inside an actor:

```
var grainB = this.GrainFactory.GetGrain<IGrainB>(id);
```

Orleans: Actor mailbox addresses are full typed

Actor model implementations on Orleans – Deadlock

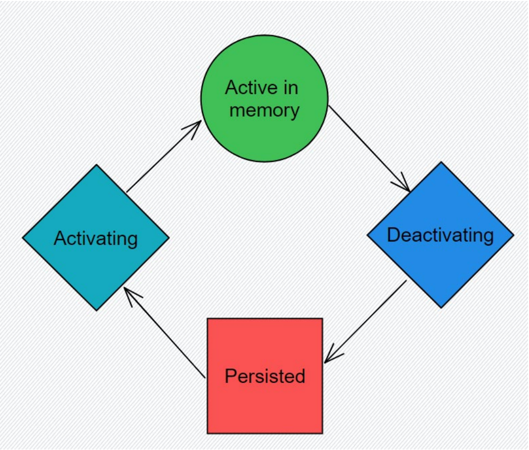
Single thread: Actors process requests one at time



https://learn.microsoft.com/it-it/dotnet/orleans/grains/request-scheduling

Es 14: MicrosoftOrleansDeadlock

Actor model implementations on Orleans – Persistence




```
public HelloGrain(
    [PersistentState("hello")] IPersistentState<HelloState> helloState,
    ILogger<HelloGrain> logger)
{
    _logger = logger;
    _helloState = helloState;
}

public override Task OnActivateAsync(CancellationToken cancellationToken)
{
    return base.OnActivateAsync(cancellationToken);
}

public async Task<string> SayHello(string greeting)
{
    _helloState.State.Counter++;
    _logger.LogInformation("Start say Hello for {grainId} with counter {counter}",
IdentityString, _helloState.State.Counter);
    await Task.Delay(1000);

    // Store state
    await _helloState.WriteStateAsync();

    //DeactivateOnIdle();
    return $"Hello, {greeting}!";
}

public override Task OnDeactivateAsync(DeactivationReason reason, CancellationToken
cancellationToken)
{
    return base.OnDeactivateAsync(reason, cancellationToken);
}
```

Es 15: MicrosoftOrleansPersistence

Actor model implementations on Orleans – Streaming

A typical scenario for Orleans Streams is when you have per-user streams and you want to perform different processing for each user, within the context of an individual user.

Producer

```
_stream = this.GetStreamProvider("StreamProvider").GetStream<int>(streamId);
```

Consumer

```
// ImplicitStreamSubscription attribute here is to subscribe implicitely to all stream within
// a given namespace: whenever some data is pushed to the streams of namespace
Constants.StreamNamespace,
// a grain of type ConsumerGrain with the same guid of the stream will receive the message.
// Even if no activations of the grain currently exist, the runtime will automatically
// create a new one and send the message to it.
[ImplicitStreamSubscription("StreamNamespace")]
public class ConsumerGrain : Grain, IConsumerGrain, IStreamSubscriptionObserver
```

https://learn.microsoft.com/en-us/dotnet/orleans/streaming/streams-why

Es 16: MicrosoftOrleansStreams

Actor model implementations on Orleans – Transactions

Orleans supports distributed ACID transactions against persistent grain state.

```
public interface IAccountGrain : IGrainWithStringKey
{
    [Transaction(TransactionOption.Join)]
    Task Withdraw(int amount);

    [Transaction(TransactionOption.Join)]
    Task Deposit(int amount);

    [Transaction(TransactionOption.CreateOrJoin)]
    Task<int> GetBalance();
}
```

```
await _transactionClient.RunTransaction(
    TransactionOption.Create,
    async () =>
    {
        await fromAccount.Withdraw(transferAmount);
        await toAccount.Deposit(transferAmount);
    });
```

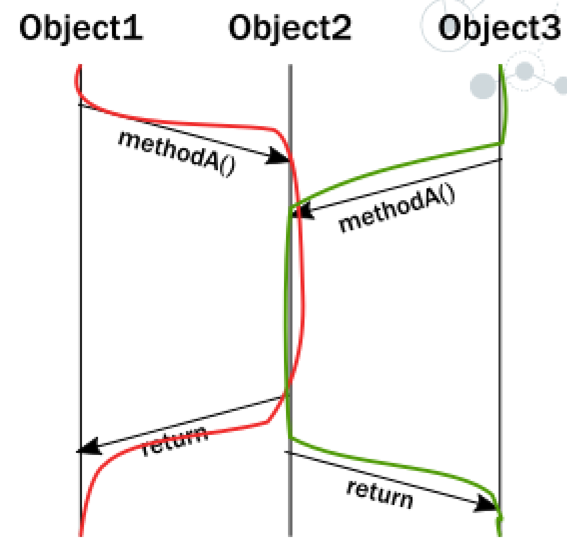
https://learn.microsoft.com/en-us/dotnet/orleans/grains/transactions

Es 17: MicrosoftOrleansTransactions

Actor model: why?

1. Problem with multi thread access

User App



1. Few users call an API
2. Shared services running on same APP
3. Few threads could access same service

https://getakka.net/articles/intro/what-are-actors.html#the-illusion-of-encapsulation

Actor model: why?

1. Problem with multi thread access – classical solution

```
public void Credit(User user, decimal amount)
{
    if (user.amount < 0)
    {
        throw new ArgumentOutOfRangeException(nameof(amount), "The credit amount cannot be negative.");
    }

    lock (balanceLock)
    {
        user.balance += amount;
    }
}
```

Test	IsCorrect	One Thread Ticks	Two Thread Ticks
No Sync	False	385315	668500
Lock Statement	True	1846390	8938287

Lock is not performant

Actor model: why?

1. Problem with multi thread access – actor model solution

1. One actor per user
2. No need to synchronize methods
3. Actors process requests one at time
4. Actors are small

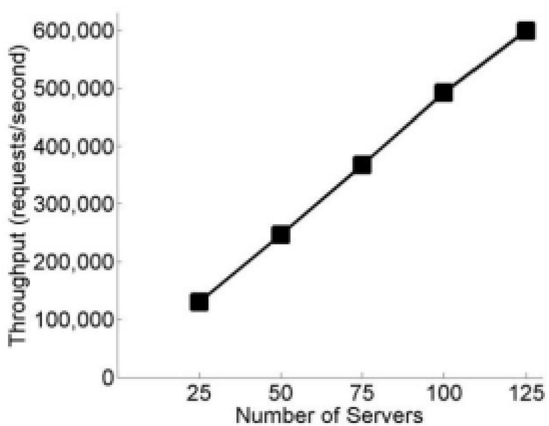


Figure 6: Throughput of Halo 4 Presence service. Linear scalability as number of server increases.

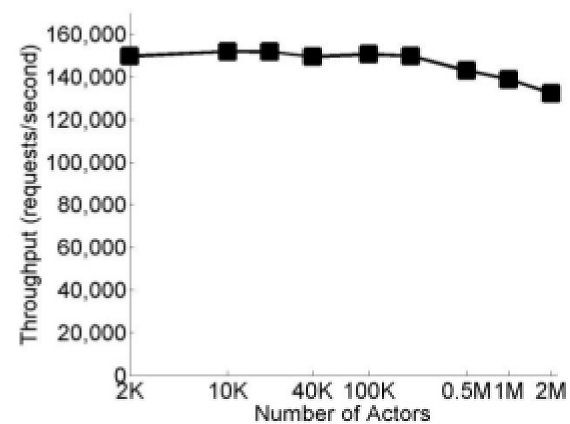
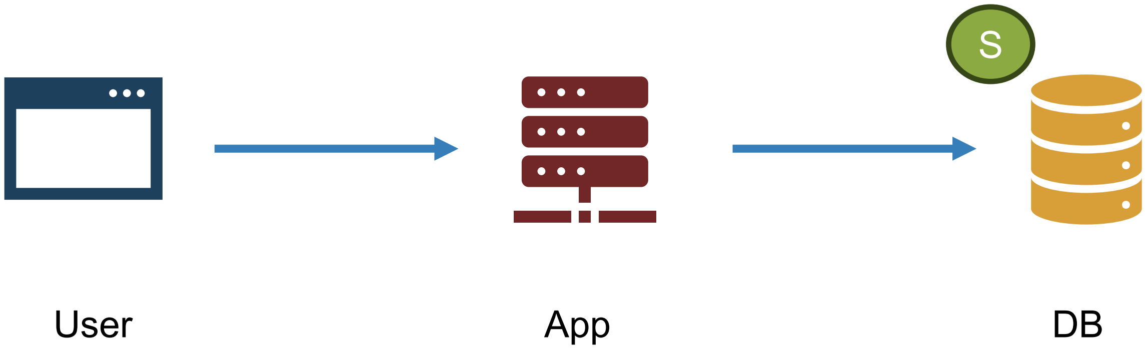


Figure 7: Throughput of Halo 4 Presence service. Linear scalability as number of actors increases.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Orleans-MSR-TR-2014-41.pdf

Actor model: why?

1. Problem with state-less services

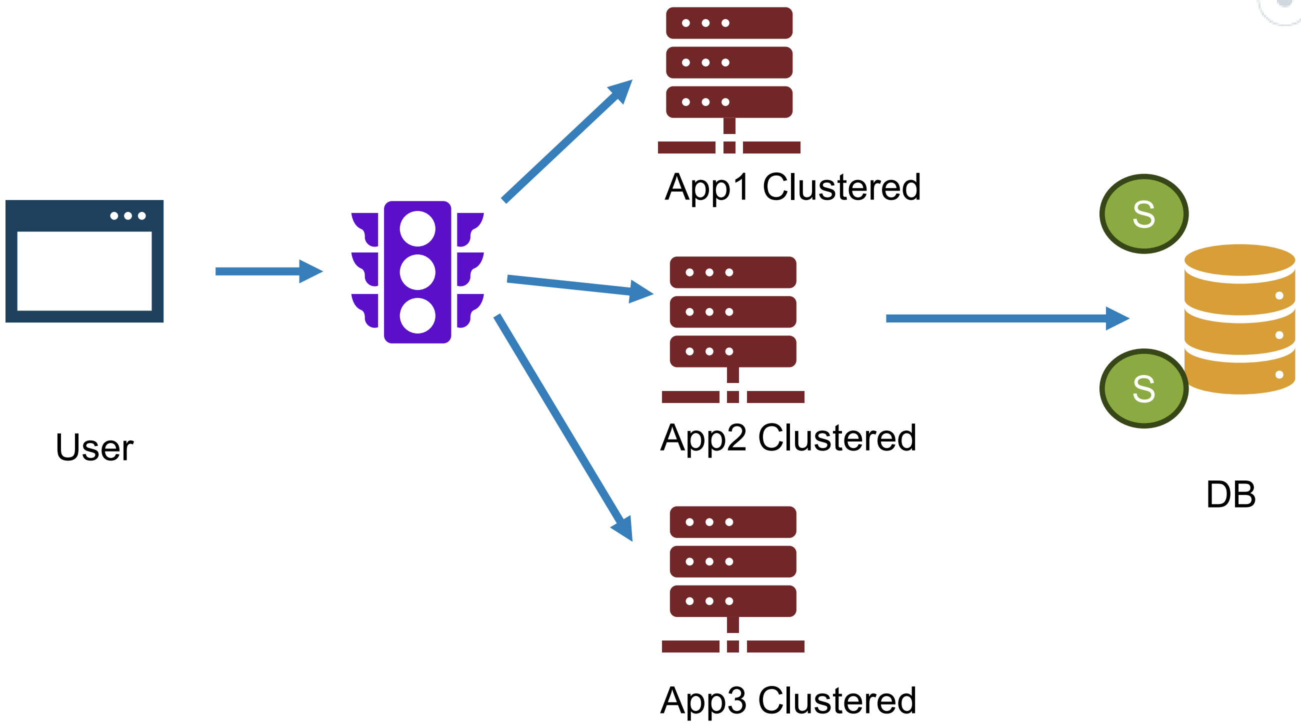


1. User calls an API
2. App loads state from DB
3. App holds state in memory for better performance

https://www.youtube.com/watch?v=iE8cisVgoj8

Actor model: why?

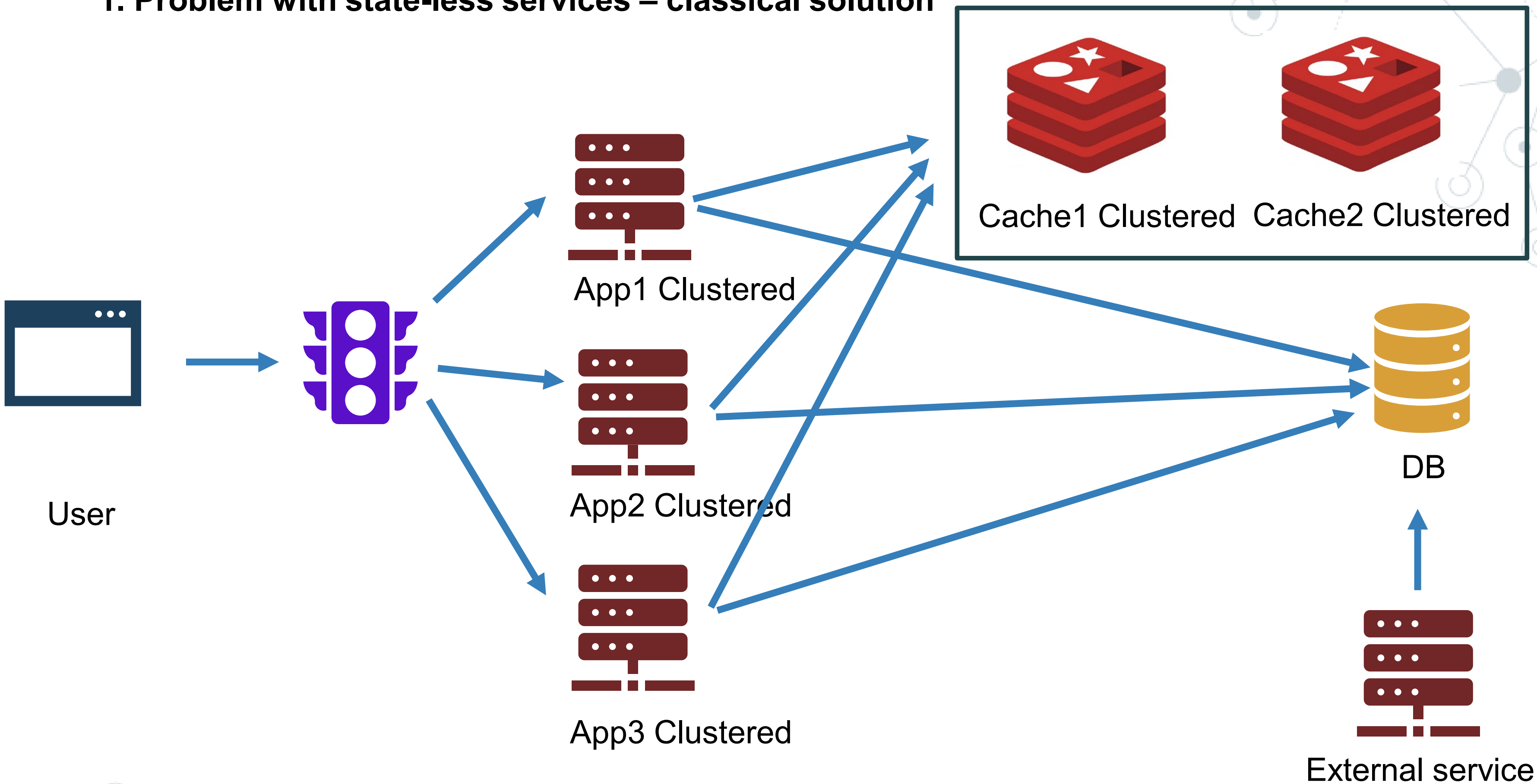
1. Problem with state-less services



1. User calls an API on App1
2. App1 loads state from DB
3. App1 holds state in memory for better performance
4. User calls an API on App3

Actor model: why?

1. Problem with state-less services – classical solution

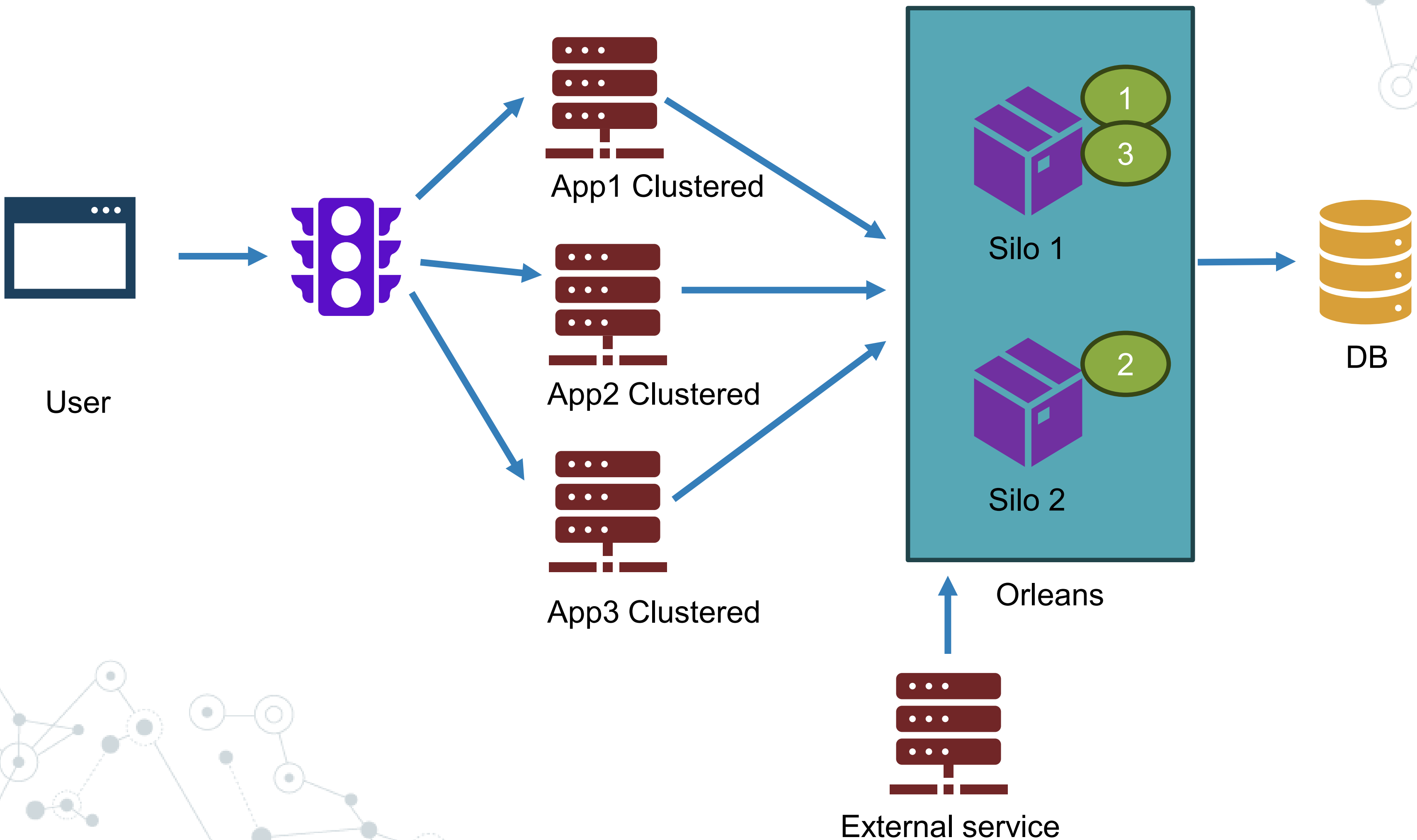


There are only two hard things in Computer Science: cache invalidation and naming things.

-- Phil Karlton

Actor model: why?

1. Problem with state-less services – actor model solution



Actor model: when?

✓

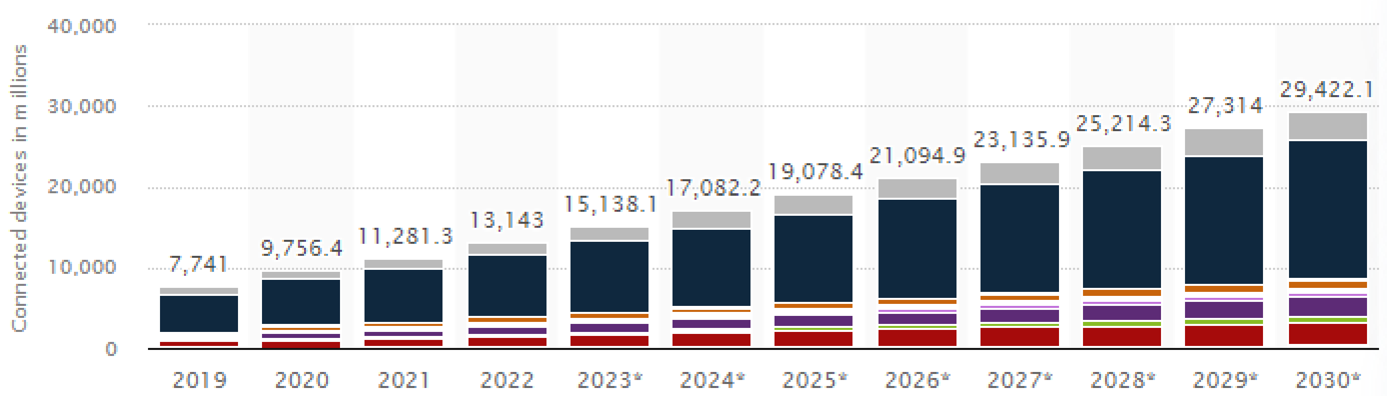
1. Actor are small enough to be single-thread
2. Many entities loosely coupled (billions!)
3. No need of a global coordinator, only between actors
4. You know your project

✗

1. Entity must access to the state of other entities
2. Entities relations are complex (ERP, MES…)
3. Small entities but fat
4. You don’t know your project

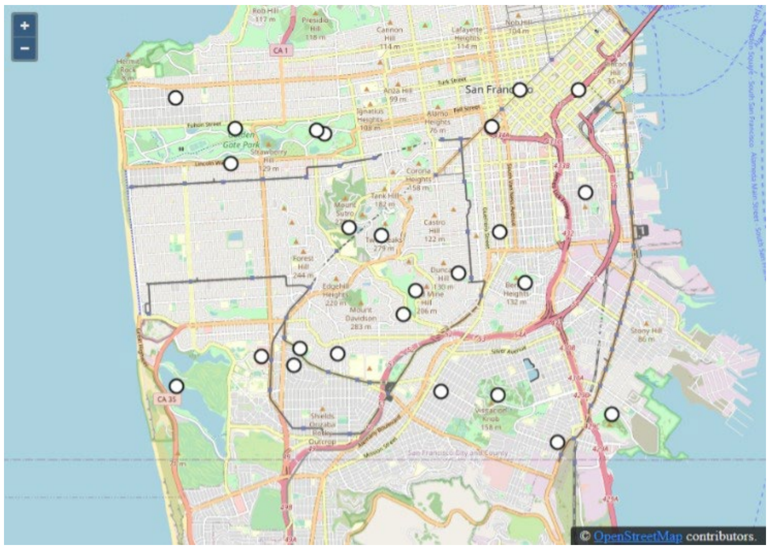
Actor model: examples

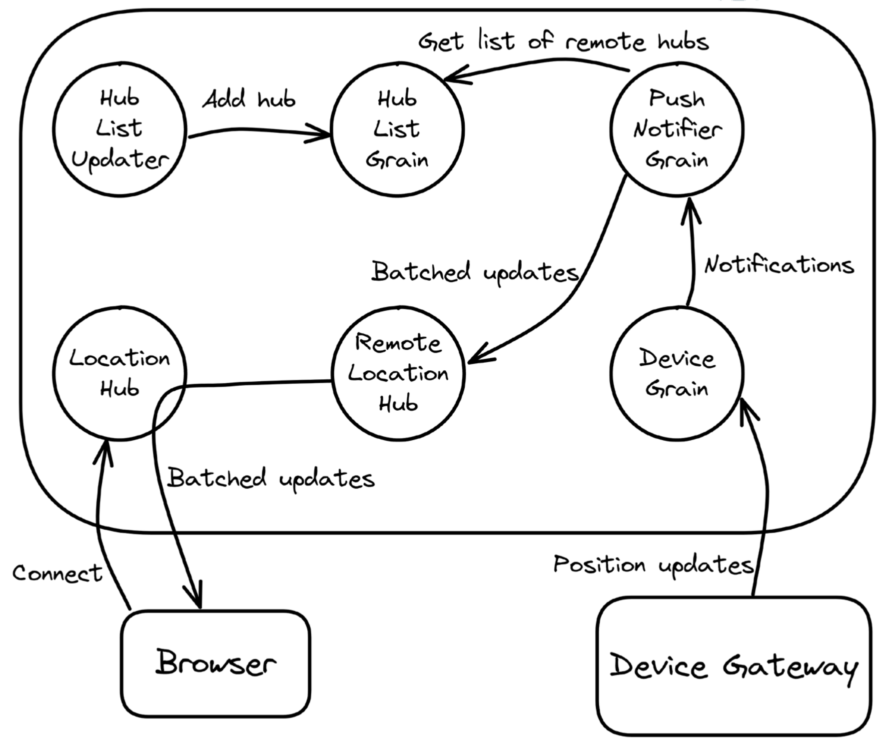
Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030



https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/

Actor model: examples





https://learn.microsoft.com/en-us/dotnet/orleans/tutorials-and-samples/

•Security in Distributed Applications

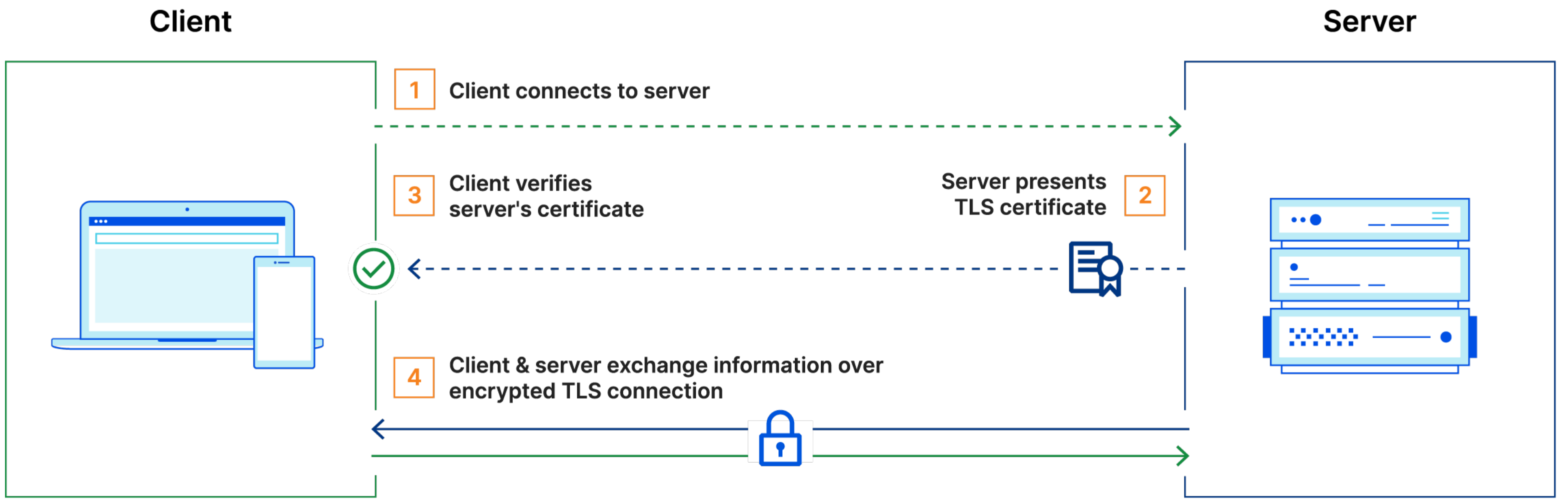
Man in the middle

Different services with different protocols:

1. **Web http/https**
2. **gRPC**
3. **AMQP**
4. **Database**

Man in the middle

TLS: the server has a TLS certificate and a public/private key pair, while the client does not

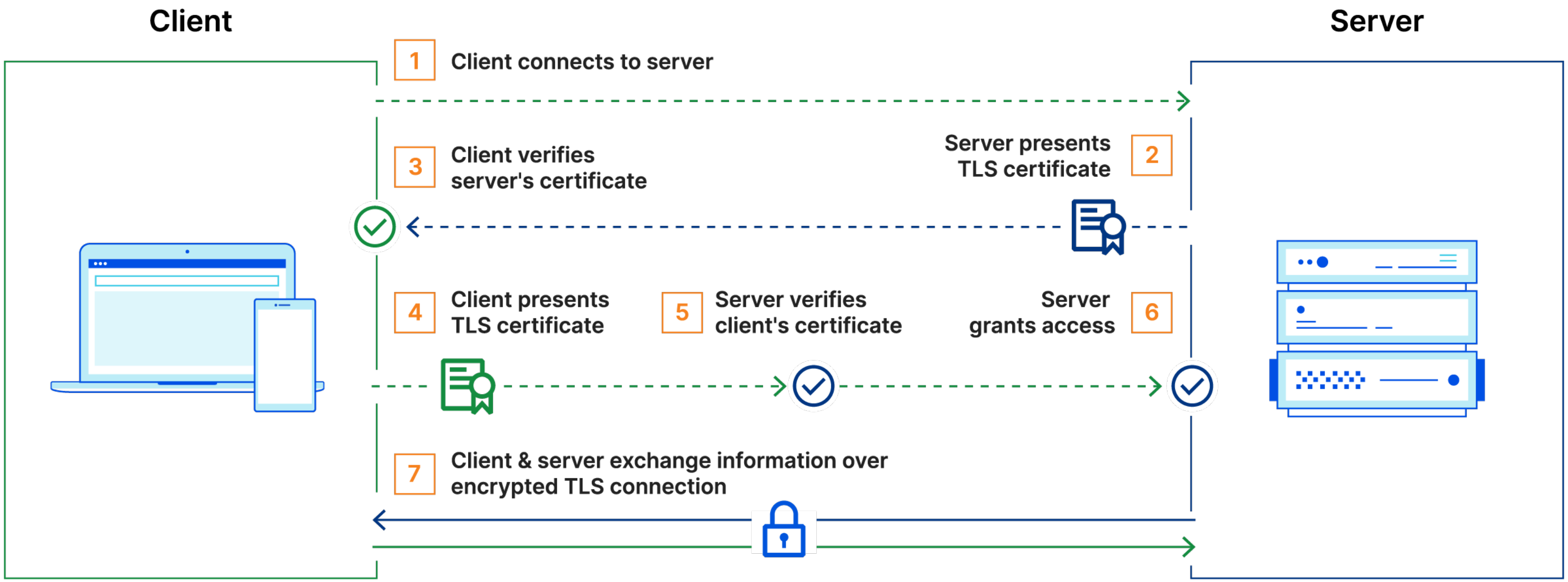


But we have server to server communications!

Man in the middle

mTLS: mutual TLS (internal CA)

**Zero Trust means that no user, device, or network traffic is trusted by default, an approach that helps eliminate many security vulnerabilities.*



https://www.elastic.co/guide/en/kibana/current/elasticsearch-mutual-tls.html

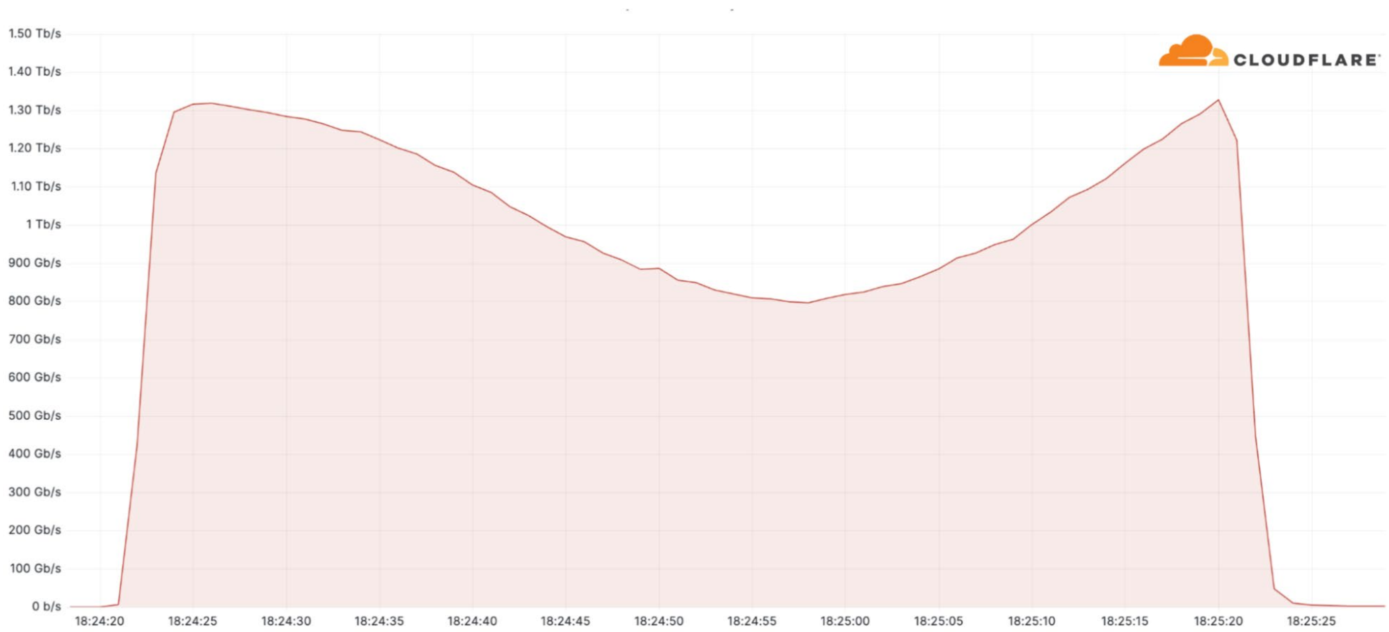
https://www.rabbitmq.com/ssl.html#peer-verification
https://learn.microsoft.com/en-us/samples/dotnet/samples/orleans-transport-layer-security-tls/

Distributed Denial of Service

Million of requests per seconds from different clients

Distributed Denial of Service

https://blog.cloudflare.com/ddos-threat-report-2023-q1/



Cloud providers have few services.

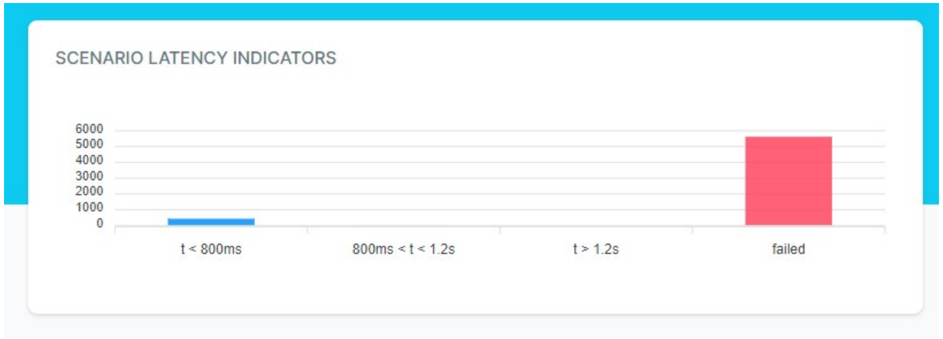
https://azure.microsoft.com/it-it/products/ddos-protection/

https://aws.amazon.com/it/shield/

Distributed Denial of Service

Rate limit on http:

`429 Too Many Requests` The 429 status code indicates that the user has sent too many requests in a given amount of time ("rate limiting").



https://learn.microsoft.com/en-us/aspnet/core/performance/rate-limit?view=aspnetcore-8.0

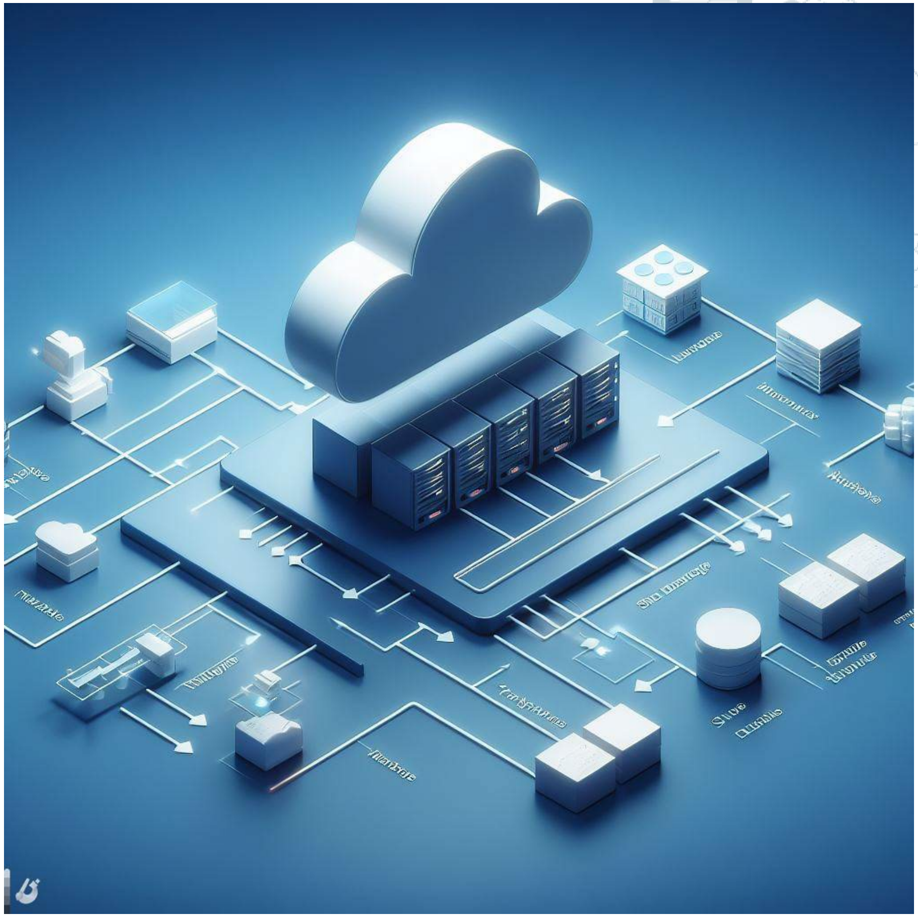
Es 18: MicrosoftRateLimit

Handling secrets

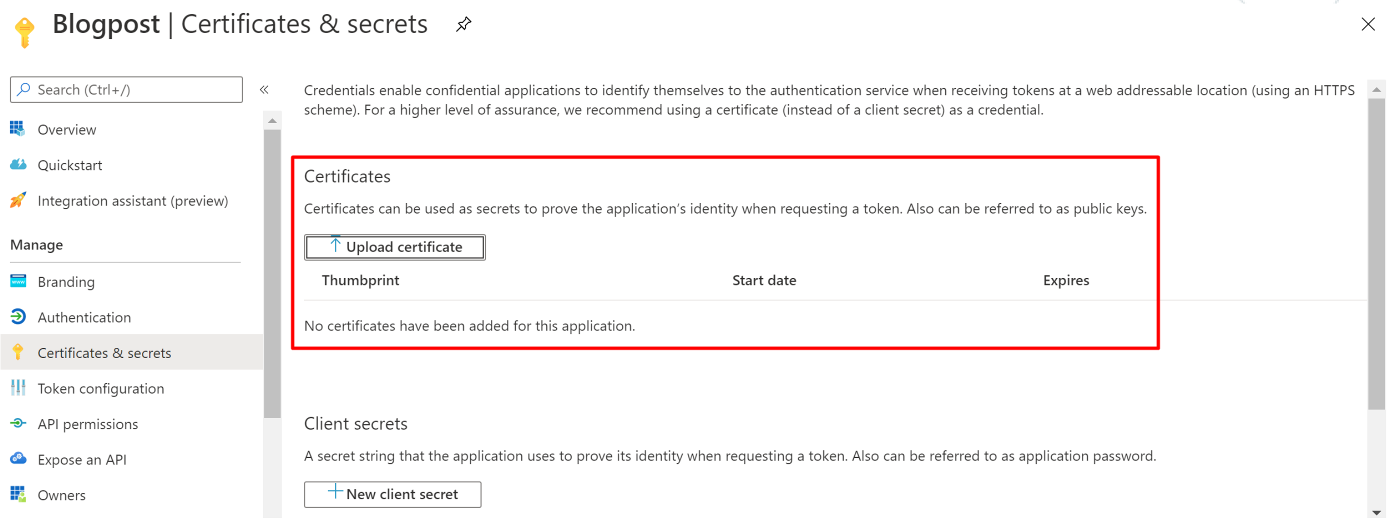
Services could need to connect:

1. Databases
2. Caches
3. External services on cloud
4. Other clusters
5. Other services

How to handle secrets correctly?



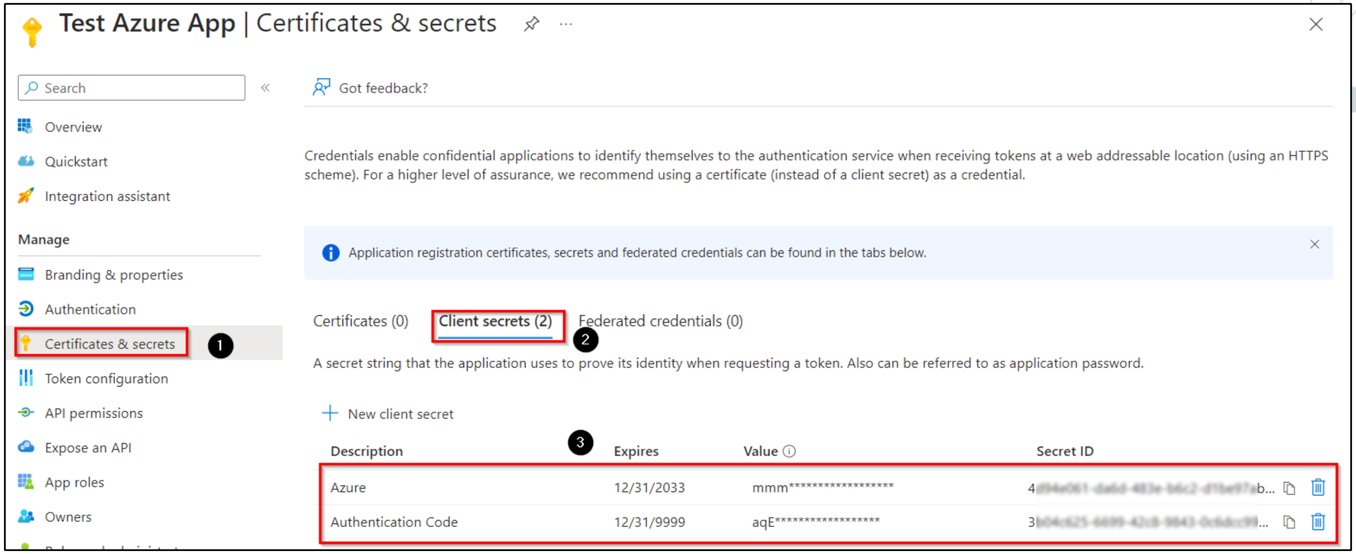
Handling secrets



Using certificates to prove application identity!

1. No need to share password
2. Security is on network layer (mTLS)

Handling secrets



Using secrets to prove application identity!

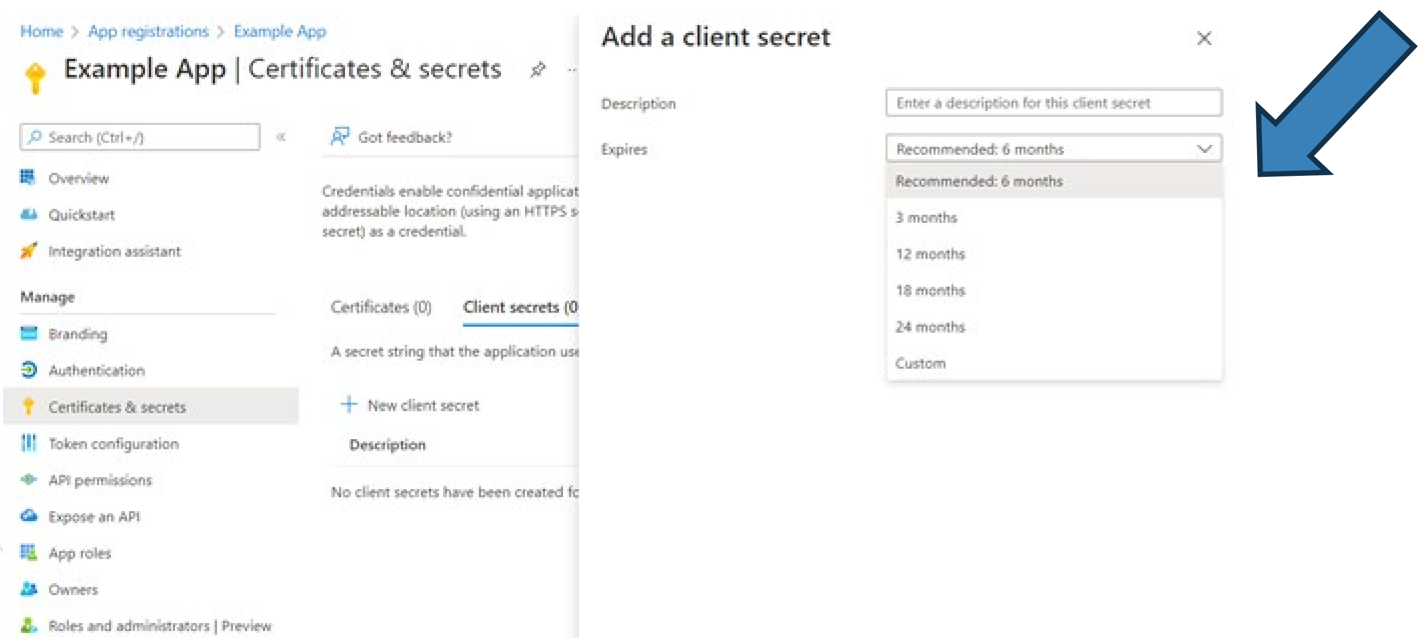
1. Services must send secret to other service
2. Security is on application layer

Handling secrets

What happens is a certificate or secrets is stolen?

Problems:

1. If a certificate/secrets is compromised on one single service, I must invalidate it
2. Change certificate/secrets could be done on runtime but on cluster is complex
3. Certificates/Secrets must have an expire time



Handling secrets

Service to handle secrets

AzureKeyVault

HashiCorp Vault

AWS Secrets Manager

Secrets management

Centrally store, access, and deploy secrets across applications, systems, and infrastructure.

Dynamic secrets

A dynamic secret is generated on demand and is unique to a client, instead of a static secret, which is defined ahead of time and shared.

Kubernetes secrets

Install Vault using a Helm chart and then leverage Vault and Kubernetes to securely inject secrets into your application stack.

Database credential rotation

Automatically rotate database passwords with Vault's database secrets engine.

Automated PKI infrastructure

Use Vault to quickly create X.509 certificates on demand and reduce the manual overhead.

Identity-based access

Authenticate and access different clouds, systems, and endpoints using trusted identities.

Data encryption and tokenization

Keep application data secure with one centralized workflow for data that resides in untrusted or semi-trusted systems outside of Vault.

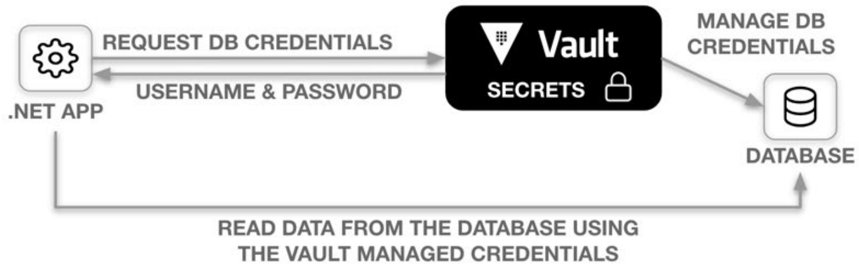
Key management

Use a standardized workflow for distribution and lifecycle management across KMS providers.

https://www.vaultproject.io/

Handling secrets

How to use it?




```
static async Task Main(string[] args)
{
    // Initialize one of the several auth methods.
    IAuthMethodInfo authMethod = new TokenAuthMethodInfo("testtoken");

    // Initialize settings. You can also set proxies, custom delegates etc. here.
    var vaultClientSettings = new VaultClientSettings("http://localhost:8200", authMethod);

    IVaultClient vaultClient = new VaultClient(vaultClientSettings);

    var myKeys = await vaultClient.V1.Secrets.Cubbyhole.ReadSecretAsync("my-path");
}
```

Es 19: SecretsWithVault

Handling secrets

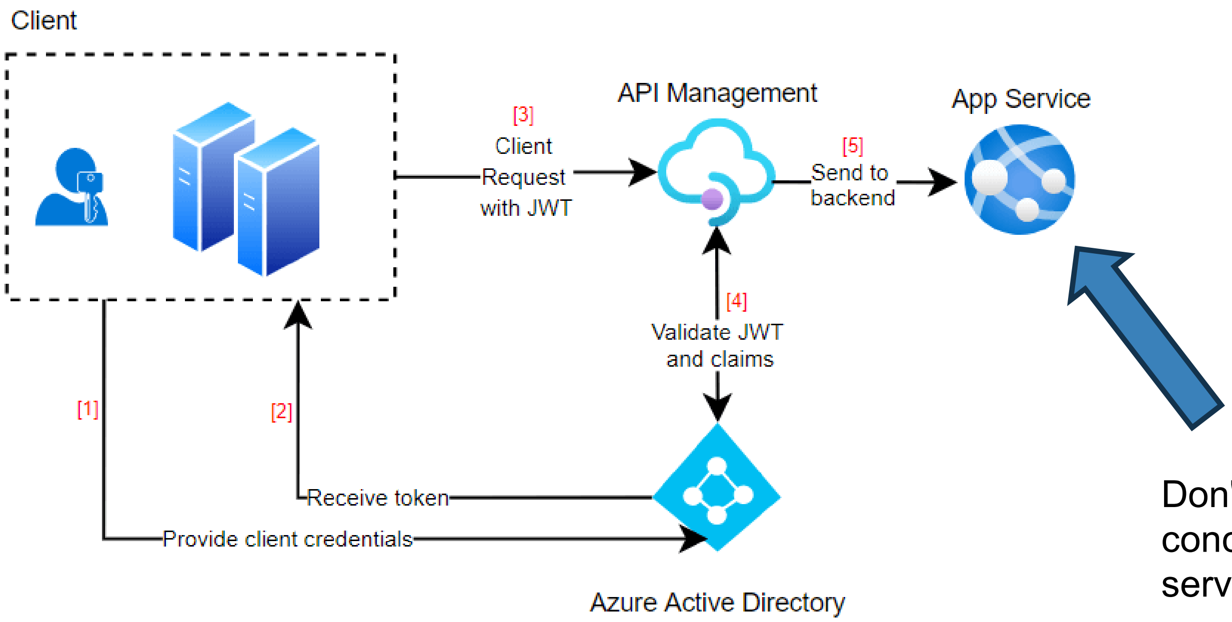
How to use it?

C#   Copy

```csharp
SecretClientOptions options = new SecretClientOptions()
    {
        Retry =
        {
            Delay= TimeSpan.FromSeconds(2),
            MaxDelay = TimeSpan.FromSeconds(16),
            MaxRetries = 5,
            Mode = RetryMode.Exponential
         }
    };
var client = new SecretClient(new Uri("https://<your-unique-key-vault-name>.vault.azure.net/"), new DefaultAz

KeyVaultSecret secret = client.GetSecret("<mySecret>");

string secretValue = secret.Value;
```

User Authorization

User authentication/authorization



Don't spread security concepts around your services

How can we manage Authorization in distributed application?

Contexts

How can we manage Authorization in distributed application?

Context: a way to pass data between methods and grains

Set context

```
RequestContext.Set("UserRole", "Admin");
```

Get context

```
RequestContext.Get("UserRole");
```

Statics methods but we are in a multi thread environment!

https://learn.microsoft.com/en-us/dotnet/orleans/grains/request-context

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/http-context

Es 20: MicrosoftOrleansRequestContext

Contexts

AsyncLocal

Represents ambient data that is local to a given asynchronous control flow, such as an asynchronous method.

AsyncLocal<T> is used to persist a value across an asynchronous flow.

https://learn.microsoft.com/en-us/dotnet/api/system.threading.asynclocal-1?view=net-8.0

.NET Aspire

.NET Aspire

A cloud ready stack for building observable, production ready, distributed applications

First Preview Available Today

aka.ms/dotnet-aspire

Engage with team on GitHub

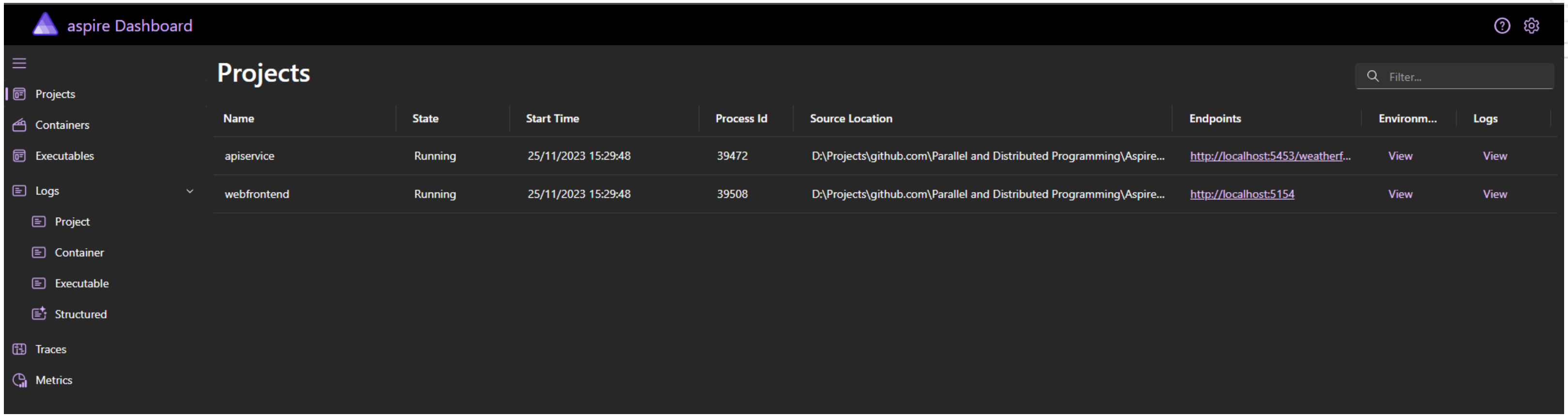
github.com/dotnet/aspire

.NET Aspire

.NET Aspire is an **opinionated** stack for building resilient, observable, and configurable cloud-native applications with .NET

```
var builder = DistributedApplication.CreateBuilder(args);

var apiservice = builder.AddProject<Projects.aspire_ApiService>("apiservice");

builder.AddProject<Projects.aspire_Web>("webfrontend")
    .WithReference(apiservice);

builder.Build().Run();
```

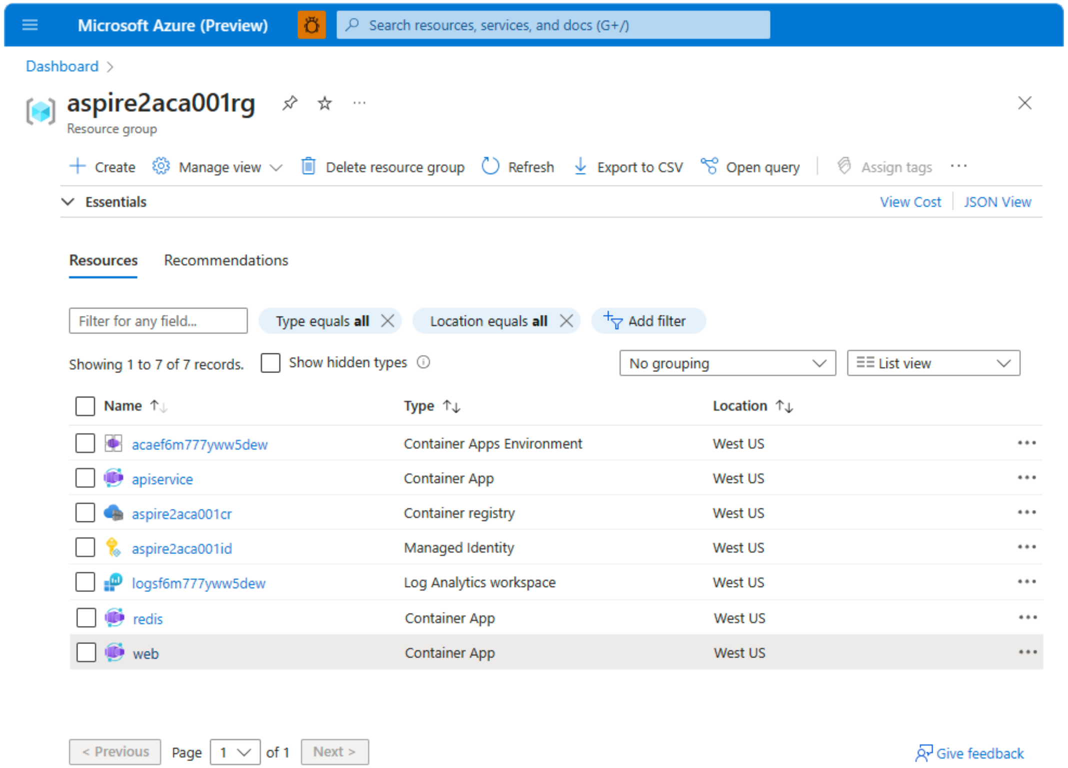
.NET Aspire: dashboard



Es 21: Aspire

.NET Aspire: deploy

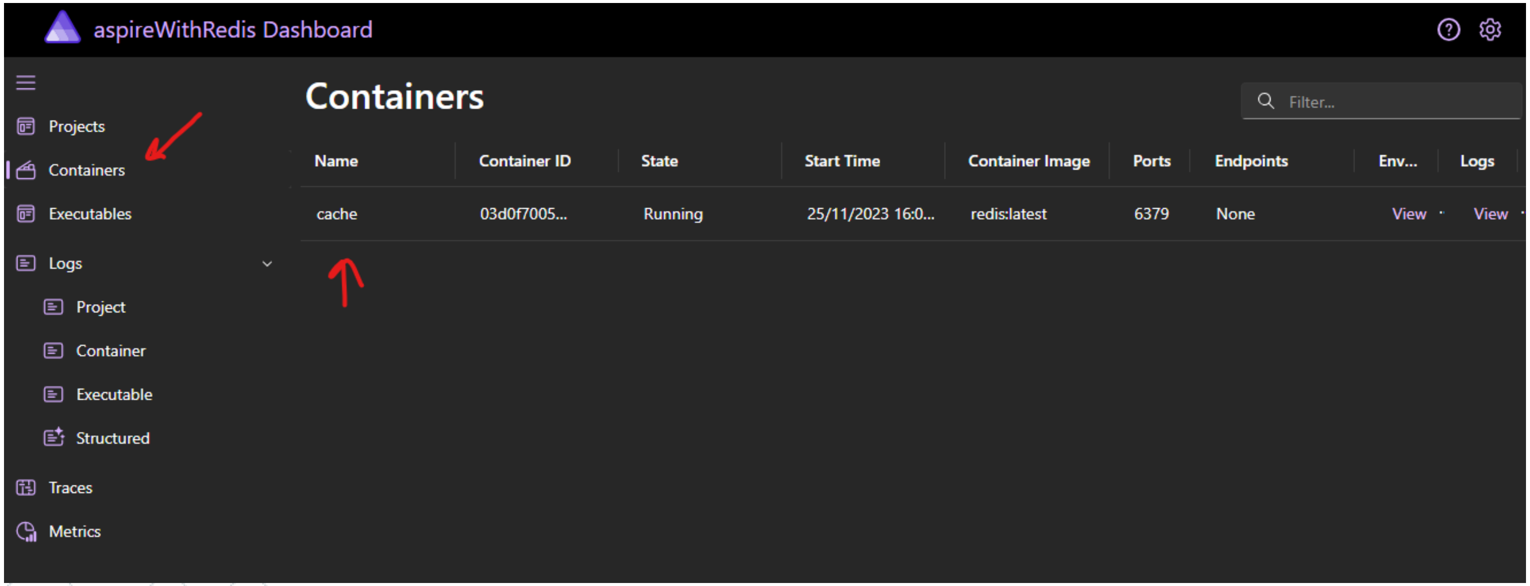
https://learn.microsoft.com/en-us/dotnet/aspire/deployment/overview




```
dotnet run --project .\aspire.AppHost\aspire.AppHost.csproj --publisher
manifest --output-path aspire-manifest.json
```

.NET Aspire: infrastructure as code

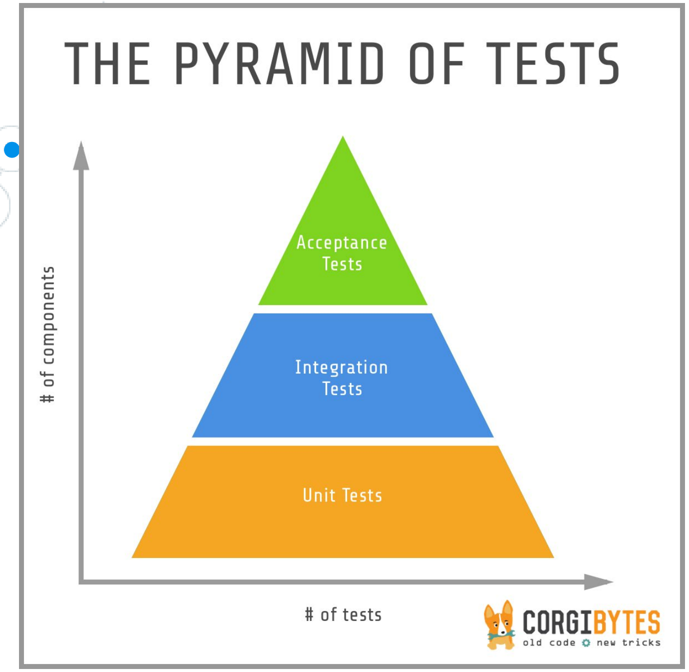
```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedisContainer("cache");

var apiservice =
builder.AddProject<Projects.aspireWithRedis_ApiService>("apiservice");

builder.AddProject<Projects.aspireWithRedis_Web>("webfrontend")
    .WithReference(apiservice)
    .WithReference(cache);

builder.Build().Run();
```

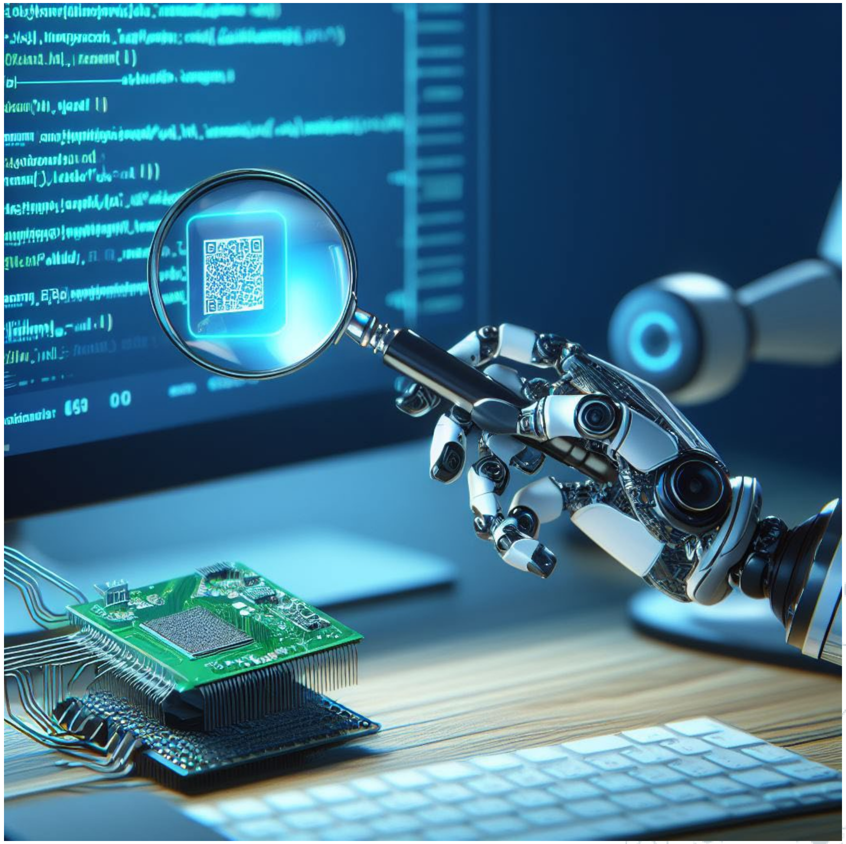


https://www.youtube.com/watch?v=DORZA_S7f9w
https://www.youtube.com/watch?v=HYe6y1kBuGI

Es 22: Aspire with Redis

Testing


THE PYRAMID OF TESTS
Acceptance Tests
Integration Tests
Unit Tests
# of components
# of tests
CORGIBYTES
old code new tricks



Unit test

```
public sealed class HelloGrain : Grain, IHelloGrain
{
    public HelloGrain()
    {
    }

    public async Task<string> SayHello(string greeting)
    {
        await Task.Delay(100);
        return $"Hello, {greeting}!";
    }
}
```

```
namespace ProjectToTest.Tests
{
    public class HelloGrainTests
    {
        [Fact]
        public async Task TestSayHello()
        {
            // ARRANGE
            var helloGrain = new HelloGrain();

            // ACT
            var result = await helloGrain.SayHello("Diego");

            // ASSERT
            Assert.Equal("Hello, Diego!", result);
        }
    }
}
```

Es 23: ProjectToTest

Unit test: mock a service

```
public sealed class HelloGrainUsingAService : Grain, IHelloGrainUsingAService
{
    private readonly IAService _service;

    public HelloGrainUsingAService(IAService service)
    {
        _service = service;
    }

    public async Task<int> Count()
    {
        return await _service.GetCoundFromDataBase();
    }
}
```

NSubstitute

A friendly substitute for .NET mocking libraries

Es 23: ProjectToTest

Unit test: mock a service

```
public class HelloGrainUsingAServiceTests
{
    [Fact]
    public async Task TestCount()
    {
        // ARRANGE
        var service = Substitute.For<IAService>();
        service.GetCoundFromDataBase().Returns(5);

        var helloGrain = new HelloGrainUsingAService(service);

        // ACT
        var result = await helloGrain.Count();

        // ASSERT
        Assert.Equal(5, result);
    }
}
```

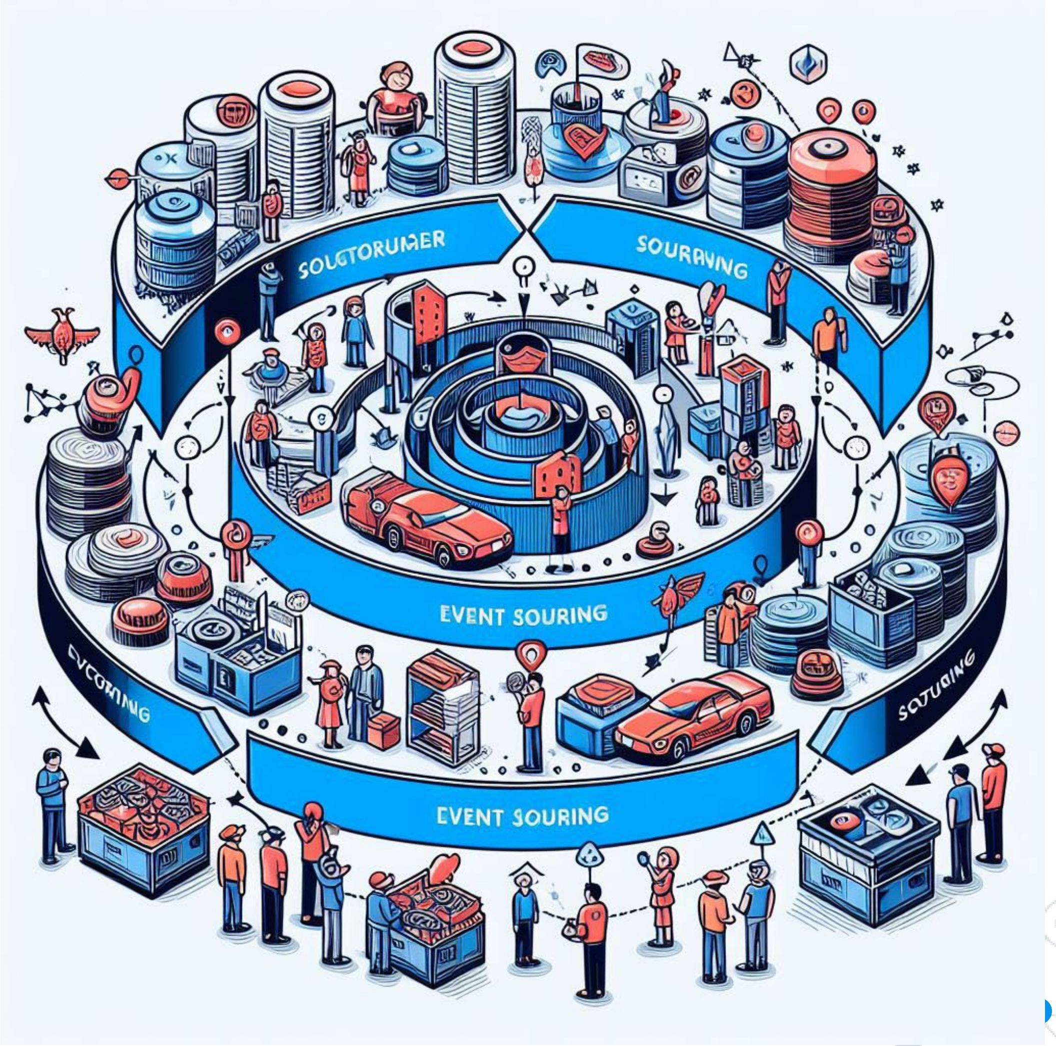
Unit test: Orleans

The **Microsoft.Orleans.TestingHost** NuGet package contains **TestCluster** which can be used to create an in-memory cluster, comprised of two silos by default, which can be used to test grains.

```
public class HelloGrainTestsTestCluster
{
    [Fact]
    public async Task TestSayHello()
    {
        // ARRANGE
        var builder = new TestClusterBuilder();
        var cluster = builder.Build();
        cluster.Deploy();

        // ACT
        var hello = cluster.GrainFactory.GetGrain<IHelloGrain>("my-id");
        var result = await hello.SayHello("Diego");
        cluster.StopAllSilos();

        // ASSERT
        Assert.Equal("Hello, Diego!", result);
    }
}
```

https://learn.microsoft.com/en-us/dotnet/orleans/tutorials-and-samples/testing

Event Sourcing



Crud

Applications store their current state in a database:

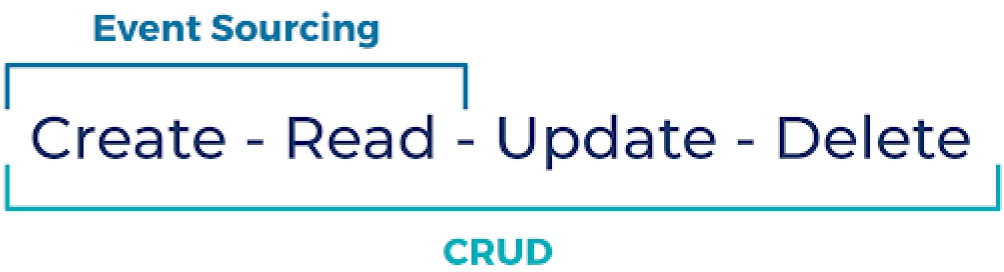
1) Previous state is lost
2) No way to restore states
3) Store operation could be slow
4) Data update conflicts
5) History is lost

Create - Read - Update - Delete

CRUD

https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing

Event Sourcing



Event Sourcing does not persist the current state of a record, but instead stores the individual changes as a series of deltas that led to the current state over time.

Similar to the way a bank manages an account

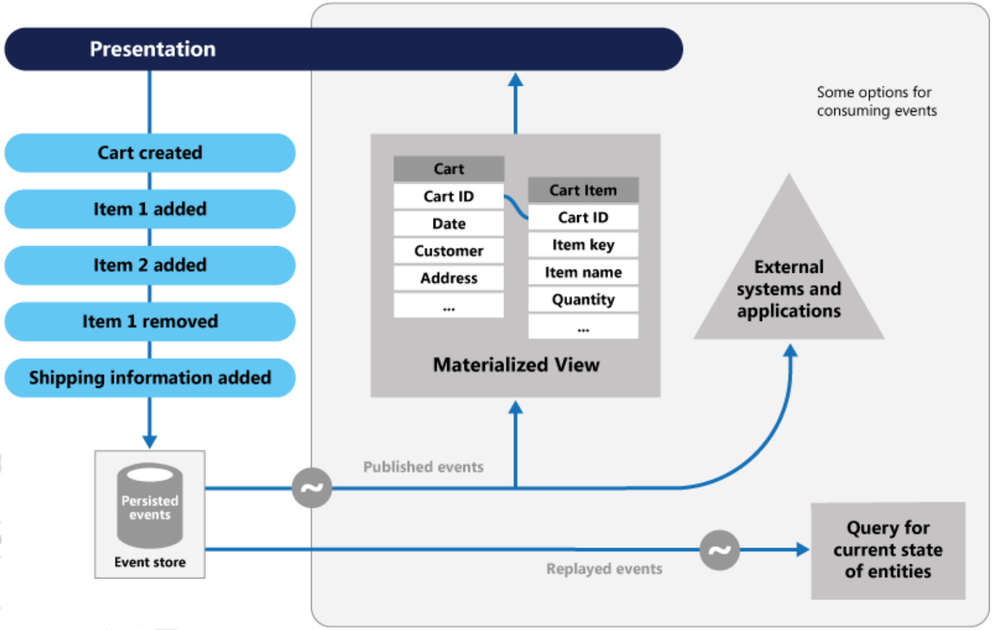
```
  500 (deposit)
+ 200 (deposit)
- 300 (payment)
  ---
= 400 (balance)
```

Events are immutable and can be stored using an append-only operation.

Event Sourcing: storing data as events

Event sourcing is a Microservice design pattern that involves capturing all changes to an application's state as a **sequence of events**, rather than simply updating the state itself. Each event **represents a discrete change** to the system and is stored in an event log, which can be used to **reconstruct the system's state at any point in time**.

1) The complete history of changes is available for auditing purposes.
2) The ability to query the state of the system at any point in time.
3) Easy integration with distributed systems.
4) Event-driven systems can scale horizontally by adding more event consumers.
5) Easier to trace and diagnose issues by examining the event log.



Event Sourcing: problems

Complexity

Event sourcing can introduce complexity, especially in understanding the flow of events and reconstructing the current state from a series of events.

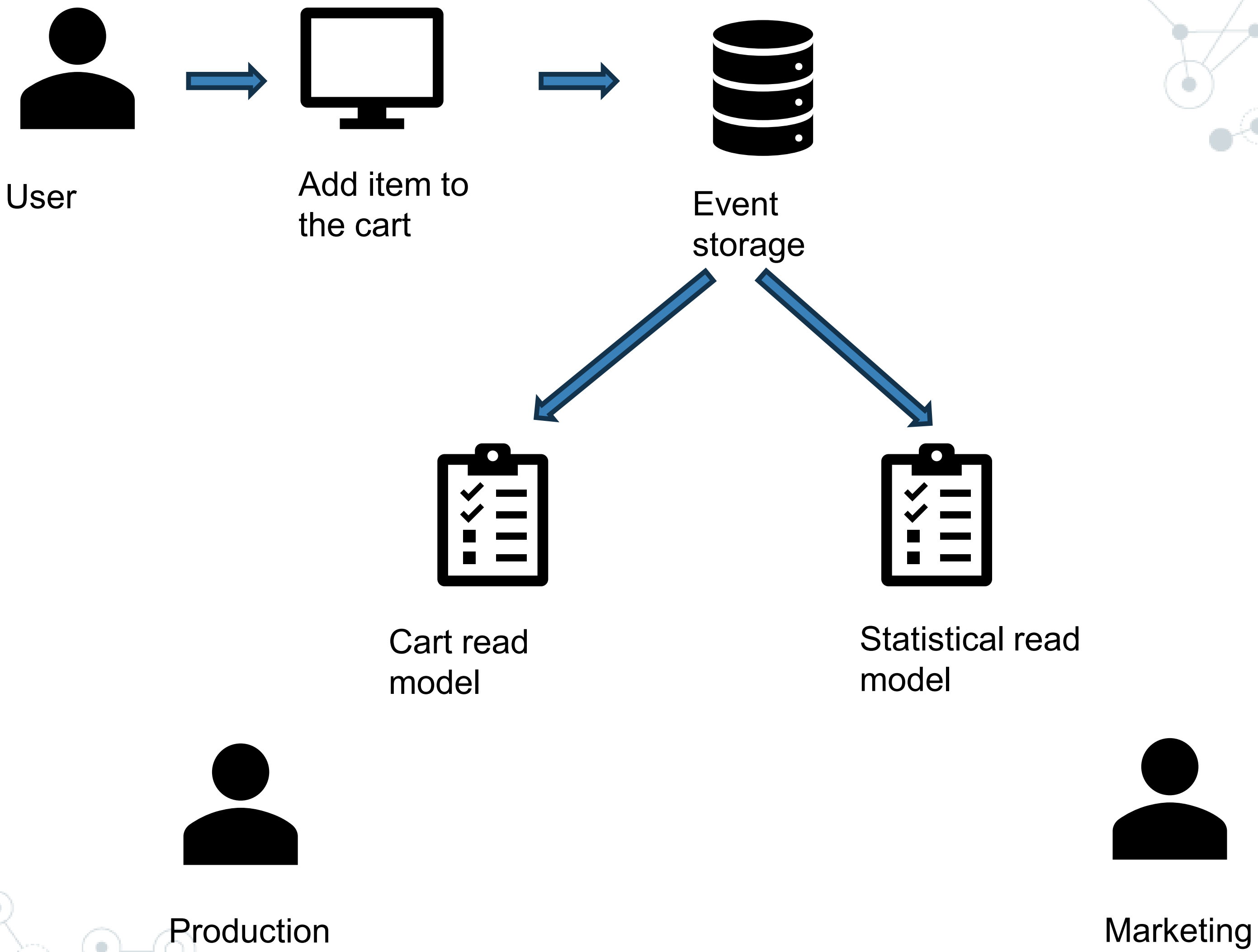
Performance

the process of replaying events to rebuild state or responding to queries might impact performance, especially as the volume of events grows

Storage

Storing every change as an event can lead to increased storage requirements compared to traditional CRUD-based approaches.

Event Sourcing: read models



Es 24: EventSourcing

https://www.davidguida.net/event-sourcing-in-net-core-part-1-a-gentle-introduction/