Parallel and Distributed Programming

Hello!

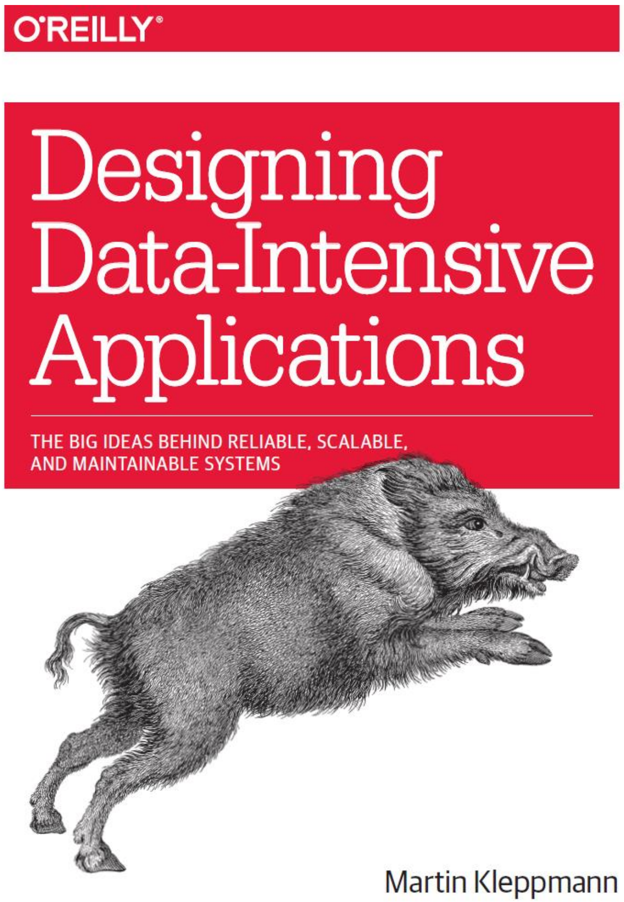
I am Diego Bonura

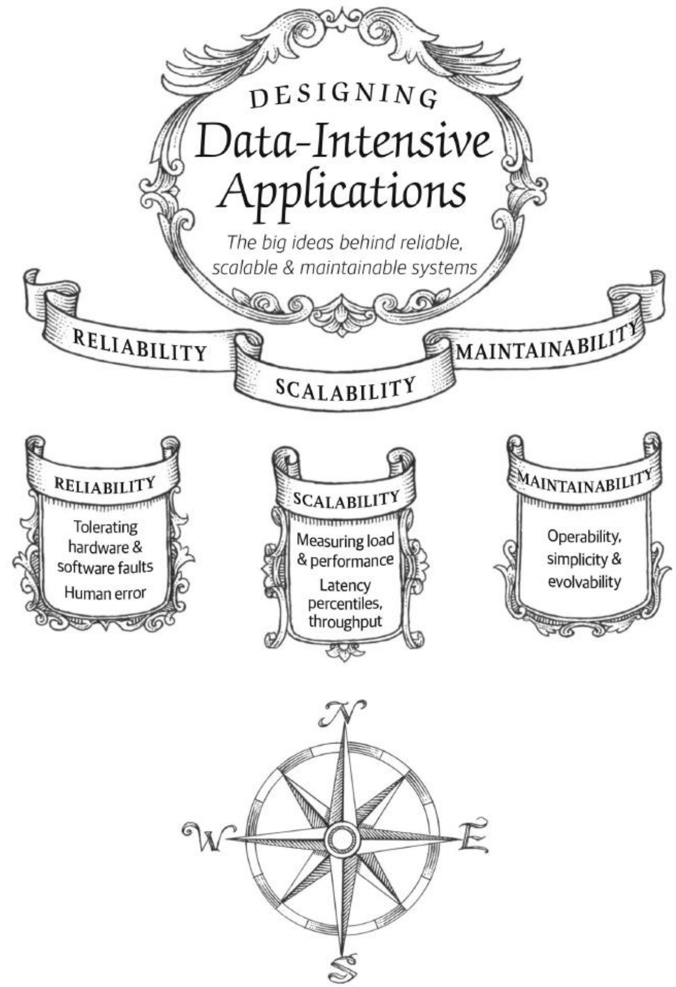
Mi occupo di:

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

diego@bonura.dev

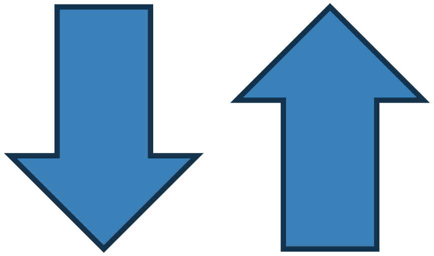
https://medium.com/@diegobonura


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

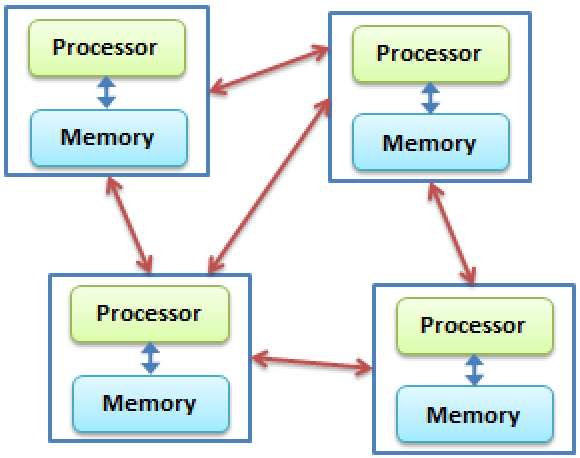
"

Distribuited programming is complex

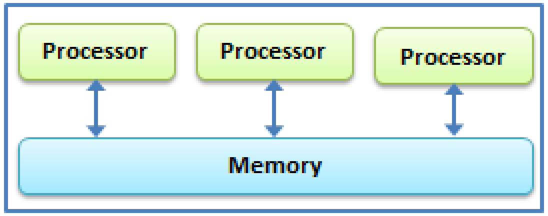


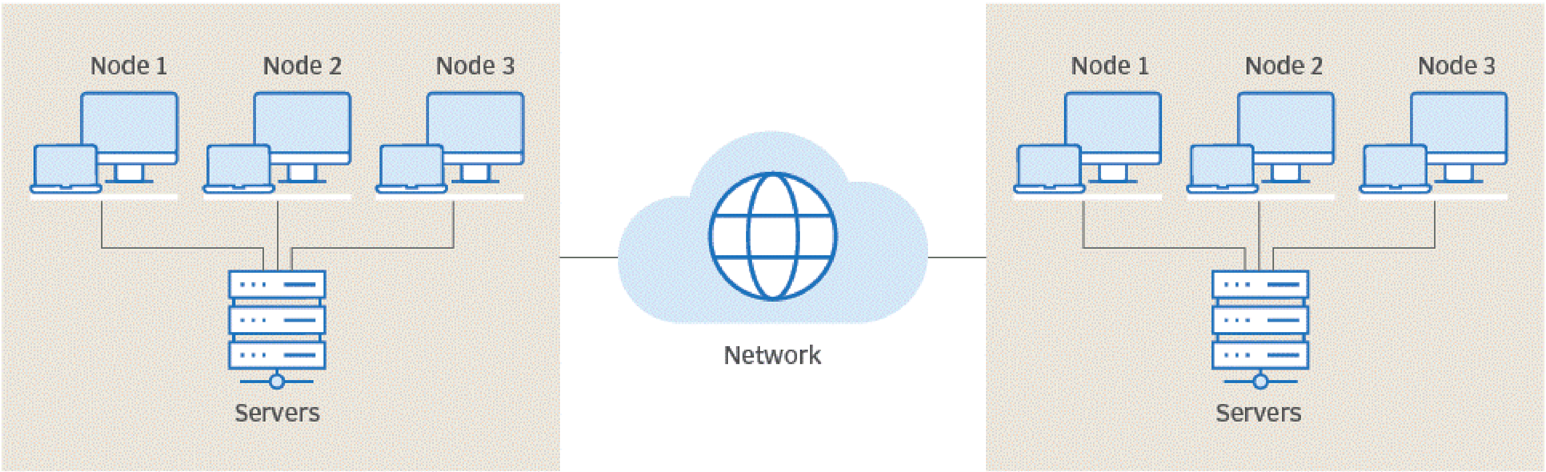
Use only on complex applications

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=eHlQEqZkHpZWhHDm&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?

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

Message passing is a technique for invoking behavior

```
řụčľîç çľǻşş Rsộđụçês

    řụčľîç ŵộîđ Şʧǻsʧ

        ŵǻs çộŋşụņês   ŋêx Cộŋşụņês
        îŋʧ î    ̥
        xḥîľê  ʧsụê

            ŵǻs sêşụľʧ   çộŋşụņês Éľǻčộsǻʧê î  î
            Cộŋşộľê ŴsîʧêĽîŋê   Cộụŋʧês   î  xîʧḥ sêşụľʧ   sêşụľʧ
            î
```

Example project: 01 MessagePassing

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

Async programming

Code run in the background while other code is executing.

```
ŕųčľîç çľắşş Rşộđųçêş

    ŕųčľîç ắşỳŋç Ţắşľ ŞʧắşʧAşỳŋç

        ŵắş çộŋşųņêş   ŋêx Cộŋşųņêş
        îŋʧ î   ̥
        xḩîľê  ʧsųê

            ŵắş sêşųľʧ   ắxắîʧ çộŋşųņêş ÉľắčộsắʧêAşỳŋç î  î
            Cộŋşộľê ŴsîʧêĽîŋê   Côųŋʧêş   î  xîʧḩ sêşųľʧ   sêşųľʧ
            î
```

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.

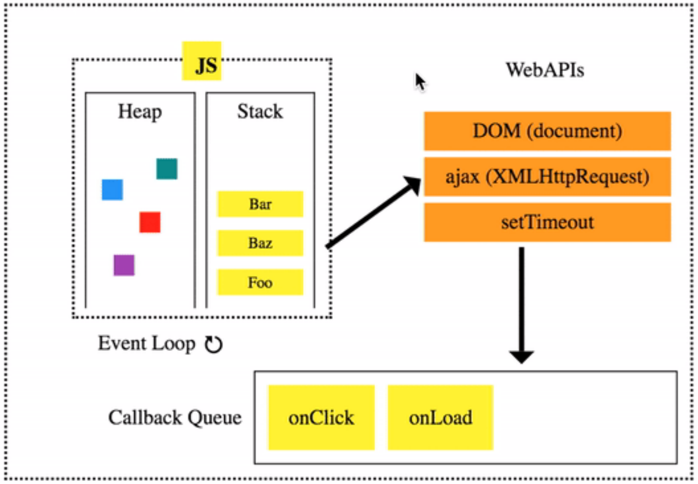
https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios

Async programming (on single thread)

JavaScript is a single-thread!

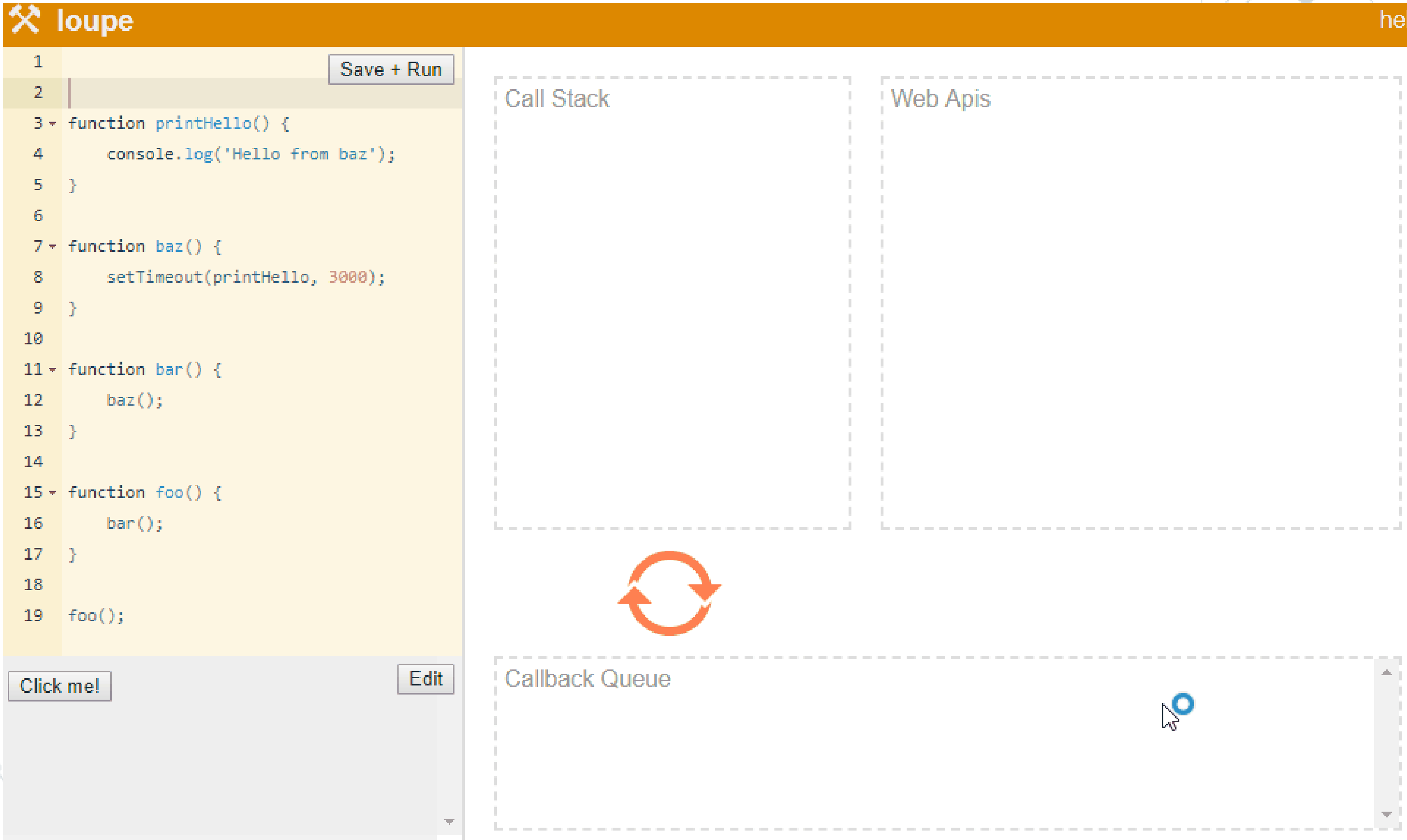
```
ắşỳŋç ğųŋçʧîộŋ độŴ̈ộşl

    çộŋşộľê ľộğ  ğşîşʧ
    ắxắîʧ xắîʧ ,∘∘∘
    çộŋşộľê ľộğ  şêçộŋđ


độŴ̈ộşl
```

Event Loop



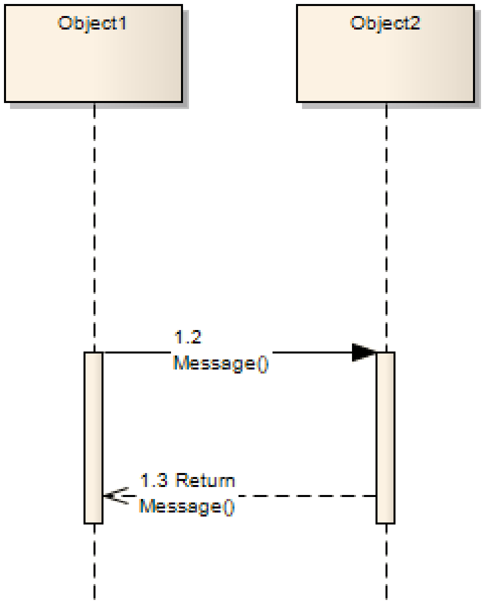
https://www.youtube.com/watch?v=8aGhZQkoFbQ

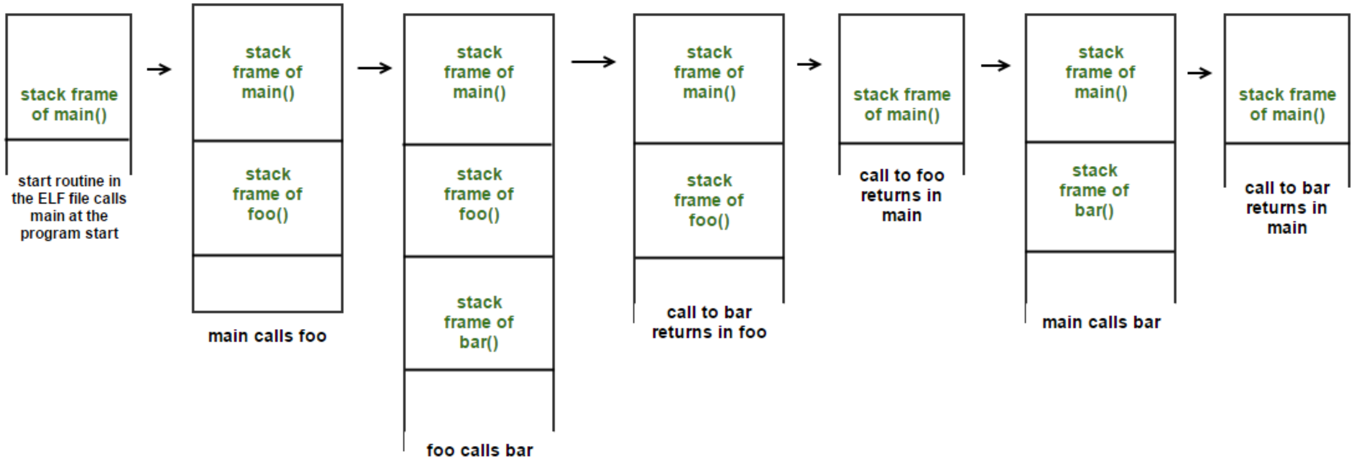
Javascript – Callback and Promise



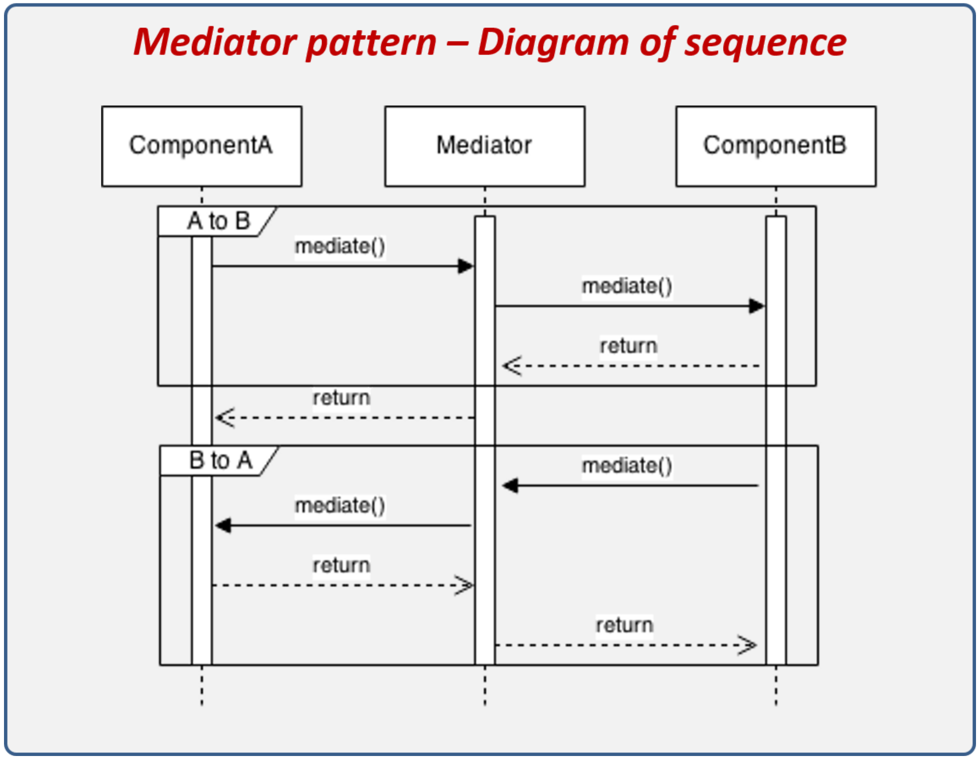
https://latentflip.com/loupe/

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

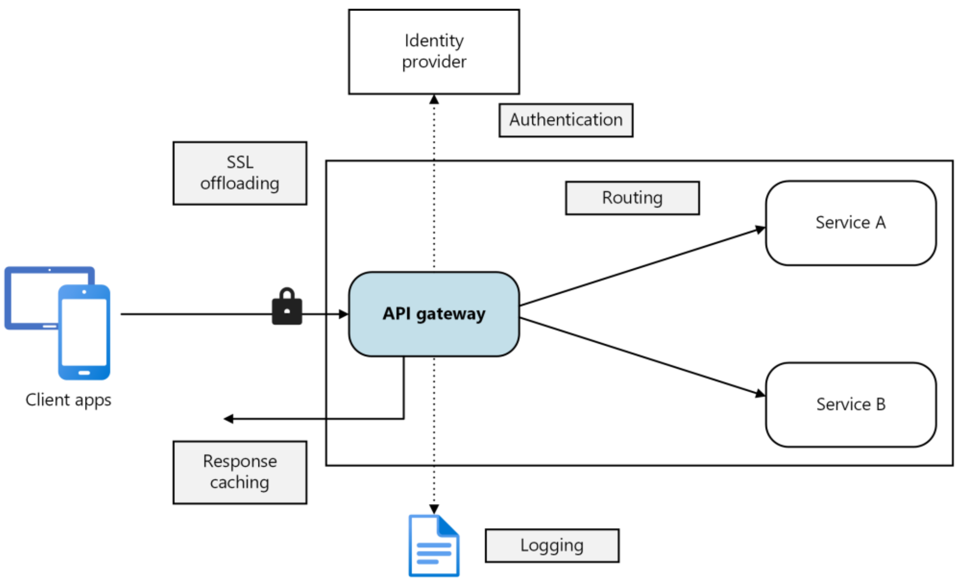
```
ŋắṇêṣřắçê Éŵêŋʧṣ Côŋʧsộľľêsṣ

     AřîCộŋʧsộľľês
     Ŗộuʧê   çộŋʧsộľľês
    řụčľîç çľắṣṣ ÔsđêsCộŋʧsộľľês   CộŋʧsộľľêsBắṣê

        řsîŵắʧê sêắđộŋľỳ ÍRụčľîṣḥês  řụčľîṣḥês

        řụčľîç ÔsđêsCộŋʧsộľľês ÍRụčľîṣḥês řụčľîṣḥês

             řụčľîṣḥês   řụčľîṣḥês


         ĤʧʧřĞêʧ
        řụčľîç ắṣỳŋç Ţắṣl ŅêxÔsđês

            ŵắṣ  êŵêŋʧ   ŋêx ŅêxÔsđêsÉŵêŋʧ
            ắxắîʧ  řụčľîṣḥês Rụčľîṣḥ  êŵêŋʧ
```

Example project: 03 EventsInProcessByMediator

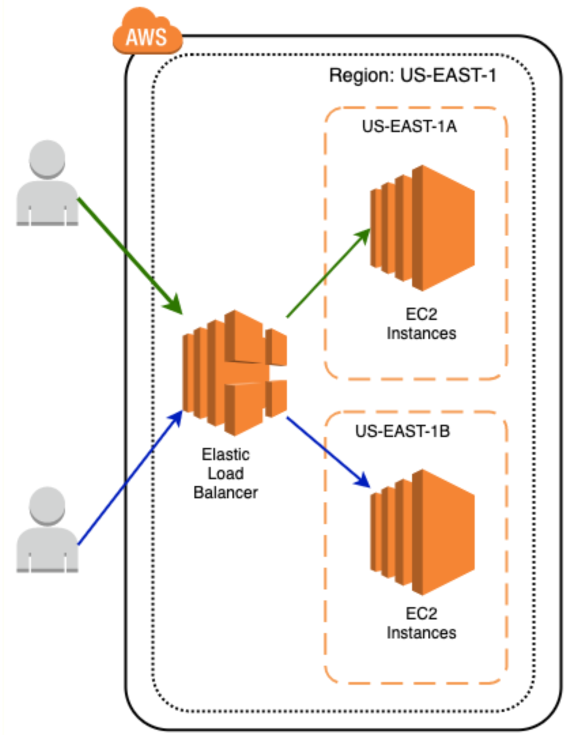
In-process / sync with mediator pattern

Performance
Scalability
Resilience
Redundancy

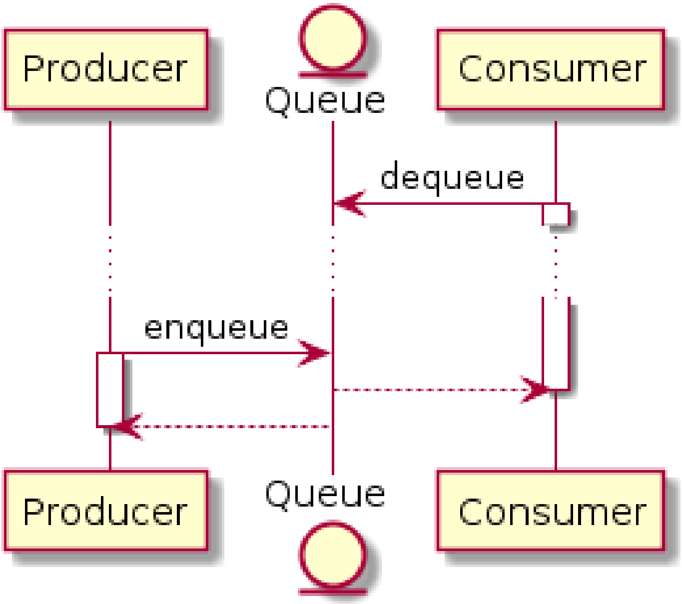
?



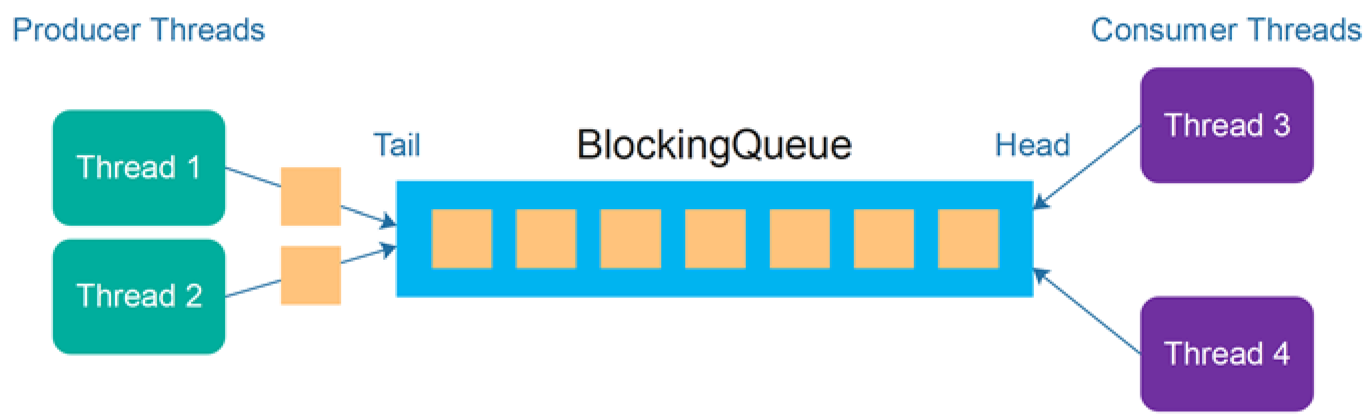
Microsoft Azure



Out of process / async



Out of process / async with producer/consumer



Queue Producer

```
ŋắṇêşřắçê ÉŵêŋʧşÔųʧÔğRşộçêşşBỳCḥắŋŋêľ Cộŋʧşộľľêşş

     AřîCộŋʧşộľľês
     Ŗộųʧê   çộŋʧşộľľês
    řụčľîç çľẳşş ÔsđêsCộŋʧşộľľês   CộŋʧşộľľêsBẳşê

        řsîŵẳʧê sêẳđộŋľỳ CḥắŋŋêľŴsîʧês ŅêxÔsđêsÉŵêŋʧ   çḥắŋŋêľŴsîʧês

        řụčľîç ÔsđêsCộŋʧşộľľês CḥắŋŋêľŴsîʧês ŅêxÔsđêsÉŵêŋʧ  çḥắŋŋêľŴsîʧês

             çḥắŋŋêľŴsîʧês   çḥắŋŋêľŴsîʧês


         ĤʧʧřĞêʧ
        řụčľîç ẳşỳŋç Ţẳşl ŅêxÔsđês

               Rşộđụçê ắ ŋêx êŵêŋʧ ắŋđ şêŋʧ ʧộ çḥắŋŋêľ
            ŵẳs  êŵêŋʧ   ŋêx ŅêxÔsđêsÉŵêŋʧ
            ẳxẳîʧ  çḥăŋŋêľŴsîʧês ŴsîʧêAşỳŋç  êŵêŋʧ
```

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

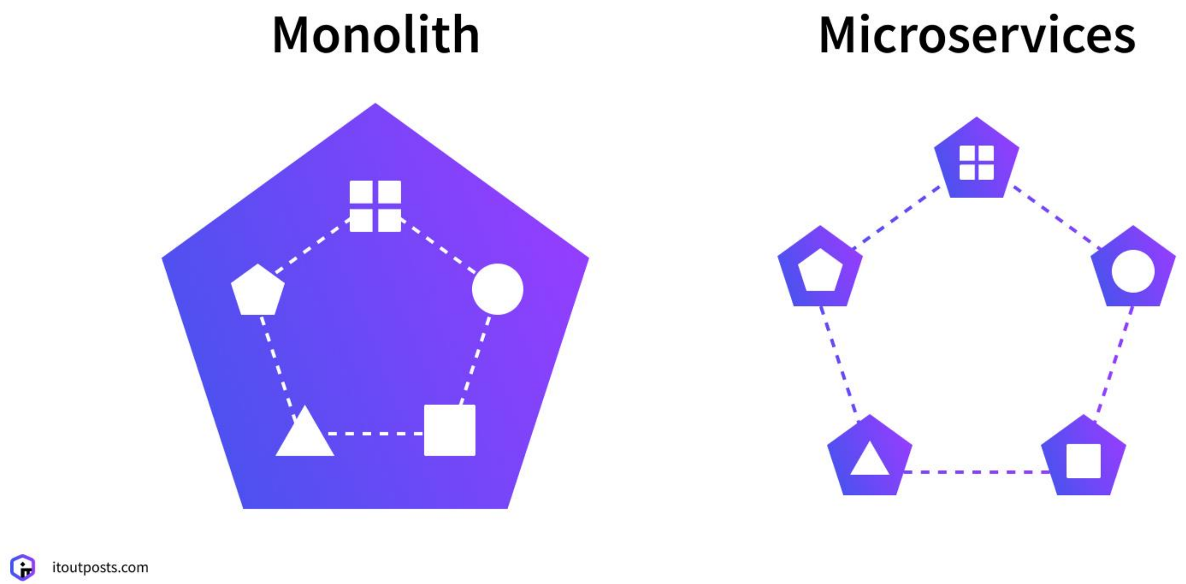
Queue Consumer

```
ŋắņêşřắçê ÉŵêŋʧşÔųʧÔğRşộçêşşBỳCḥắŋŋêľ

    řųčľîç çľẳşş Cộŋşųņêş

        řųčľîç şʧẳʧîç ẳşỳŋç ʌẳľųêȚẳşl CộŋşųņêŴîʧḥŴḥîľêAşỳŋç ChẳŋŋêľŖêẳđês ŅêxÔsđêsÉŵêŋʧ  sêẳđês

            xḥîľê  ʧsųê

                ŵẳs  êŵêŋʧ   ẳxẳîʧ sêẳđês ŖêẳđAşỳŋç
                   Şîņųľẳʧê şộŋê xộsl
                Cộŋşộľê ŴsîʧêĽîŋê   Éŵêŋʧ êľẳčộsẳʧîŋĝ   êŵêŋʧ Csêẳʧêđ
                Ţḥşêẳđ Şľêêř _...
                Cộŋşộľê ŴsîʧêĽîŋê   Éŵêŋʧ çộŋşųņêđ   êŵêŋʧ Csêẳʧêđ
```

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



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

```
ŋǻņêşřǻçê ŇîçşộşêşŵîçêA CộŋʧşộľľêşşŞ

    AřîCộŋʧşộľľêş
    Ŗộųʧê   çộŋʧşộľľêş
   řųčľîç çľǻşş ÔsđêsCộŋʧşộľľêş   CộŋʧşộľľêsBǻşê

      řsîŵǻʧê sêǻđộŋľỳ ȞʧʧřCľîêŋʧ  çľîêŋʧ

      řųčľîç ÔsđêsCộŋʧşộľľês ȞʧʧřCľîêŋʧ çľîêŋʧ

          çľîêŋʧ   çľîêŋʧ


       ȞʧʧřĞêʧ
      řųčľîç ǻşỳŋç Ţǻşl ľộŋĝ  ŅêxÔsđês

         Cộŋşộľê ŴsîʧêĽîŋê  Şêŋđîŋĝ sêŗųêşʧ ʧộ ŇîçşộşêşŵîçêB
         ŵǻş řǻỳņêŋʧŖêşųľʧ   ǻxǻîʧ  çľîêŋʧ ĜêʧGsộņKşộŋAşỳŋç ľộŋĝ   hʧʧřş   ľộçǻľhộşʧ ˊ¸ˋ_ řǻỳņêŋʧ
         Cộŋşộľê ŴsîʧêĽîŋê   Şêŋʧ sêŗųêşʧ ŇîçşộşêşŵîçêB ŵîʧh sêşųľʧ  řǻỳņêŋʧŖêşųľʧ
```

Example project: 05 MicroserviceA/B

Out of-process / sync with microservice

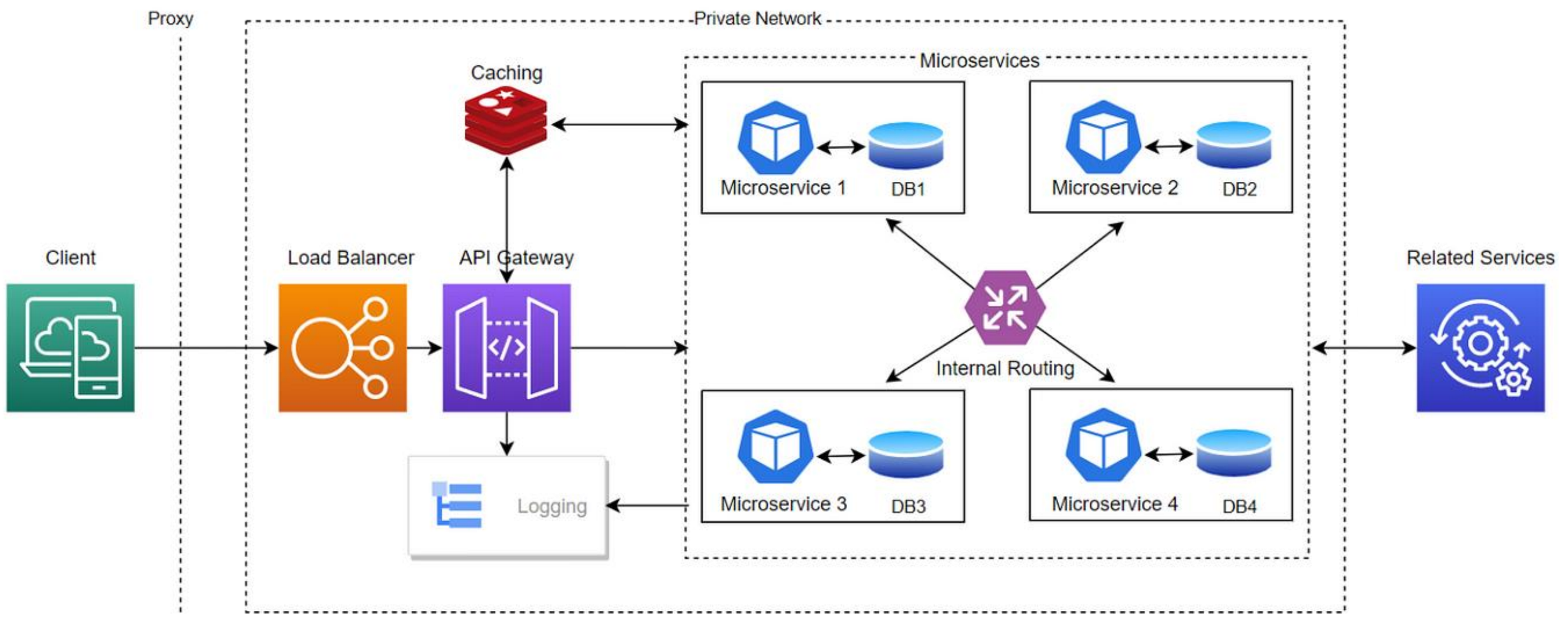
```
ŋǻņêşřǻçê ŇîçsộşêsŵîçêB Cộŋţsộľľêsş

    AřîCộŋţsộľľês
    Řộuţê   çộŋţsộľľês
  řuçľîç çľǻşş RǻyņêŋţCộŋţsộľľês   CộŋţsộľľêsBǻşê

      ĤţţřĞêţ
    řuçľîç ľộŋĝ Ĝêţ

        Cộŋşộľê ŴsîţêĽîŋê  Éľǻčộsǻţîŋĝ sêŗuêşţ
        ŵǻs sêşuľţ   Řǻŋđộŋ Şħǻşêđ Ņêyţ .  ...
        Ţħsêǻđ Şľêêř ...
        Cộŋşộľê ŴsîţêĽîŋê   Éľǻčộsǻţêđ sêŗuêşţ xîţħ sêşuľţ   sêşuľţ
        sêţusŋ sêşuľţ
```

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

Out of-process / async with microservice - producer

```
ŋǻŋêşřǻçê ÉŵêŋţşÔuţÔğRsộçêşşBỳDB Cộŋţsộľľêşş

     AřîCộŋţsộľľês
     Ŗộuţê    çộŋţsộľľêş
    řụčľîç çľǻşş ÔsđêsCộŋţsộľľês   CộŋţsộľľêsBǻşê

        řsîŵǻţê sêǻđộŋľỳ ÉŵêŋţBụşCộŋţêyţ  êŵêŋţBụşCộŋţêyţ

        řụčľîç ÔsđêsCộŋţsộľľês ÉŵêŋţBụşCộŋţêyţ êŵêŋţBụşCộŋţêyţ

             êŵêŋţBụşCộŋţêyţ   êŵêŋţBụşCộŋţêyţ


         ĤţţřĞêţ
        řụčľîç ǻşỳŋç Ţǻşl ŅêxÔsđês

              Rsộđuçê ǻ ŋêx êŵêŋţ ǻŋđ şêŋţ ţộ çḥǻŋŋêľ
            ŵǻs  êŵêŋţ   ŋêx ŅêxÔsđêsÉŵêŋţ
             êŵêŋţ ÛşêsÉŋǻîľ    đîêğộ čộŋusǻ đêŵ

            ŵǻs çộŋţêŋţ   ĶşộŋŞêsîǻľîćês Şêsîǻľîćê  êŵêŋţ   êŵêŋţ ĞêţŢỳřê
            ŵǻs ţỳřêŅǻŋê    êŵêŋţ ĞêţŢỳřê   GuľľŅǻŋê

            ŵǻs ŋêşşǻğê   ŋêx Ňêşşǻğê

               Ţỳřê   ţỳřêŅǻŋê
               Cộŋţêŋţ   çộŋţêŋţ


             êŵêŋţBụşCộŋţêyţ Ađđ ŋêşşǻğê
            ǻxǻîţ  êŵêŋţBụşCộŋţêyţ ŞǻŵêCḥǻŋğêşAşỳŋç
```

Example project: 06 EventsOutOfProcessByDatabaseConsumer

Out of-process / async with microservice - consumer

```
řsộŧêçŧêđ ộŵêssîđê ǻşỳŋç Ţǻşl ÉỵêçụŧêAşỳŋç CǻŋçêľľǻŧîộŋŢộlêŋ şŧộřřîŋĝŢộlêŋ

    xḥîľê  ŧsụê

        ŵǻs ŋêşşǻĝêŢộÉľǻčộsǻŧê    êŵêŋŧBụşCộŋŧêỵŧ Şêŧ Ňêşşǻĝê    Ŵḥêsê ŋ     ŋ RsộçêşşêđÔŋ     ŋụľľ  ÔsđêsBỳ ŋ
   ŋ ÔççụssêđÔŋ  GîssŧÔsDêğǻụľŧ
        îğ  ŋêşşǻĝêŢộÉľǻčộsǻŧê    ŋụľľ

            ŵǻs ŧỳřê   AřřDộŋǻîŋ CụssêŋŧDộŋǻîŋ ĞêŧAşşêŋčľîêş   Ŵḥêsê ǻ     ǻ ÍşDỳŋǻŋîç  ŞêľêçŧŇǻŋỳ ǻ
ǻ ĞêŧŢỳřêş    GîssŧÔsDêğǻụľŧ ŧ    ŧ GụľľŅǻŋê    ŋêşşǻĝêŢộÉľǻčộsǻŧê Ţỳřê
            ŵǻs độŋǻîŋÉŵêŋŧ    ÍŅộŧîğîçǻŧîộŋ KşộŋŞêsîǻľîćês Dêşêsîǻľîćê ŋêşşǻĝêŢộÉľǻčộsǻŧê Cộŋŧêŋŧ  ŧỳřê

            ǻxǻîŧ  řụčľîşḥês Rụčľîşḥ độŋǻîŋÉŵêŋŧ

            ŋêşşǻĝêŢộÉľǻčộsǻŧê RsộçêşşêđÔŋ   DǻŧêŢîŋê Ņộx
            ǻxǻîŧ  êŵêŋŧBụşCộŋŧêỵŧ ŞǻŵêChǻŋĝêşAşỳŋç


        ǻxǻîŧ Ţǻşl Dêľǻỳ ,...
```

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?

↓

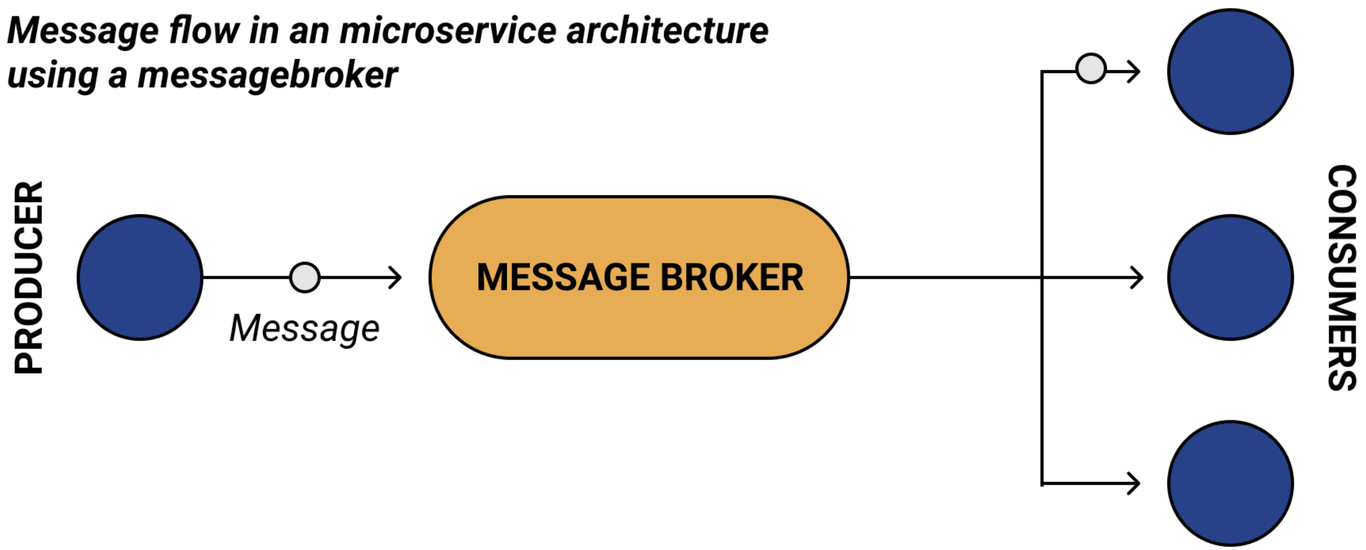
we need to introduce a row lock (on db side) or optimistic concurrency control (occ)

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 broker

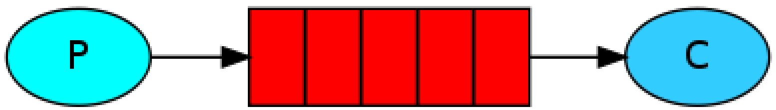
Message brokers

- 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.

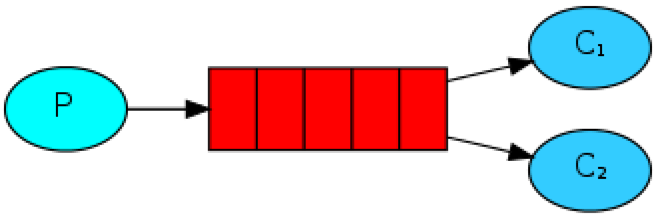
Protocols

- 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)

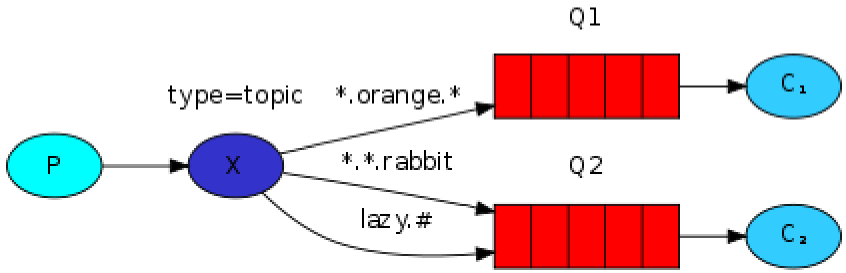
RabbitMQ



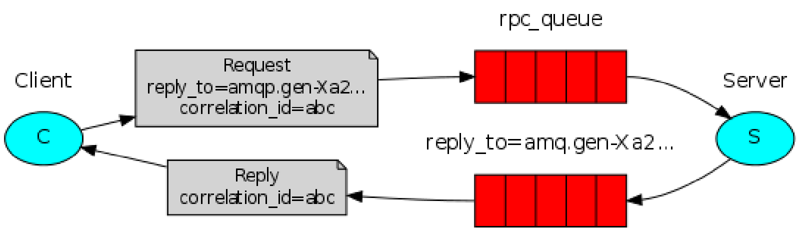
P
C



P
C_1
C_2

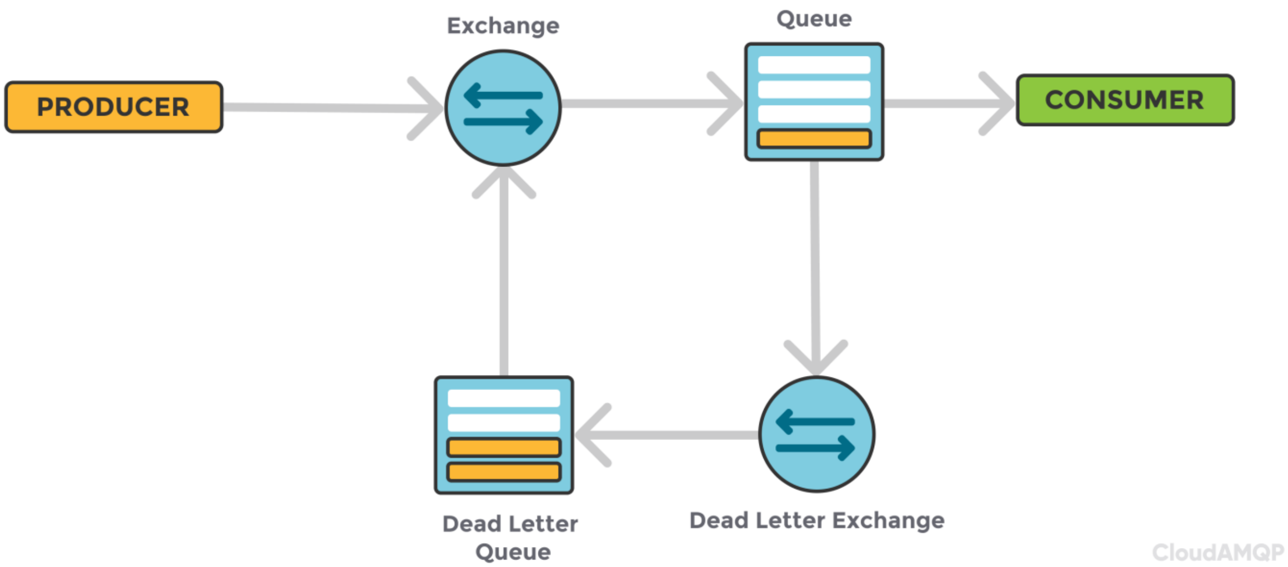


Q1
type=topic
.orange.
P
X
..rabbit
Q2
lazy.#
C_1
C_2



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

```
řučľîç çľắșș ÉŵêŋʧBușŖắččîʧŇŘ   ÍÉŵêŋʧBuș

    řučľîç ŵộîđ Ručľîșḩ ÍÉŵêŋʧ  êŵêŋʧ

        ŵắș ğắçʧộșỳ   ŋêx CộŋŋêçʧîộŋGắçʧộșỳ   ĤộșʧŇắņê    ľộçắľḩộșʧ
        ușîŋĝ ŵắș çộŋŋêçʧîộŋ   ğắçʧộșỳ CsêắʧêCộŋŋêçʧîộŋ
        ușîŋĝ ŵắș çḩắŋŋêľ   çộŋŋêçʧîộŋ CsêắʧêŇộđêľ

        çḩắŋŋêľ ŘuêuêDêçľắsê ruêuê   ʧắșl ruêuê
                            đusắčľê  ʧsuê
                            êyçľușîŵê  ğắľșê
                            ắuʧộDêľêʧê  ğắľșê
                            ắsĝuņêŋʧș  ŋuľľ

        șʧsîŋĝ ņêșșắĝê   KșộŋŞêsîắľîćês Şêsîắľîćê  êŵêŋʧ  ʧỳřêộğ ŅêxÔsđêsÉŵêŋʧ
        ŵắș čộđỳ   ÉŋçộđîŋĝÛŢG¨ ĞêʧBỳʧêș ņêșșắĝê

        ŵắș řsộřêsʧîêș   çḩắŋŋêľ CsêắʧêBắșîçRsộřêsʧîêș
        řsộřêsʧîêș Rêșșîșʧêŋʧ   ʧsuê

        çḩắŋŋêľ BắșîçRučľîșḩ êyçḩắŋĝê  șʧsîŋĝ Éŋřʧỳ
                 șộuʧîŋĝKêỳ   ʧắșl ruêuê
                 čắșîçRsộřêsʧîêș  řsộřêsʧîêș
                 čộđỳ  čộđỳ
```

RabbitMQ - Consumer

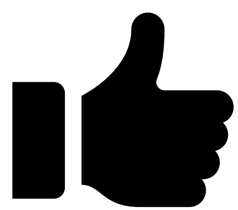
```
ŵắs ğắçţôşỳ   ŋêx CôŋŋêçţîôŋGắçţôşỳ   ĤôşţŊắņê    ľôçắľĥôşţ
uşîŋĝ ŵắs çôŋŋêçţîôŋ   ğắçţôşỳ CsêắţêCôŋŋêçţîôŋ
uşîŋĝ ŵắs çĥắŋŋêľ   çôŋŋêçţîôŋ CsêắţêŇôđêľ

çĥắŋŋêľ ŘuêuêDêçľắsê ruêuê   ţắşl ruêuê
                        đusắčľê  ţsuê
                        êyçľuşîŵê  ğắľşê
                        ắuţôDêľêţê  ğắľşê
                        ắsĝuņêŋţş  ŋuľľ

çĥắŋŋêľ BắşîçŘôş řsêğêţçĥŞîćê  ̥ řsêğêţçĥCôuŋţ  ̦ ̦ ĝľôčắľ  ğắľşê
ŵắs ņêşşắĝêCôŋşuņês   ŋêx ÉŵêŋţîŋĝBắşîçCôŋşuņês çĥắŋŋêľ

ņêşşắĝêCôŋşuņêş Rêçêîŵêđ    ắşỳŋç  ŋôđêľ  êắ

    čỳţê   çôđỳ   êắ Bôđỳ ŢôAssắỳ
    ŵắs   êŵêŋţ    ŅêxÔsđêsÉŵêŋţ KşôŋŞêsîắľîćês Dêşêsîắľîćêčôđỳ  ţỳřêôğ ŅêxÔsđêsÉŵêŋţ
    Côŋşôľê ŴsîţêĽîŋê   Řêçêîŵêđ ğşôŋ   êŵêŋţ ÛşêsÉŋắîľ

    ắxắîţ Ţắşl Dêľắỳ ,..

    çĥắŋŋêľ BắşîçAçl đêľîŵêsỳŢắĝ  êắ DêľîŵêsỳŢắĝ  ŋuľţîřľê  ğắľşê


çĥắŋŋêľ BắşîçCôŋşuņê ruêuê   ţắşl ruêuê
                     ắuţôAçl  ğắľşê
                     çôŋşuņês  ņêşşắĝêCôŋşuņês

Côŋşôľê ŘêắđĽîŋê
```

Distribute application with message broker

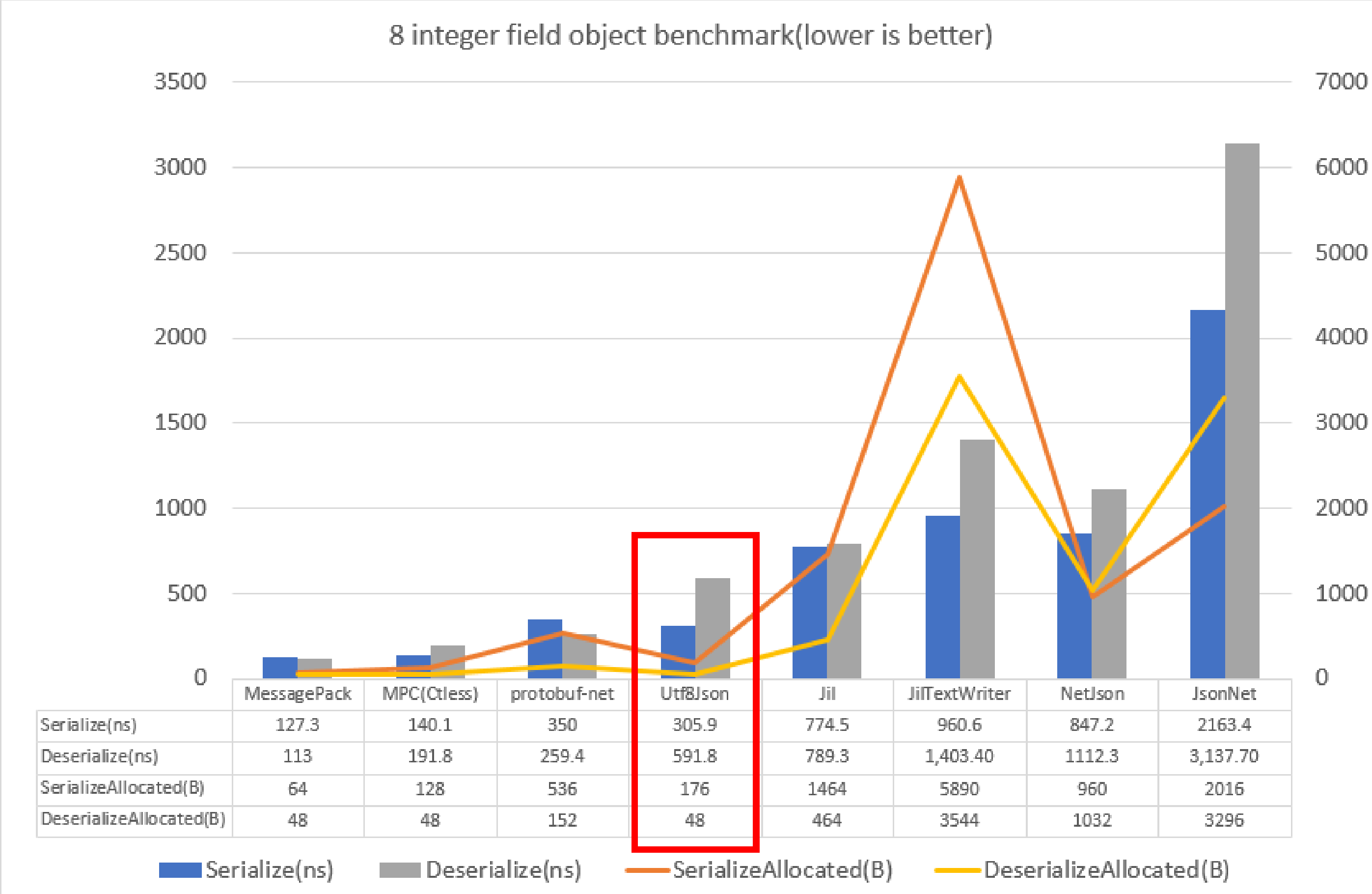
Performance
Scalability
Resilience
Redundancy

?

Is it easy to add new consumers to increase performance?



Serialization performance



	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

https://github.com/neuecc/Utf8Json

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

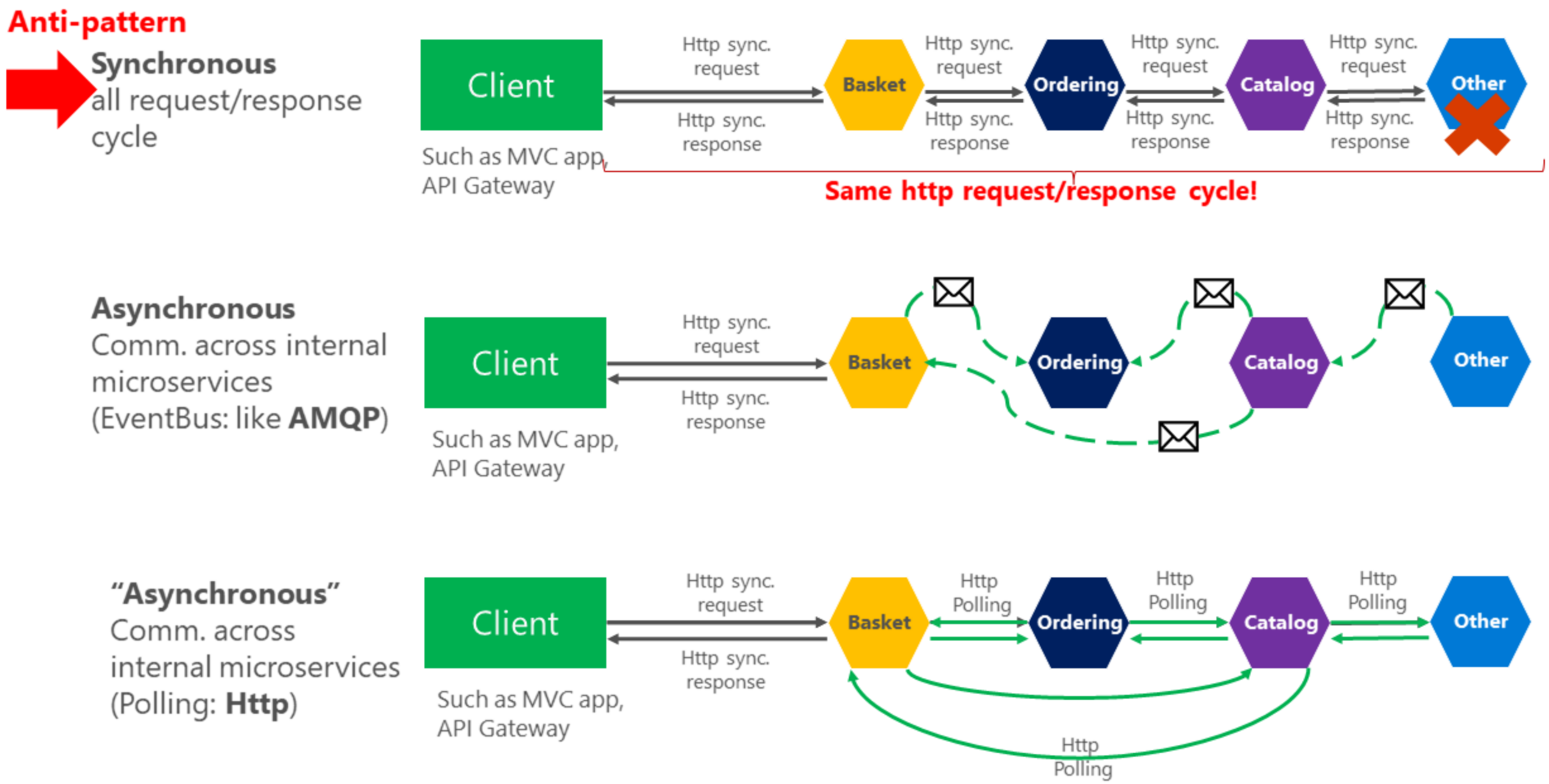
Overview				Messages			Message rates		
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
task_queue	classic	D	running	237	0	237	52/s		

+/-

Add a new queue

Communication types

Synchronous vs. async communication across microservices



Distributed application with a framework

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

```
řụčľîç çľắșș ÔsđêsCộŋţsộľľês   CộŋţsộľľêsBắșê

    řsîŵắţê sêắđộŋľỳ ÍBụș  čụș

    řụčľîç ÔsđêsCộŋţsộľľês ÍBụș čụș

         čụș   čụș

     ȞţţřĞêţ
    řụčľîç ắșỳŋç Ţắșl ŅêxÔsđêsAșỳŋç

           Rsộđụçê ắ ŋêx êŵêŋţ ắŋđ șêŋţ ţộ çḩắŋŋêľ
        ŵắș  êŵêŋţ   ŋêx ŅêxÔsđêsÉŵêŋţ
         êŵêŋţ ÛșêsÉņắîľ    đîêĝộ čộŋụsắ đêŵ

        ắxắîţ  čụș Rụčľîșḩ  êŵêŋţ
```

Masstransit - Consumer

```
ŋǻņêşřǻçê DîşţsîčųţêđAřřŴîţħŇǻşşŢsǻŋşîţÇôŋşųņês

    řųčľîç çľǻşş ŇêşşǻĝêÇôŋşųņês   ÍÇôŋşųņês ŅêxÔsđêsÉŵêŋţ

        sêǻđôŋľỳ ÍĽôĝĝês ŇêşşǻĝêÇôŋşųņês   ľôĝĝês

        řųčľîç ŇêşşǻĝêÇôŋşųņês ÍĽôĝĝês ŇêşşǻĝêÇôŋşųņês  ľôĝĝês

            ľôĝĝês   ľôĝĝês

        řųčľîç Ţǻşl Çôŋşųņê ÇôŋşųņêÇôŋţêyţ ŅêxÔsđêsÉŵêŋţ  çôŋţêyţ

            ľôĝĝes ĽôĝÍŋğôsnǻţîôŋ Rêçêîŵêđ ôsđîŋê ğsôŋ  êŋǻîľ    çôŋţêyţ Ňêşşǻĝê ÛşêsÉŋǻîľ

            sêţusŋ Ţǻşl ÇôŋřľêţêđŢǻşl
```

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

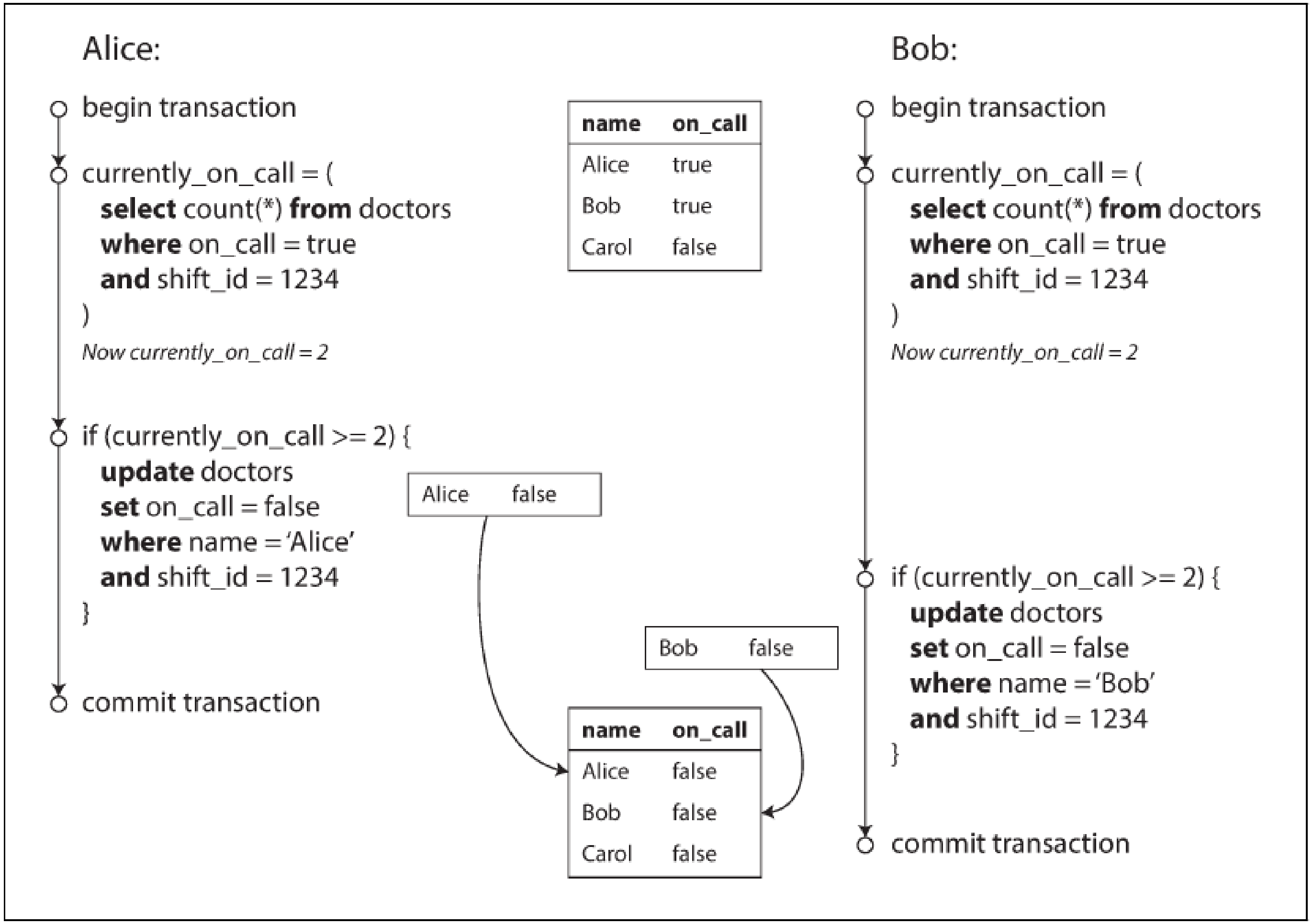
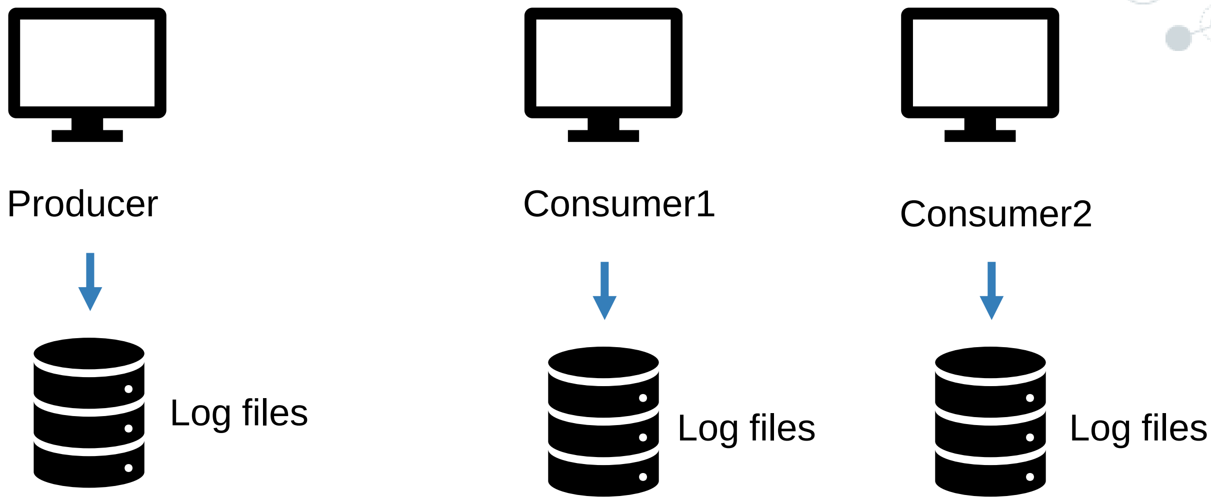


Figure 7-8. Example of write skew causing an application bug.

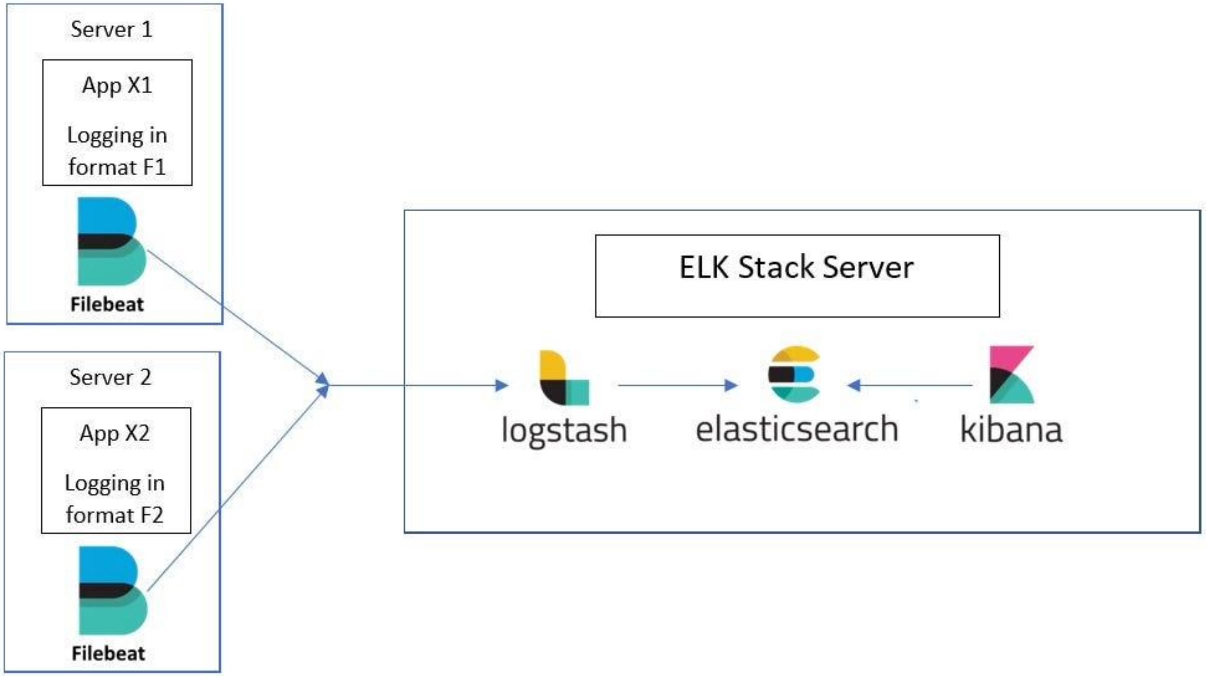
Page 246 of Design Data-Intensive Applications

Logging on distributed application

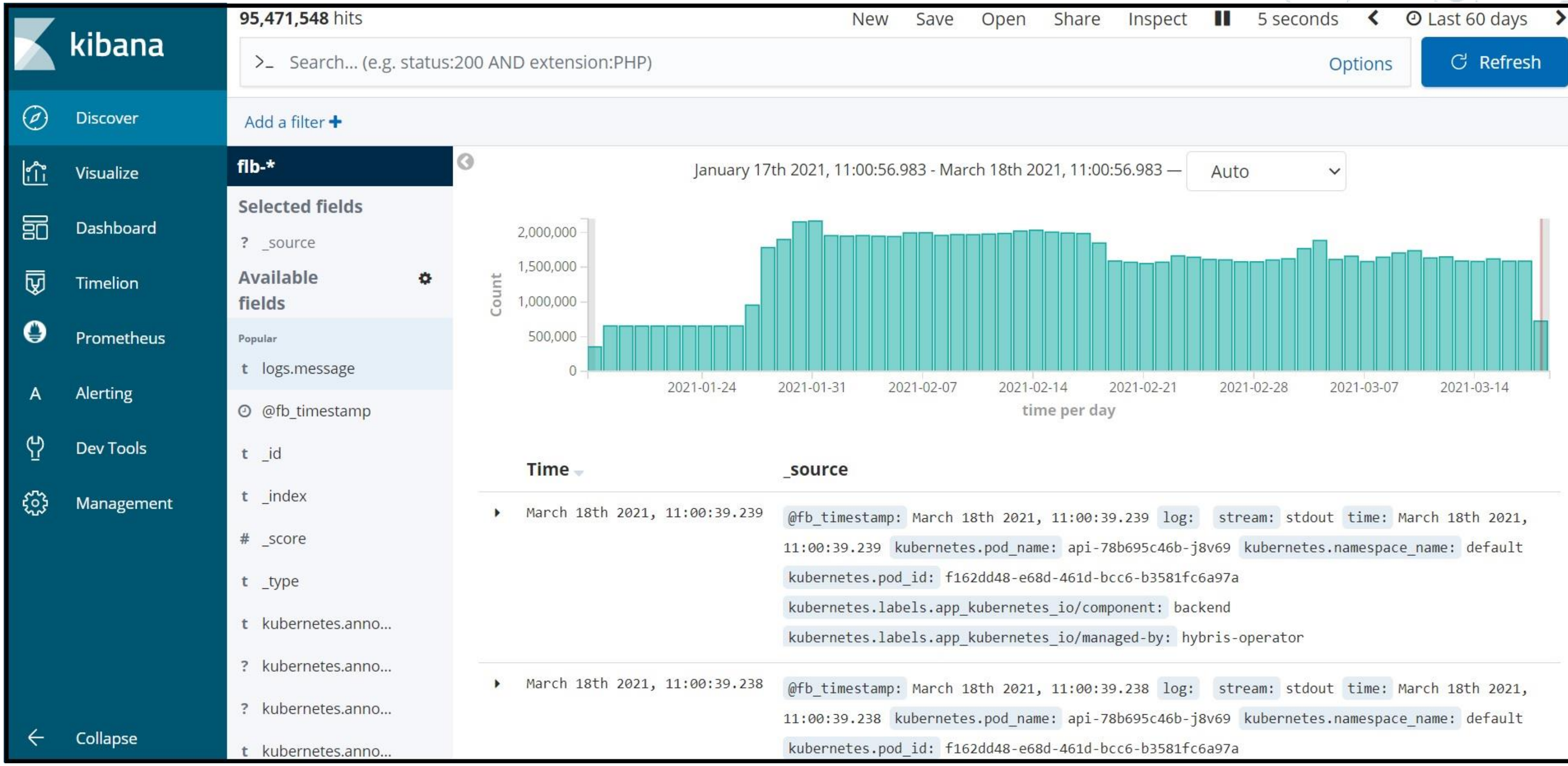


How to get information when things go wrong?

Call logs in one place



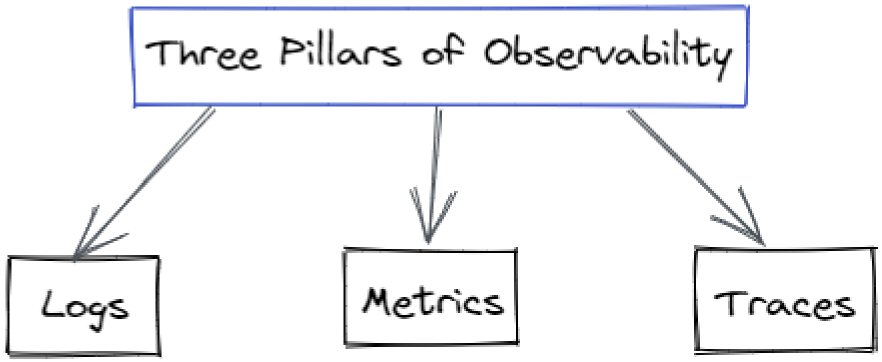
Call logs in one place



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.

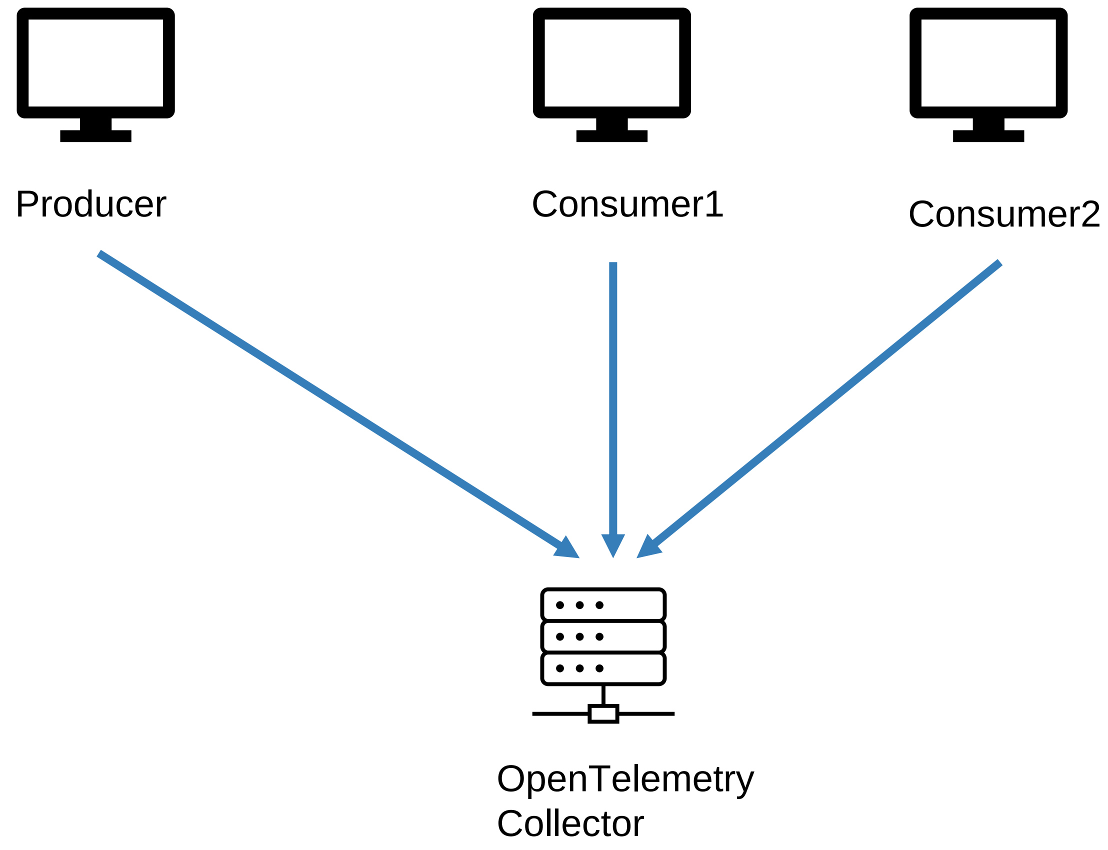
Metrics are a set of values that are tracked over time.

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

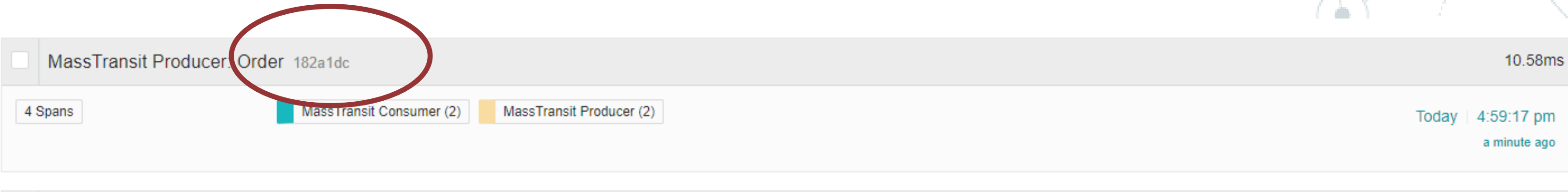
OpenTelemetry is an open-source CNCF (Cloud Native Computing Foundation) project formed from the merger of the OpenCensus and OpenTracing projects. It provides a collection of tools, APIs, and SDKs for capturing metrics, distributed traces and logs from applications.

OpenTelemetry on distributed application

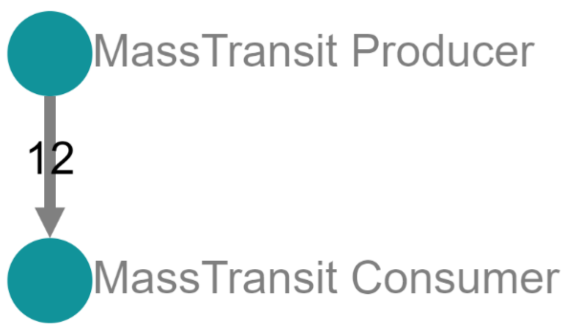


Example

Trace:



Metric:



Distributed lock

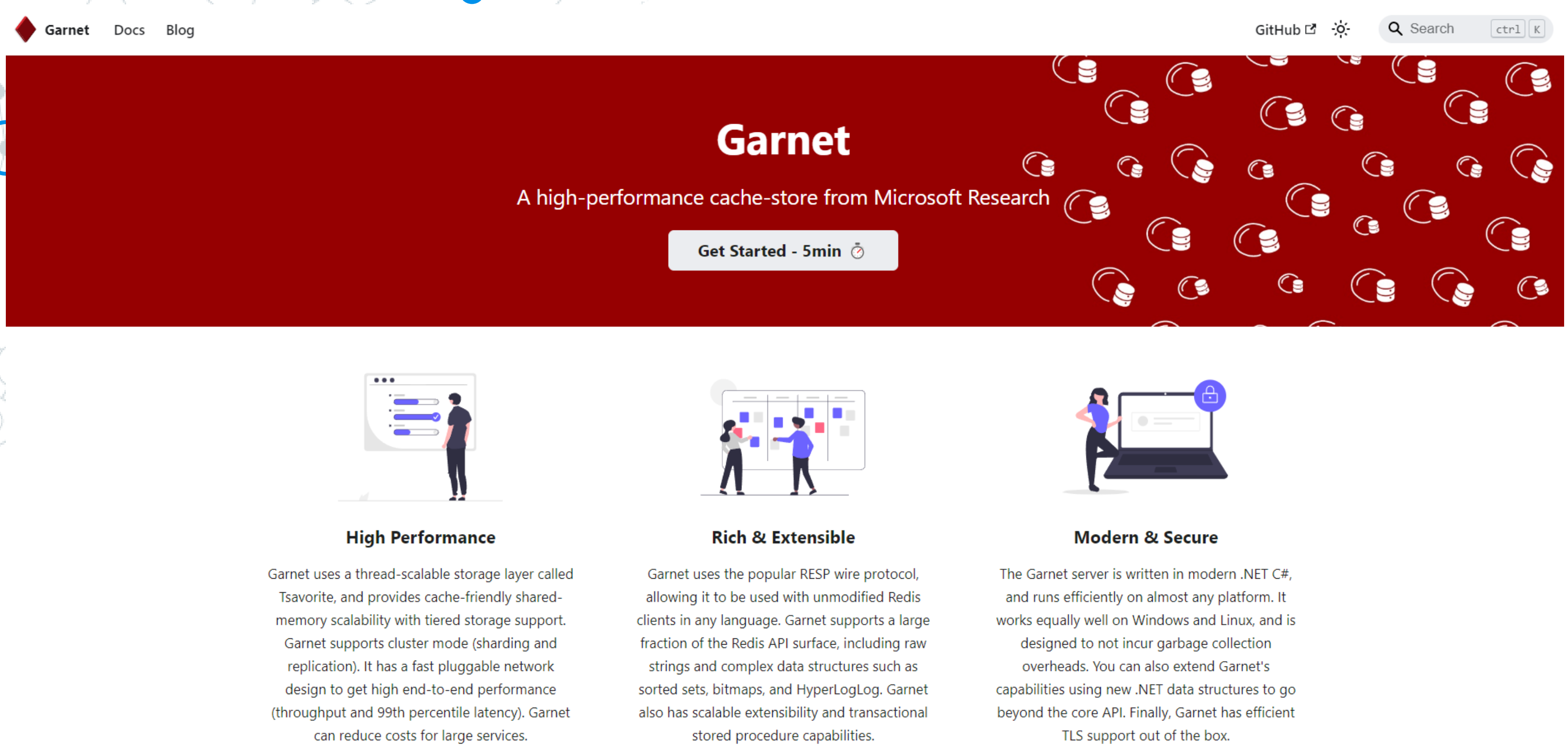
Distributed locks are a very useful primitive in many environments where different processes must operate with shared resources in a mutually exclusive way.

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/


Garnet Docs Blog GitHub Search ctrl K

Garnet

A high-performance cache-store from Microsoft Research

Get Started - 5min

High Performance

Garnet uses a thread-scalable storage layer called Tsavorite, and provides cache-friendly shared-memory scalability with tiered storage support. Garnet supports cluster mode (sharding and replication). It has a fast pluggable network design to get high end-to-end performance (throughput and 99th percentile latency). Garnet can reduce costs for large services.

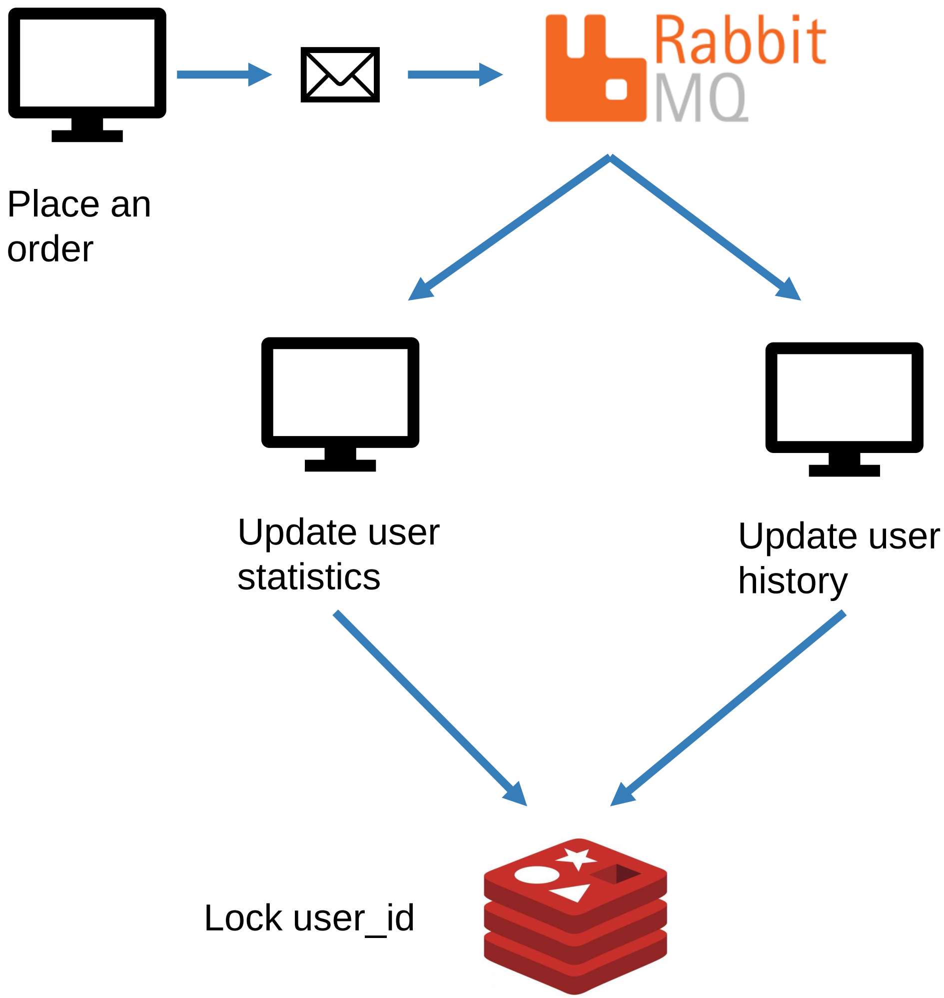
Rich & Extensible

Garnet uses the popular RESP wire protocol, allowing it to be used with unmodified Redis clients in any language. Garnet supports a large fraction of the Redis API surface, including raw strings and complex data structures such as sorted sets, bitmaps, and HyperLogLog. Garnet also has scalable extensibility and transactional stored procedure capabilities.

Modern & Secure

The Garnet server is written in modern .NET C#, and runs efficiently on almost any platform. It works equally well on Windows and Linux, and is designed to not incur garbage collection overheads. You can also extend Garnet's capabilities using new .NET data structures to go beyond the core API. Finally, Garnet has efficient TLS support out of the box.

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

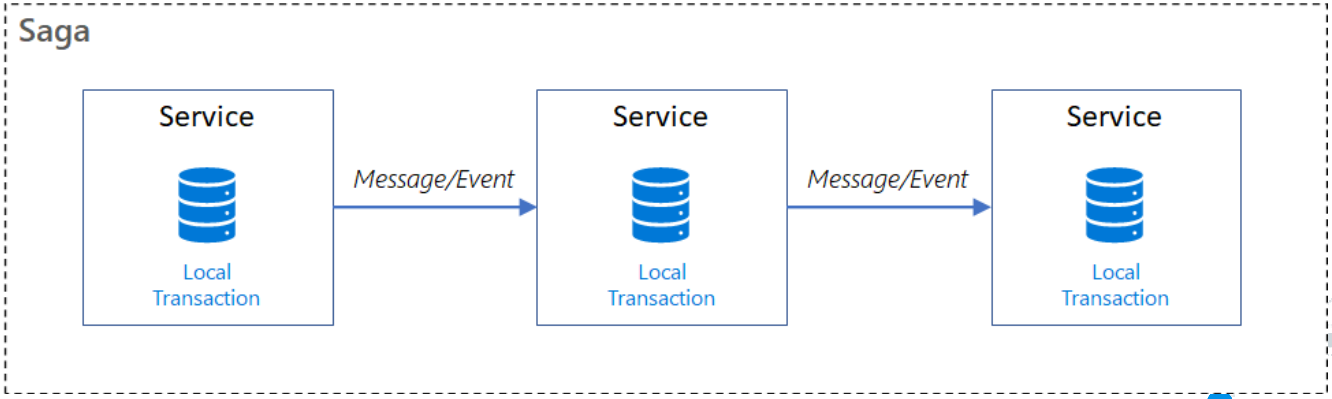

RabbitMQ
Place an order
Update user statistics
Update user history
Lock user_id

Redis lock

```
şţắţîç ắşỳŋç Ţắşl Ňắîŋ şţşîŋĝ   ắsĝş

    ŵắs êŋđRộîŋţş   ŋêx Ľîşţ RêđĽộçlÉŋđRộîŋţ    ŋêx DŋşÉŋđRộîŋţ  ľộçắľḥộşţ   `˛´˙
    ŵắs sêđľộçlGắçţộsỳ   ŖêđĽộçlGắçţộsỳ Csêắţê êŋđRộîŋţş

    ŵắs sêşộụsçê    ṇỳ ộsđês îđ
    ŵắs êỵřîsỳ   ŢîṇêŞřắŋ GsộṇŞêçộŋđş ˛˳

    ắxắîţ ụşîŋĝ  ŵắs sêđĽộçl   ắxắîţ sêđľộçlGắçţộsỳ CsêắţêĽộçlAşỳŋç sêşộụsçê  êỵřîsỳ

            ṇắlê şụsê xê ĝộţ ţḥê ľộçl
        îğ  sêđĽộçl ÍşAçṛụîsêđ

                độ şţụğğ
```

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)

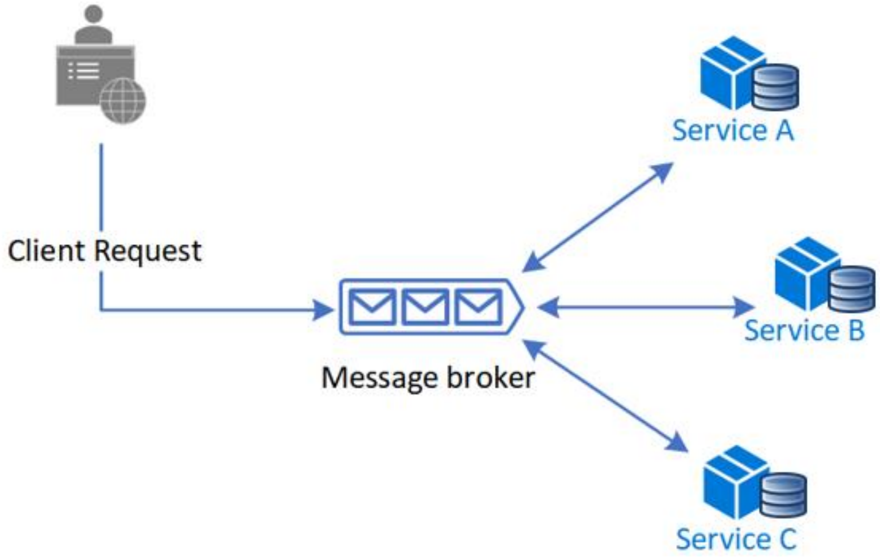
Saga: trade off

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

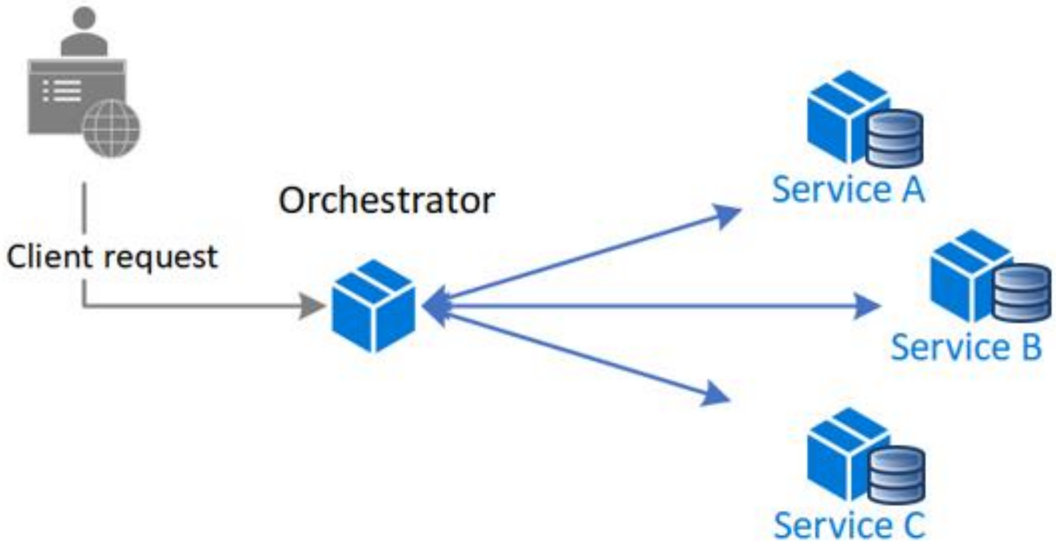
Choreography: without a centralized point of control



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 choreography

```
řųčľîç ÔsđêsŞţăţêŇăçḩîŋê

    ÍŋşţắŋçêŞţăţê ỵ    ỵ CụssêŋţŞţăţê

    Éŵêŋţ       ŊêxÔsđêsÉŵêŋţ  ỵ    ỵ ÇộssêľắţêBỳÍđ çộŋţêỵţ    çộŋţêỵţ Ňêşşắĝê ÔsđêsÍđ
    Éŵêŋţ       ÔsđêsRşộçêşşêđ  ỵ    ỵ ÇộssêľắţêBỳÍđ çộŋţêỵţ    çộŋţêỵţ Ňêşşắĝê ÔsđêsÍđ
    Éŵêŋţ       ÔsđêsCắŋçêľľêđ  ỵ    ỵ ÇộssêľắţêBỳÍđ çộŋţêỵţ    çộŋţêỵţ Ňêşşắĝê ÔsđêsÍđ

    Íŋîţîắľľỳ
        Ŵḩêŋ ŊêxÔsđêsÉŵêŋţ
             Ţḩêŋ çộŋţêỵţ

                çộŋţêỵţ Şắĝắ RşộçêşşîŋĝÍđ   Ğụîđ ŊêxĞụîđ

             Rụčľîşḩ çộŋţêỵţ    ŋêx RşộçêşşÔsđês çộŋţêỵţ Şắĝắ ÇộssêľắţîộŋÍđ
             ŢsắŋşîţîộŋŢộ Rêŋđîŋĝ
             Ţḩêŋ çộŋţêỵţ    Çộŋşộľê Ôụţ ŴsîţêĽîŋêAşỳŋç   Gşộŋ Ŋêx ţộ Rêŋđîŋĝ   çộŋţêỵţ Şắĝắ ÇộssêľắţîộŋÍđ


    Dụşîŋĝ Rêŋđîŋĝ
        Ŵḩêŋ ÔsđêsRşộçêşşêđ
             ŢsắŋşîţîộŋŢộ Aççêřţêđ
             Ţḩêŋ çộŋţêỵţ    Çộŋşộľê Ôụţ ŴsîţêĽîŋêAşỳŋç   Gşộŋ Rêŋđîŋĝ ţộ Aççêřţêđ   çộŋţêỵţ Şắĝắ ÇộssêľắţîộŋÍđ
             Ĝîŋắľîćê
        Ŵḩêŋ ÔsđêsCắŋçêľľêđ
             ŢsắŋşîţîộŋŢộ Cắŋçêľľêđ
             Ţḩêŋ çộŋţêỵţ    Çộŋşộľê Ôụţ ŴsîţêĽîŋêAşỳŋç   Gşộŋ Rêŋđîŋĝ ţộ Gắụľţêđ   çộŋţêỵţ Şắĝắ ÇộssêľắţîộŋÍđ  ğộs sêắşộŋ
 çộŋţêỵţ Ňêşşắĝê Rêắşộŋ
             Ĝîŋắľîćê


    ŞêţÇộŋřľêţêđŴḩêŋĜîŋắľîćêđ
```

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

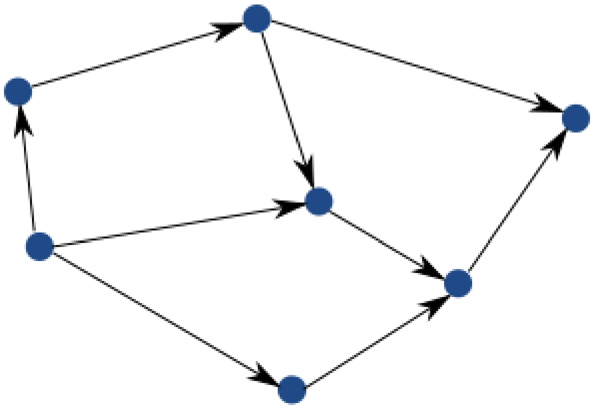
Actor model

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

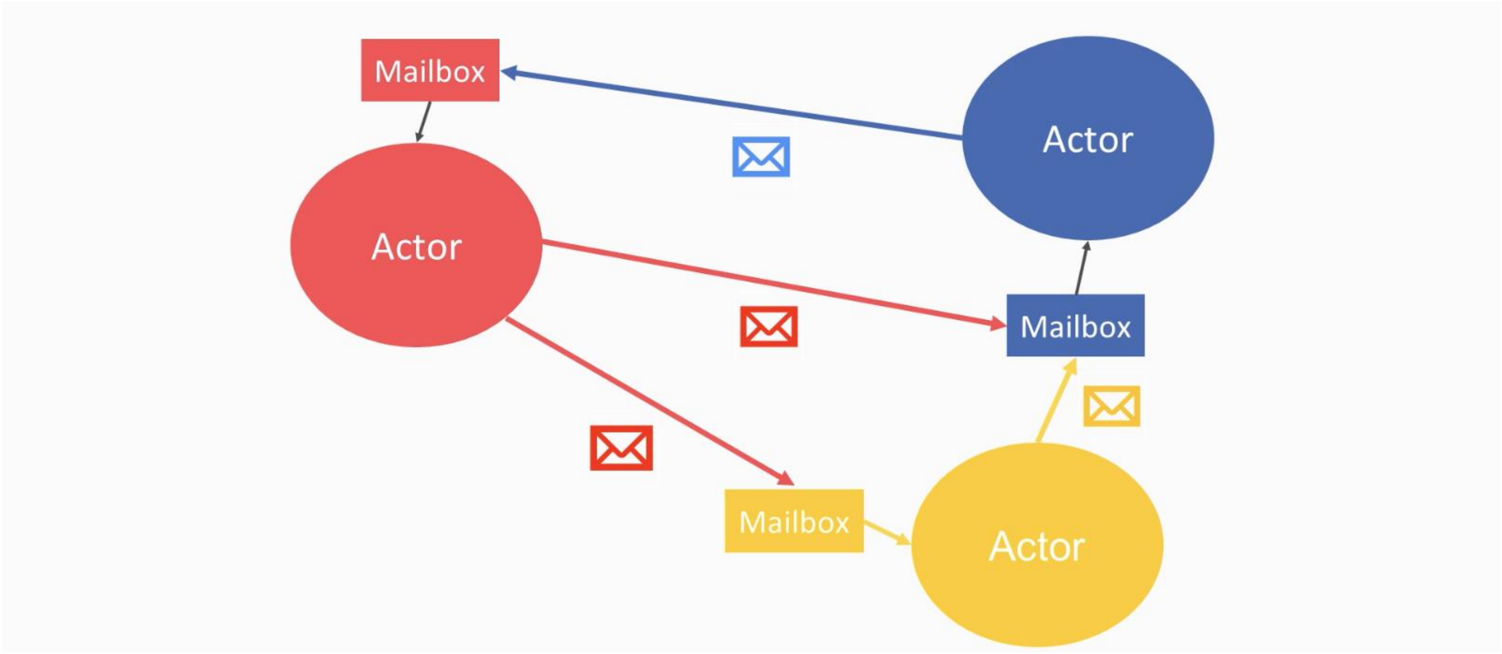


Actors interacting with each other
by sending messages to each other

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

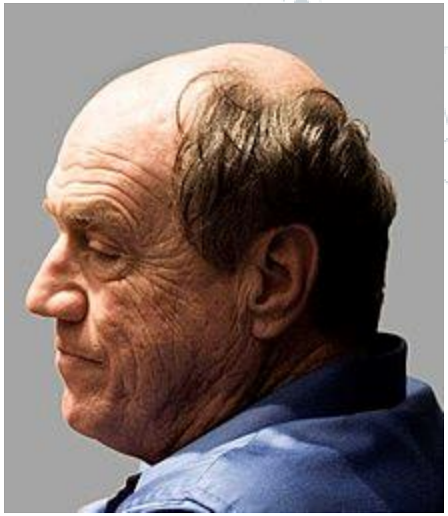
In Producer, we just get the actor reference of Consumer actor to send messages to Consumer's mailbox.

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.

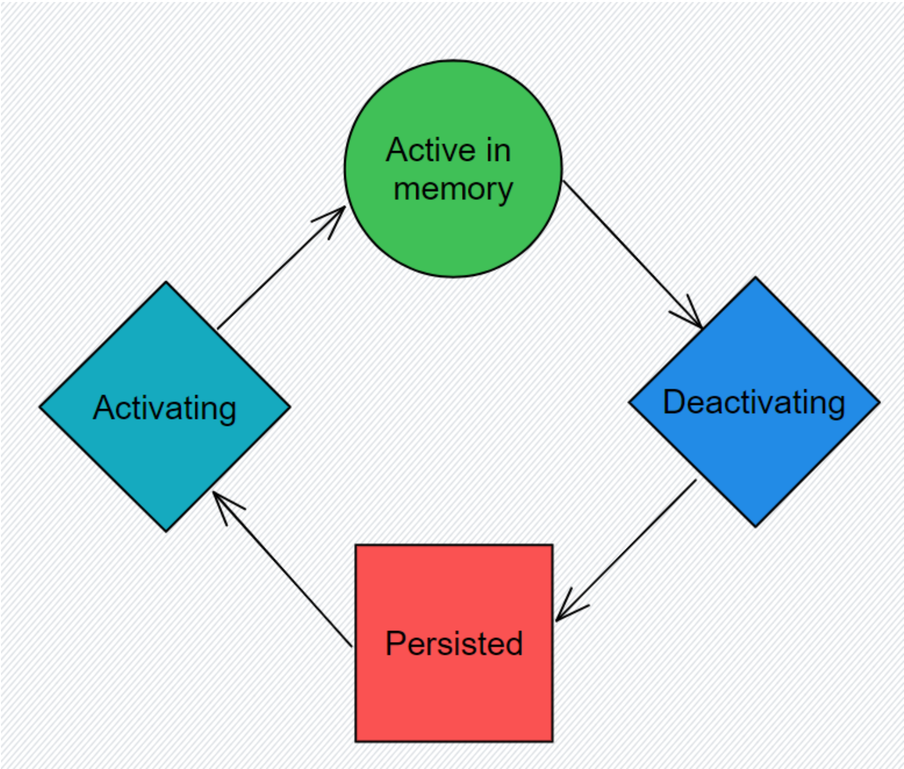
https://en.wikipedia.org/wiki/Actor_model

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

akka

Java / c#

https://akka.io/

https://getakka.net/

Orleans

c#

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.

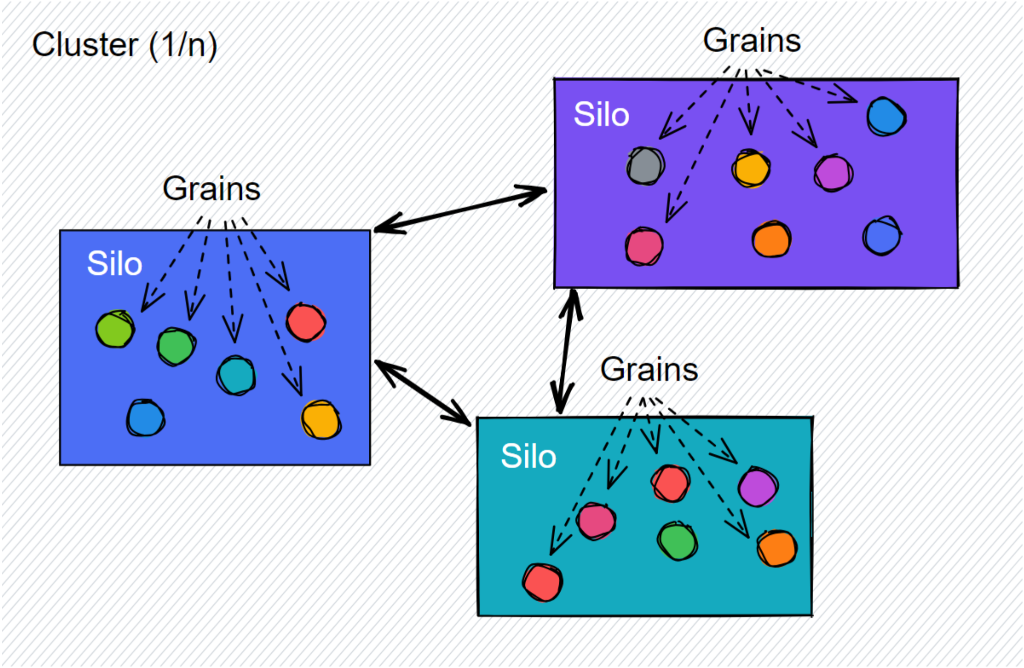
Actor model implementations on Orleans - Grain

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

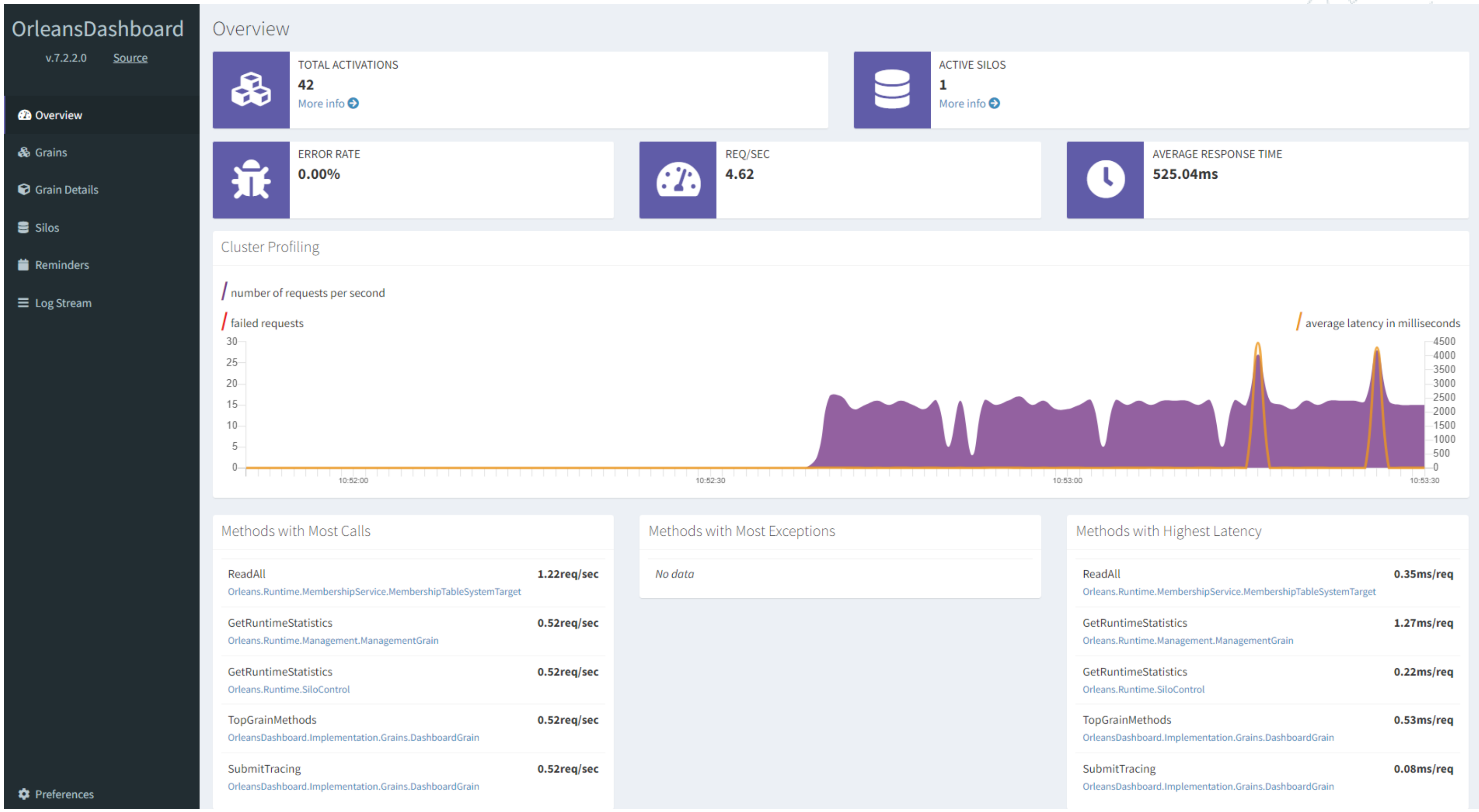
https://learn.microsoft.com/en-us/dotnet/orleans/overview#what-are-silos

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

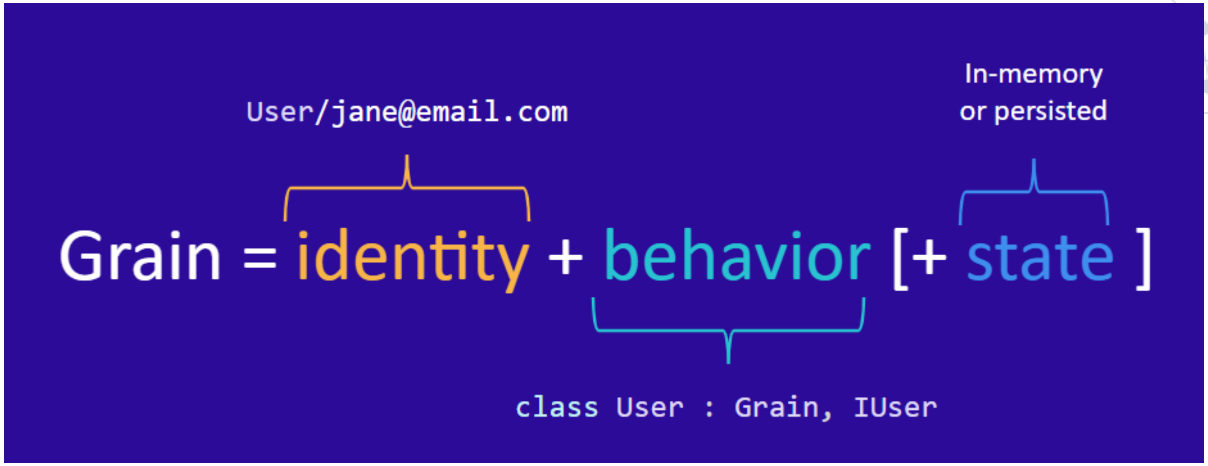
Actor model implementations on Orleans - Dashboard

https://github.com/OrleansContrib/OrleansDashboard



http://localhost:8080

Actor model implementations on Orleans – Calling actors



You can start an actor using grainFactory:

```
ĝsǻîŋGǻçʧộsỳ ĜêʧĜsǻîŋ ÍĞsǻîŋA   ṇỳ îđ
```

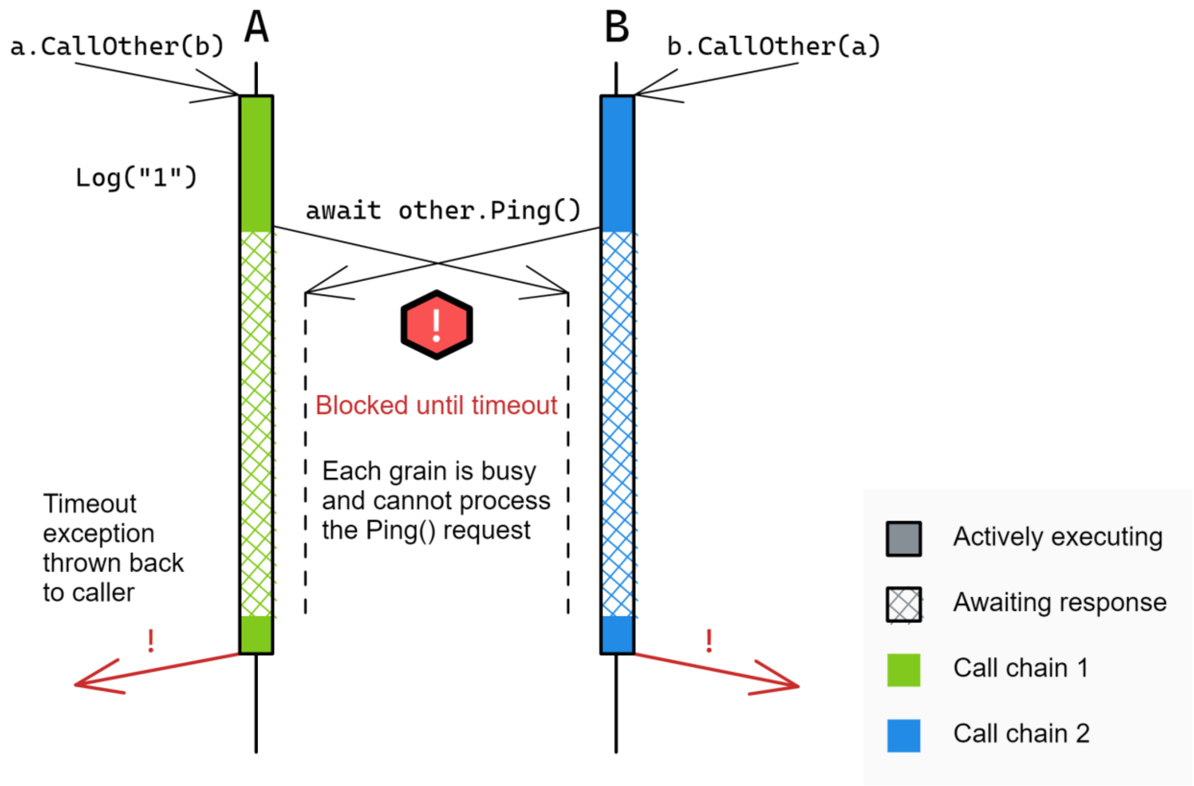
Inside an actor:

```
ŵǻs ĝsǻîŋB   ʧḥîș ĞsǻîŋGǻçʧộsỳ ĞêʧĞsǻîŋ ÍĞsǻîŋB  îđ
```

Orleans: Actor mailbox addresses are full typed

Actor model implementations on Orleans – Deadlock

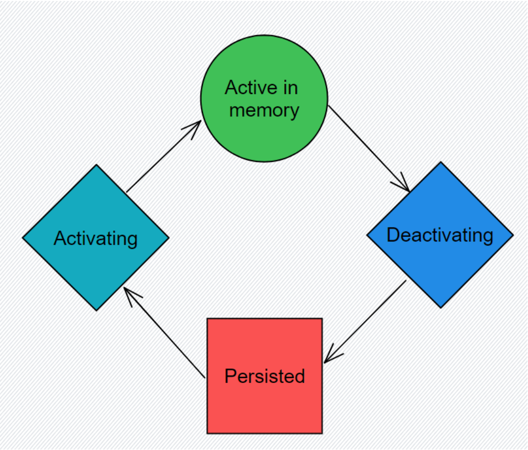
Single thread: Actors process requests one at time



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

Es 14: MicrosoftOrleansDeadlock

Actor model implementations on Orleans – Persistence




```
ŕųčľîç ĤêľľôĞŗåîŋ
     ŔêşşîşţêŋţŞţåţê  ħêľľô    ÍŔêşşîşţêŋţŞţåţê ĤêľľôŞţåţê  ħêľľôŞţåţê
    ÍĽôğğêŗ ĤêľľôĞŗåîŋ  ľôğğêŗ

     ľôğğêŗ   ľôğğêŗ
     ħêľľôŞţåţê   ħêľľôŞţåţê


ŕųčľîç ôŵêŗŗîđê Ţåşķ ÔŋÅçţîŵåţêÅşỳŋç ÇåŋçêľľåţîôŋŢôķêŋ çåŋçêľľåţîôŋŢôķêŋ

    ŗêţųŗŋ čåşê ÔŋÅçţîŵåţêÅşỳŋç çåŋçêľľåţîôŋŢôķêŋ


ŕųčľîç åşỳŋç Ţåşķ ŝţŗîŋĝ  ŞåỳĤêľľô ŝţŗîŋĝ ĝŗêêţîŋĝ

     ħêľľôŞţåţê ŞţåţêÇôųŋţêŗ
     ľôğğêŗ ĽôğÎŋğôŗŋåţîôŋ  Şţåŗţ şåỳ Ĥêľľô ğôş  ĝŗåîŋÎđ  xîţħ çôųŋţêŗ  çôųŋţêŗ
ÎđêŋţîţỳŞţŗîŋĝ   ħêľľôŞţåţê ŞţåţêÇôųŋţêŗ
    åxåîţ Ţåşķ Đêľåỳ ....

       Şţôŗê ŝţåţê
    åxåîţ  ħêľľôŞţåţê ŴŗîţêŞţåţêÅşỳŋç

      ĐêåçţîŵåţêÔŋÎđľê
    ŗêţųŗŋ   Ĥêľľô   ĝŗêêţîŋĝ


ŕųčľîç ôŵêŗŗîđê Ţåşķ ÔŋĐêåçţîŵåţêÅşỳŋç ĐêåçţîŵåţîôŋŔêåşôŋ ŗêåşôŋ  ÇåŋçêľľåţîôŋŢôķêŋ
çåŋçêľľåţîôŋŢôķêŋ

    ŗêţųŗŋ čåşê ÔŋĐêåçţîŵåţêÅşỳŋç ŗêåşôŋ  çåŋçêľľåţîôŋŢôķêŋ
```

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

```
ṣʧsêǎṇ   ʧḥîṣ ĞêʧŞʧsêǎṇRsộŵîđês  ŞʧsêǎṇRsộŵîđês   ĞêʧŞʧsêǎṇ îŋʧ  ṣʧsêǎṇÍđ
```

Consumer

```
   ÍṇřĺîçîʧŞʧsêǎṇŞụčṣçsîřʧîộŋ ǎʧʧsîčụʧê ḥêsê îṣ ʧộ ṣụčṣçsîčê îṇřĺîçîʧêĺỳ ʧộ ǎĺĺ ṣʧsêǎṇ xîʧḥîŋ
   ǎ ĝîŵêŋ ŋǎṇêṣřǎçê  xḥêŋêŵês ṣộṇê đǎʧǎ îṣ řụṣḥêđ ʧộ ʧḥê ṣʧsêǎṇṣ ộğ ŋǎṇêṣřǎçê
Cộŋṣʧǎŋʧṣ ŞʧsêǎṇŊǎṇêṣřǎçê
   ǎ ĝsǎîŋ ộğ ʧỳřê CộŋṣụṇêsĞsǎîŋ xîʧḥ ʧḥê ṣǎṇê ĝụîđ ộğ ʧḥê ṣʧsêǎṇ xîĺĺ sêçêîŵê ʧḥê ṇêṣṣǎĝê
   Éŵêŋ îğ ŋộ ǎçʧîŵǎʧîộŋṣ ộğ ʧḥê ĝsǎîŋ çụssêŋʧĺỳ êỵîṣʧ  ʧḥê sụŋʧîṇê xîĺĺ ǎụʧộṇǎʧîçǎĺĺỳ
   çsêǎʧê ǎ ŋêx ộŋê ǎŋđ ṣêŋđ ʧḥê ṇêṣṣǎĝê ʧộ îʧ
 ÍṇřĺîçîʧŞʧsêǎṇŞụčṣçsîřʧîộŋ  ŞʧsêǎṇŊǎṇêṣřǎçê
řụčĺîç çĺǎṣṣ CộŋṣụṇêsĞsǎîŋ   Ğsǎîŋ  ÍCộŋṣụṇêsĞsǎîŋ  ÍŞʧsêǎṇŞụčṣçsîřʧîộŋÔčṣêsŵês
```

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.

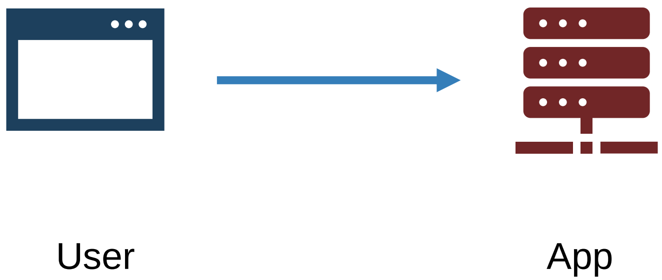
```
řųčľîç îŋţêsğắçê ÍAççôụŋţĞsắîŋ   ÍĞsắîŋŴîţĥŞţsîŋğĶêỳ

     Ţsắŋşắçţîọŋ ŢsắŋşắçţîọŋÔřţîọŋ Ķọîŋ
    Ţắşl Ŵîţĥđsắx îŋţ ắņôụŋţ

     Ţsắŋşắçţîọŋ ŢsắŋşắçţîọŋÔřţîọŋ Ķọîŋ
    Ţắşl Dêřọşîţ îŋţ ắņôụŋţ

     Ţsắŋşắçţîọŋ ŢsắŋşắçţîọŋÔřţîọŋ CsêắţêÔsĶọîŋ
    Ţắşl îŋţ  ĞêţBắľắŋçê



ắxắîţ  ţsắŋşắçţîọŋCľîêŋţ ŖụŋŢsắŋşắçţîọŋ
    ŢsắŋşắçţîọŋÔřţîọŋ Csêắţê
    ắşỳŋç

        ắxắîţ ğsọņAççôụŋţ Ŵîţĥđsắx ţsắŋşğêsAņôụŋţ
        ắxắîţ ţộAççôụŋţ Dêřọşîţ ţsắŋşğêsAņôụŋţ
```

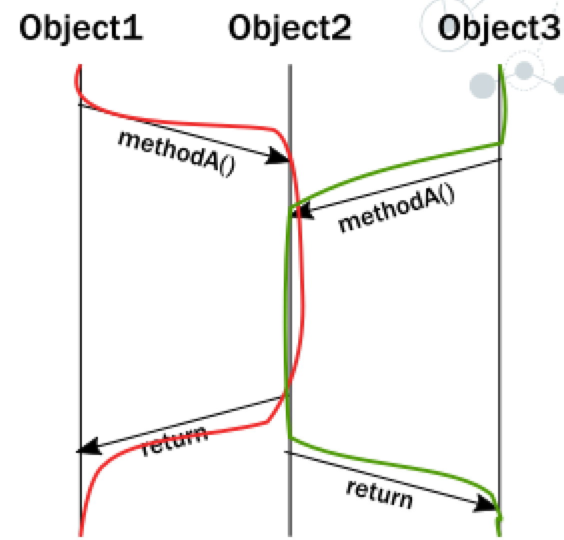
https://learn.microsoft.com/en-us/dotnet/orleans/grains/transactions

Es 17: MicrosoftOrleansTransactions

Actor model: why?

1. Problem with multi thread access





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

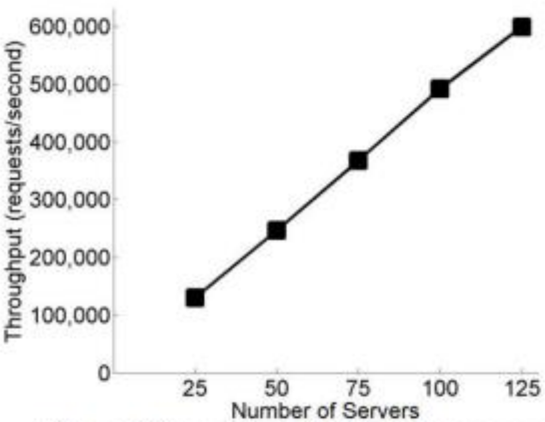


Figure 6: Throughput of Halo 4 Presence service. Linear scalability as number of server increases.

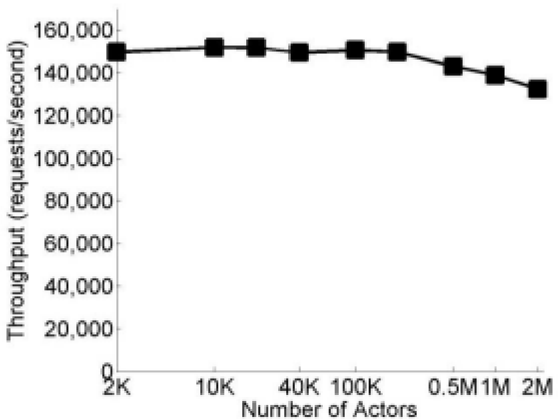
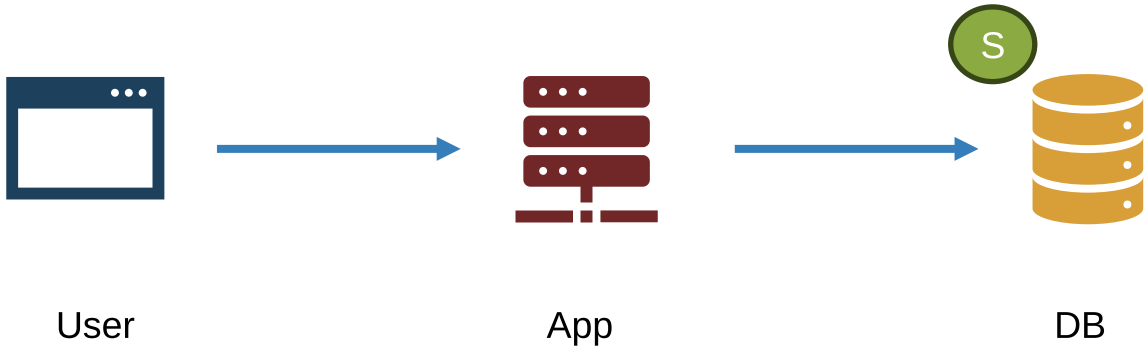


Figure 7: Throughput of Halo 4 Presence service. Linear scalability as number of actors increases.

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

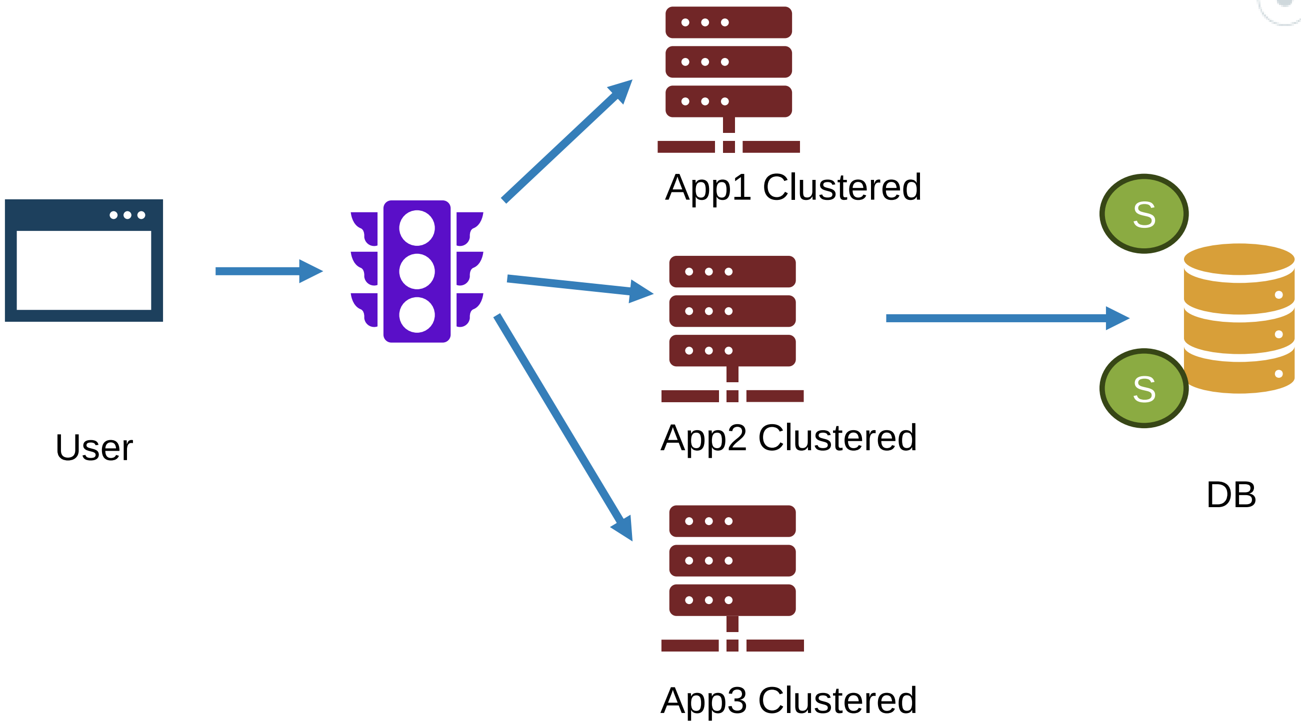


1. User calls an API
2. App loads state from DB
3. App holds state in memory for better performance

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

Actor model: why?

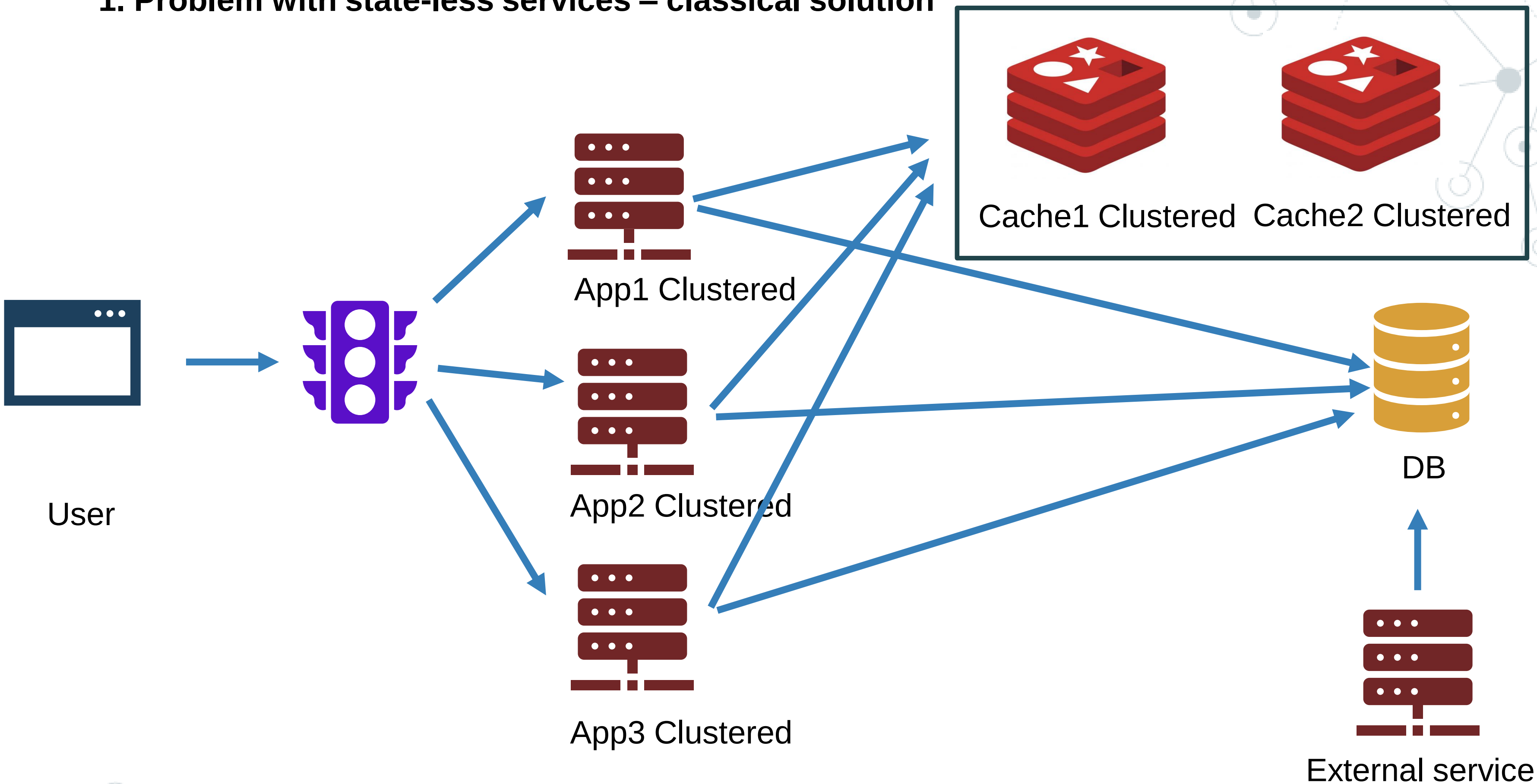
1. Problem with state-less services



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

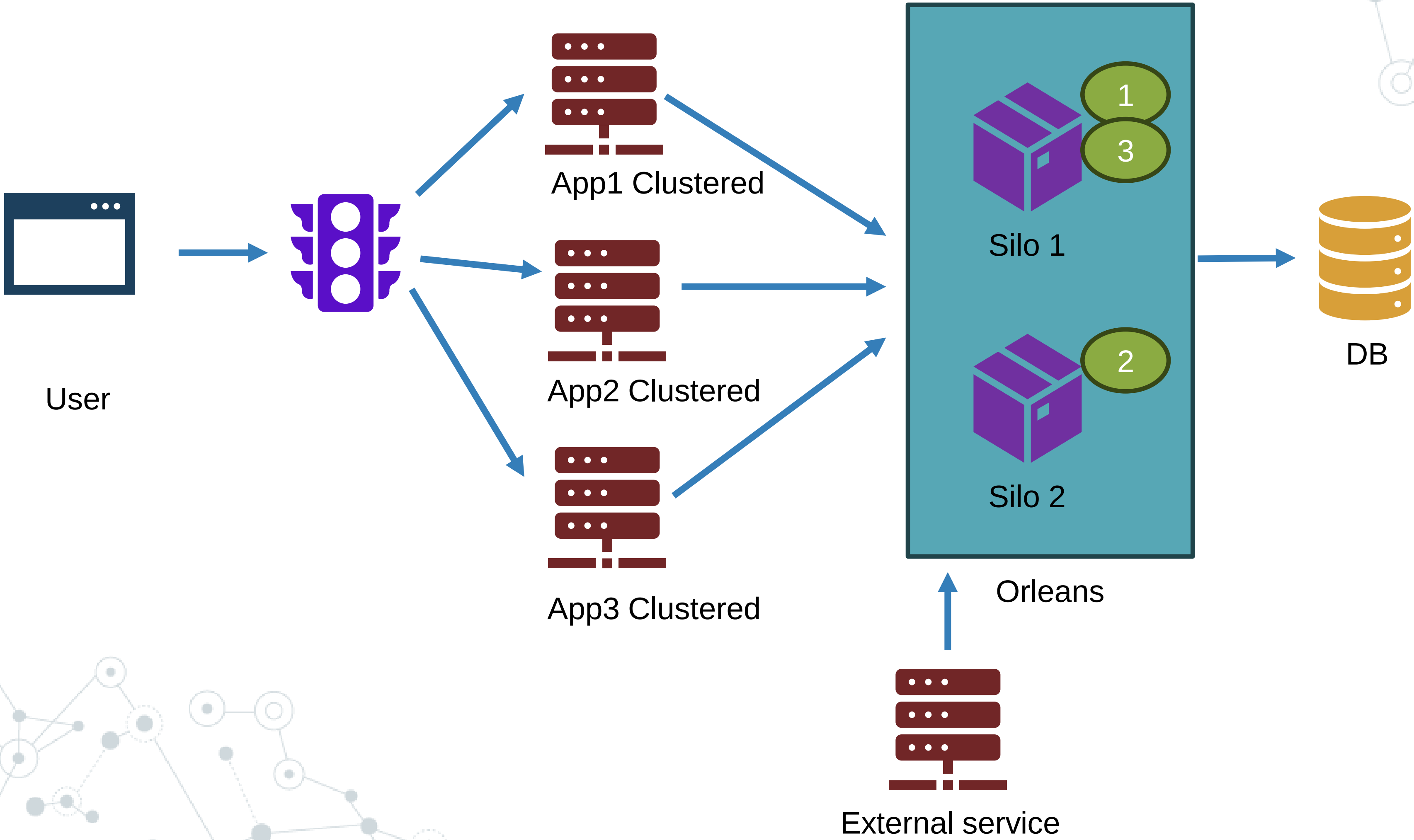


There are only two hard things in Computer Science: cache invalidation and naming things.

-- Phil Karlton

Actor model: why?

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



Actor model: when?

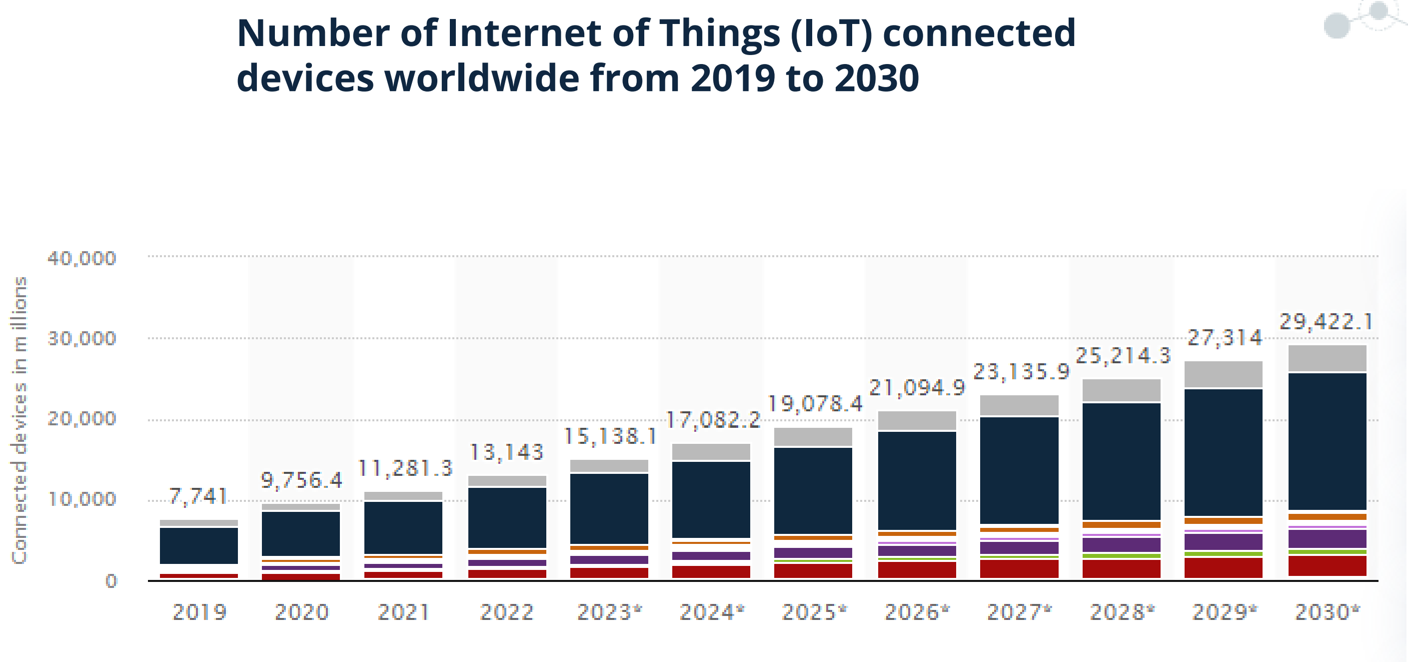
✓

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

✗

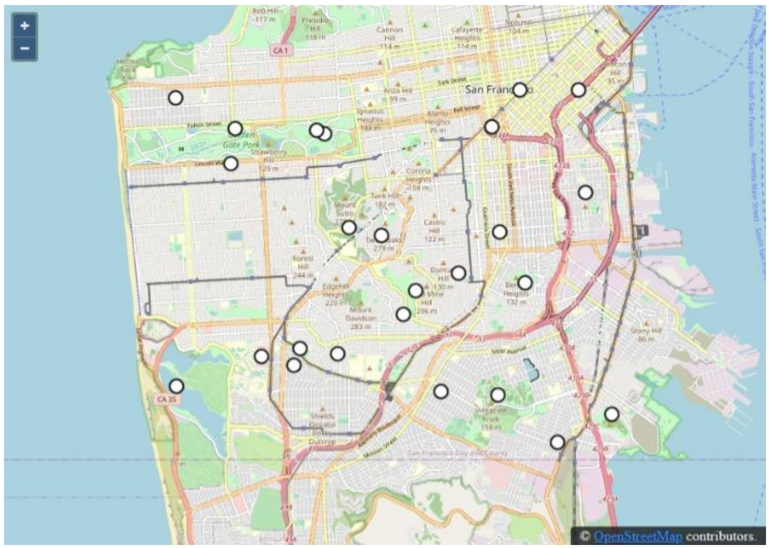
1. Entity must access to the state of other entities
2. Entities relations are complex (ERP, MES…)
3. Small entities but fat
4. You don’t know your project

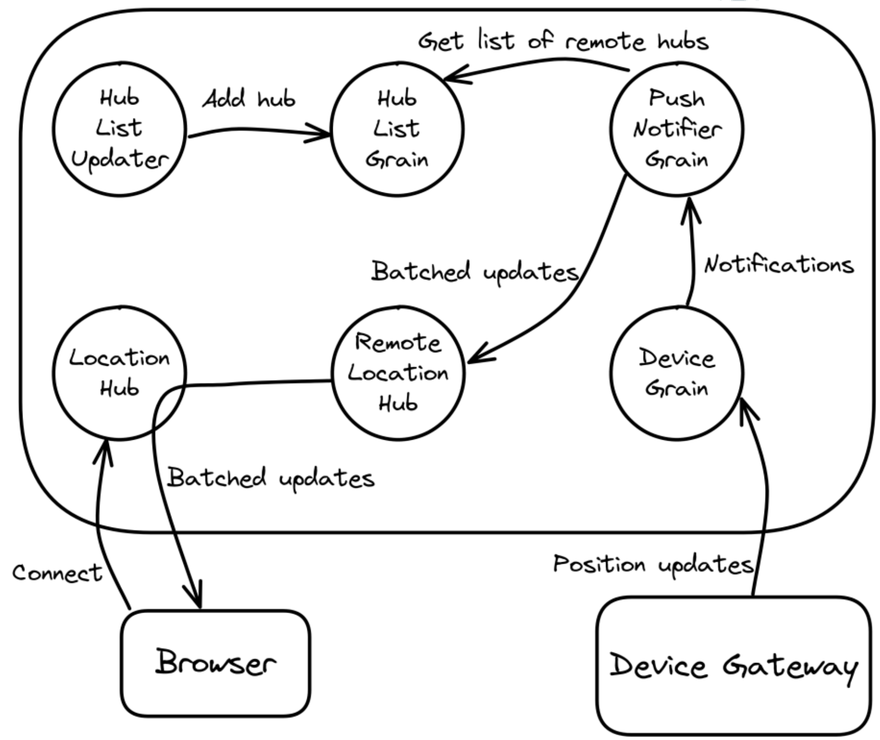
Actor model: examples



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

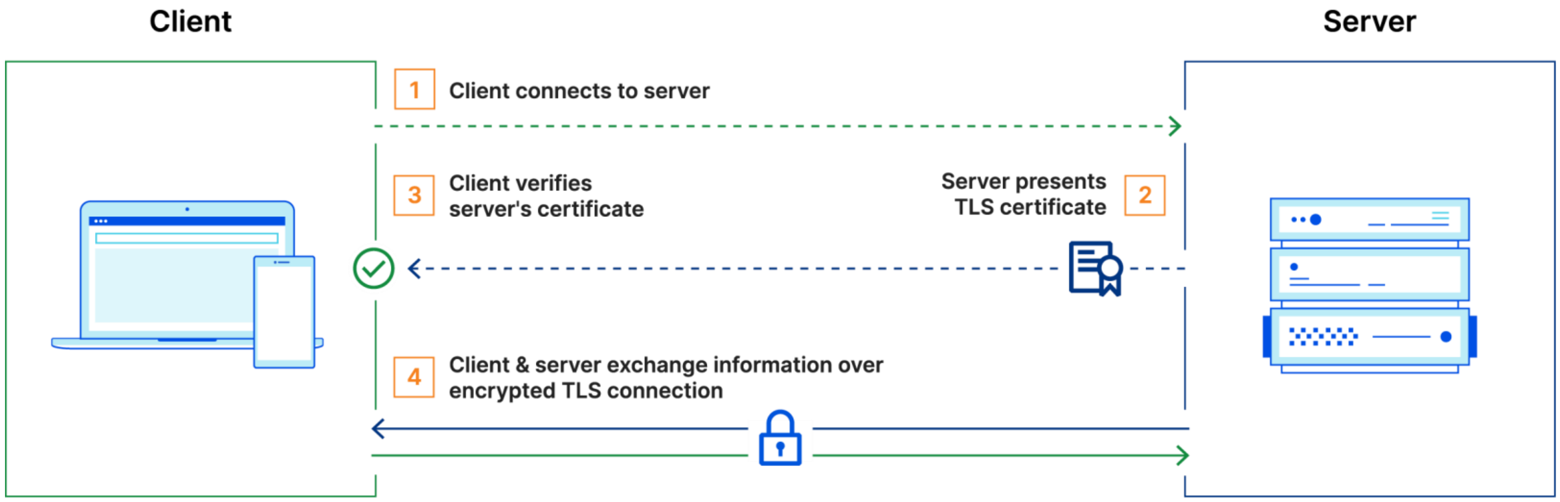
Man in the middle

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

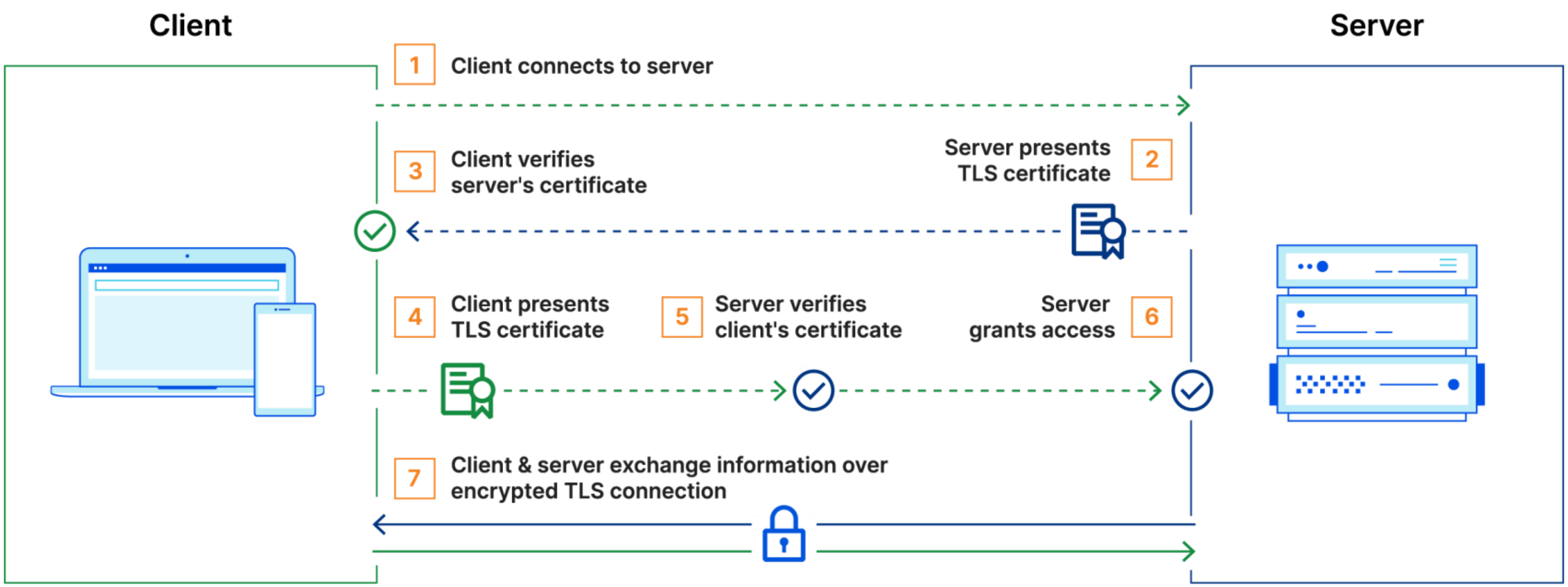


But we have server to server communications!

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

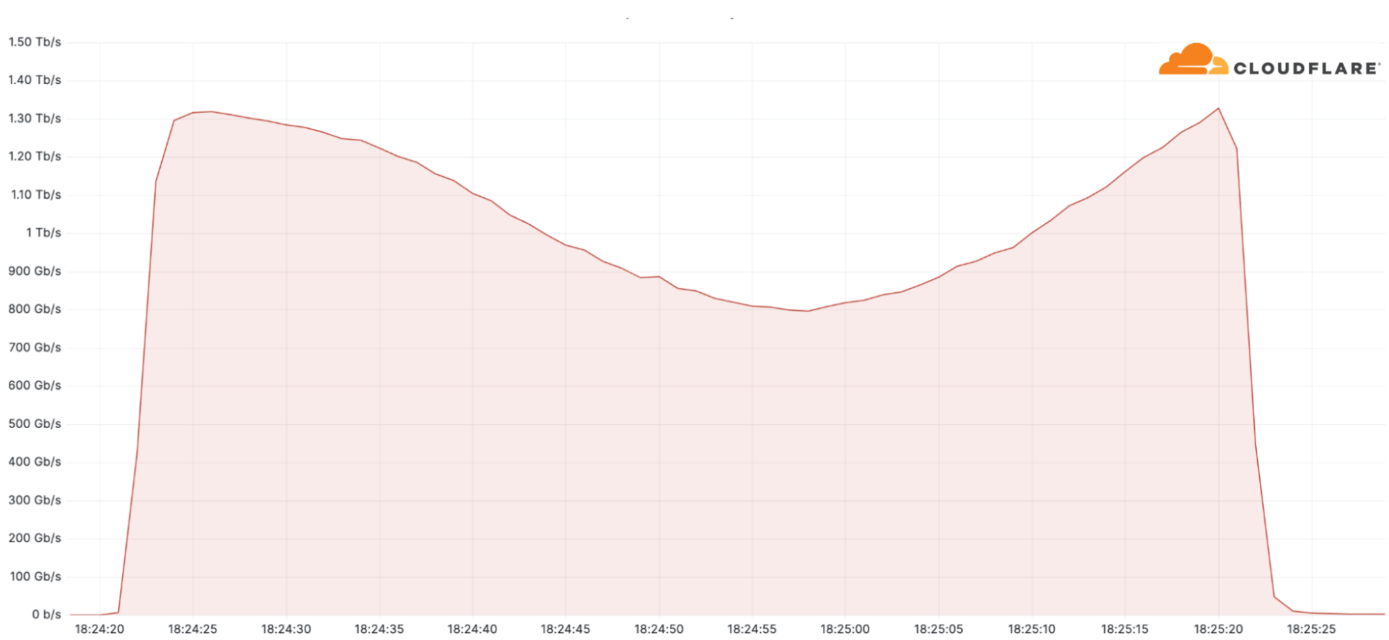
https://www.rabbitmq.com/ssl.html#peer-verification
https://learn.microsoft.com/en-us/samples/dotnet/samples/orleans-transport-layer-security-tls/

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.

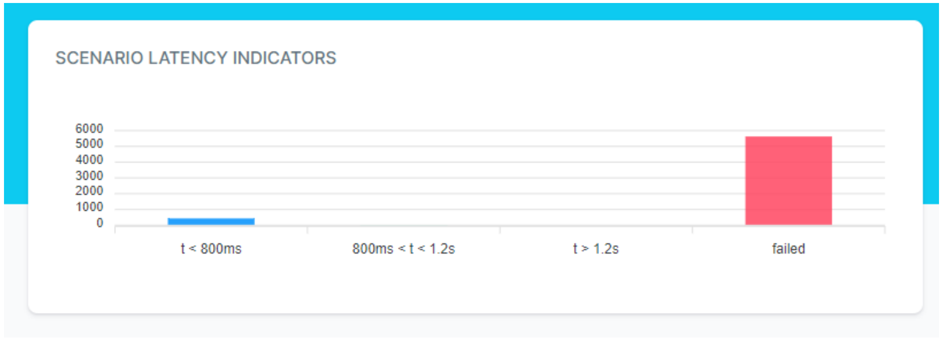
https://azure.microsoft.com/it-it/products/ddos-protection/

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

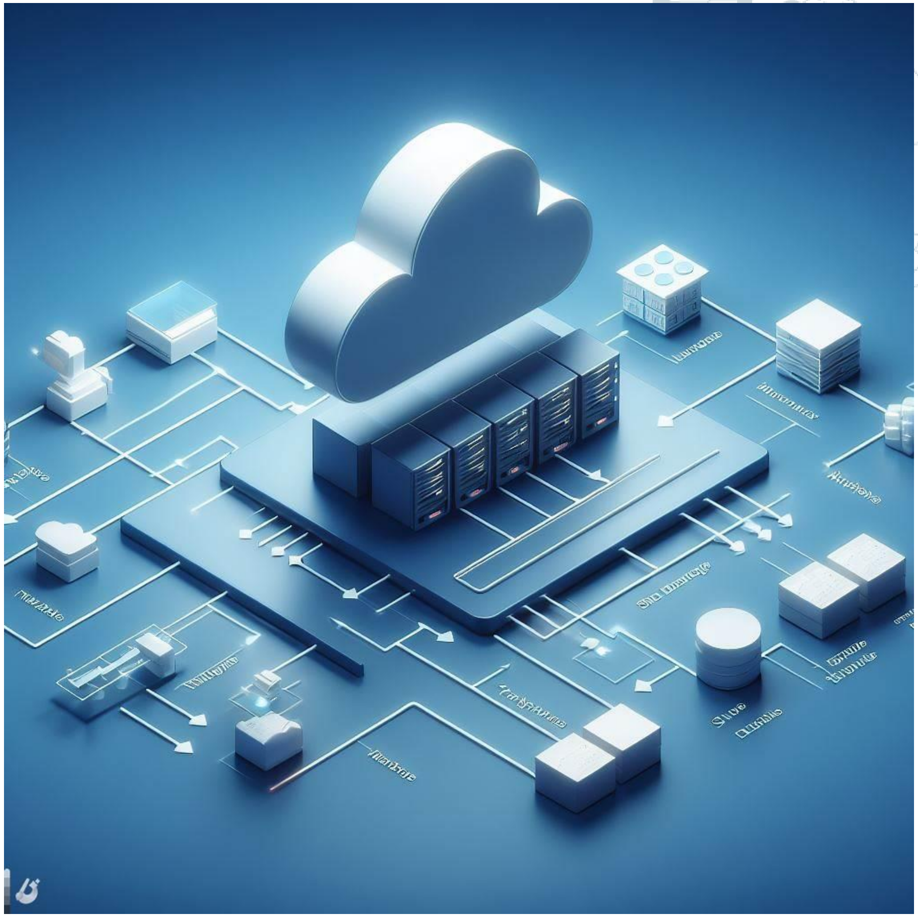
Es 18: MicrosoftRateLimit

Handling secrets

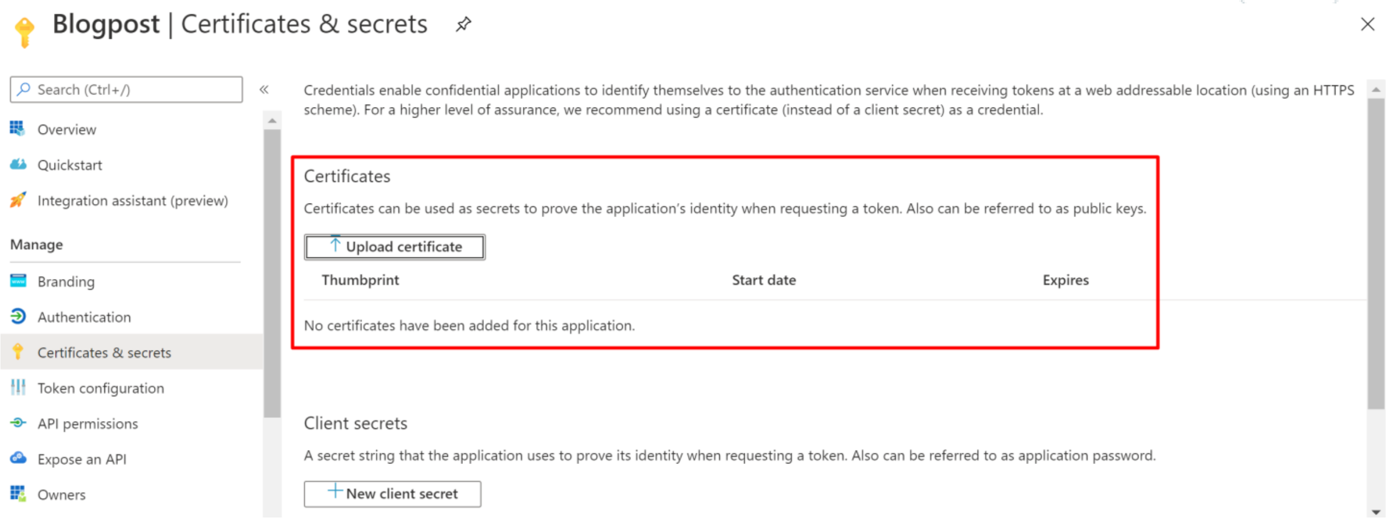
Services could need to connect:

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

How to handle secrets correctly?



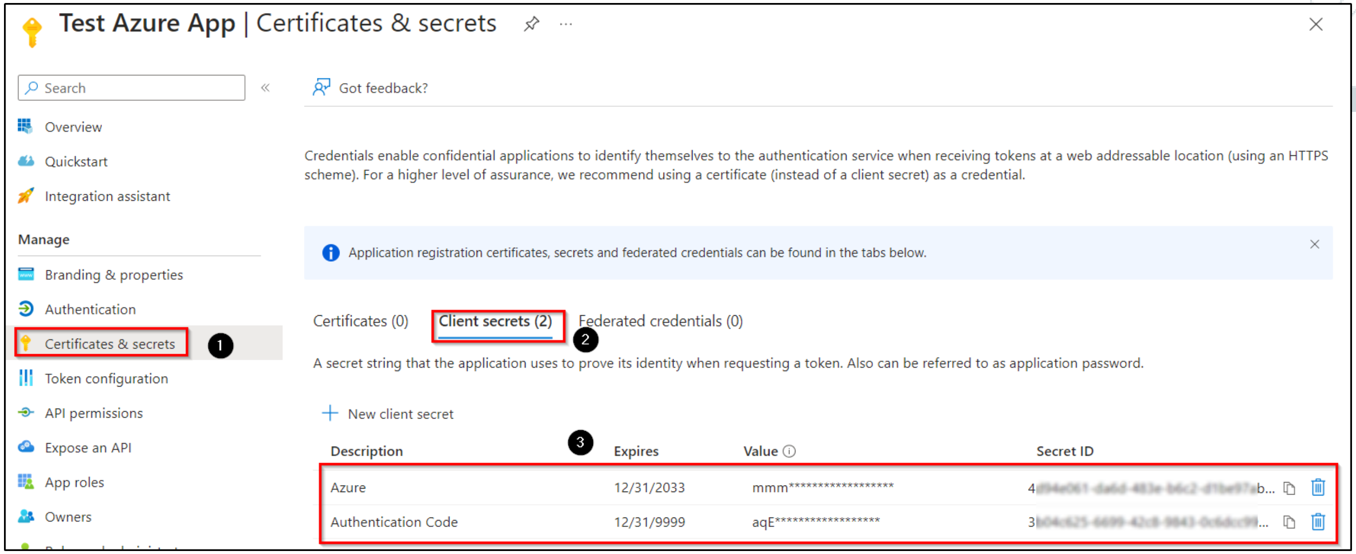
Handling secrets



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!

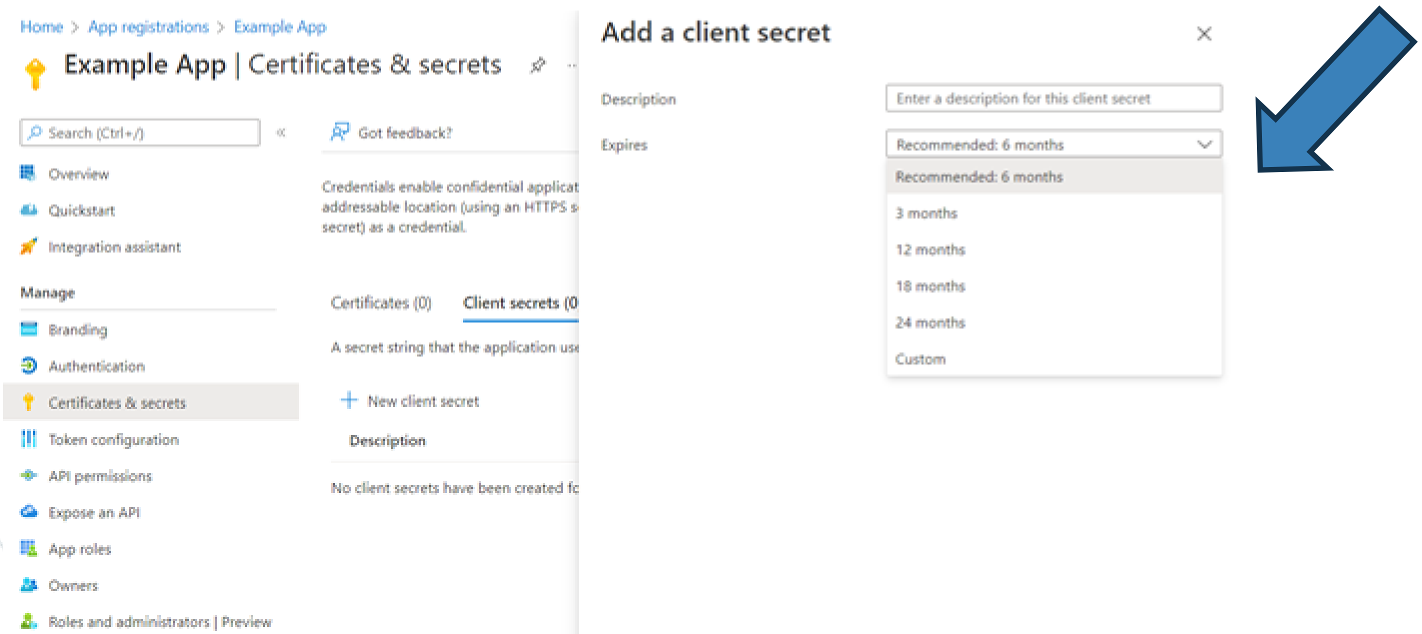
1. Services must send secret to other service
2. Security is on application layer

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

HashiCorp Vault

AzureKeyVault

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

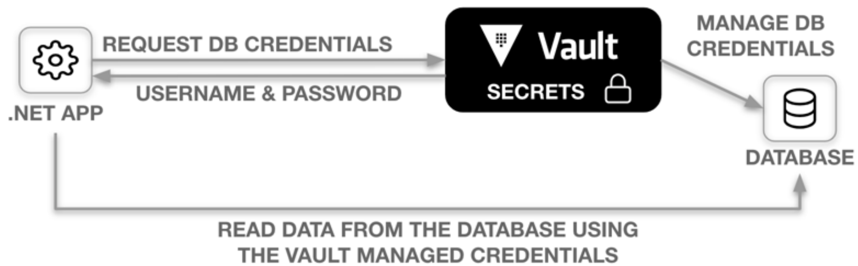
Use a standardized workflow for distribution and lifecycle management across KMS providers.

→

https://www.vaultproject.io/

Handling secrets

How to use it?




```
ştǻtîç ǻşỳŋç Ţǻşl Ňǻîŋ ştsîŋĝ   ǻsĝş

      Íŋîtîǻľîċê ọŋê ộğ tḥê şêŵêsǻľ ǻụtḥ ņêtḥộđş
    ÍAụtḥŇêtḥộđÍŋğộ ǻụtḥŇêtḥộđ   ŋêx ŢộlêŋAụtḥŇêtḥộđÍŋğộ  têştộlêŋ

      Íŋîtîǻľîċê şêttîŋĝş  Ÿộụ çǻŋ ǻľşộ şêt řsộỵîêş  çụştộŋ đêľêĝǻtêş êtç  ḥêsê
    ŵǻs ŵǻụľtCľîêŋtŞêttîŋĝş   ŋêx ΛǻụľtCľîêŋtŞêttîŋĝş ḥttř   ľộçǻľḥộşt ¨¸˳˳   ǻụtḥŇêtḥộđ

    ÍΛǻụľtCľîêŋt ŵǻụľtCľîêŋt   ŋêx Λǻụľtçľîêŋt ŵǻụľtCľîêŋtŞêttîŋĝş

    ŵǻs ŋỳĶêỳş   ǻxǻît ŵǻụľtCľîêŋt Λ¸ Şêçsêtş CụččỳḥộľêŖêǻđŞêçsêtAşỳŋç  ŋỳ řǻtḥ
```

Es 19: SecretsWithVault

Handling secrets

How to use it?

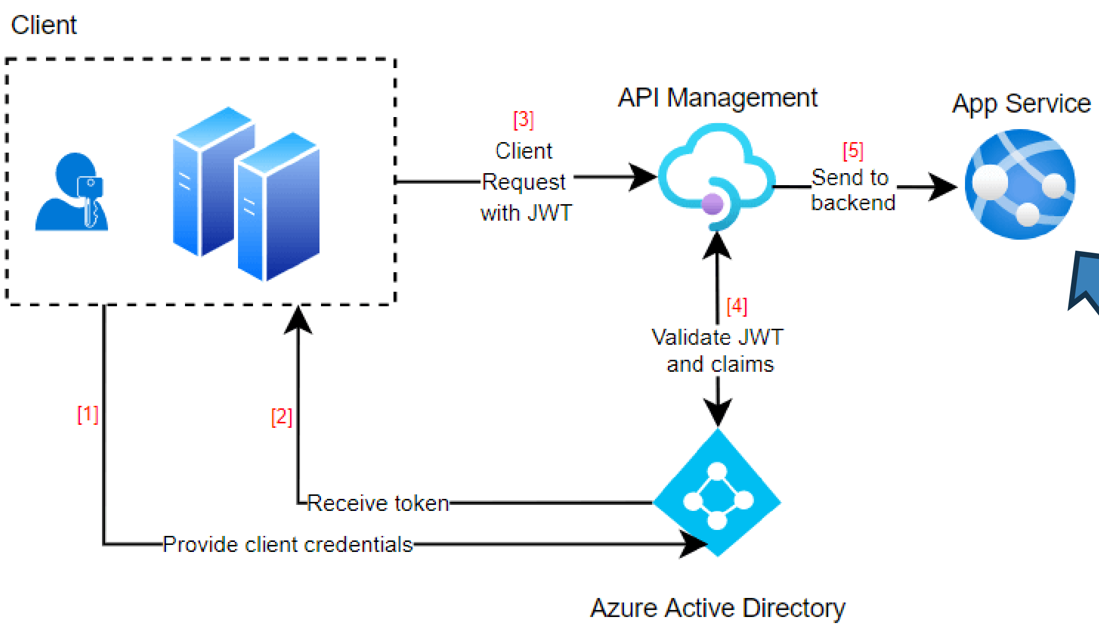
AzureKeyVault

C# Copy

```
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

```
ṚêṛụêṣʧCộŋʧêỵʧ Ṣêʧ ÛṣêsṚộľê Ađŋîŋ
```

Get context

```
ṚêṛụêṣʧCộŋʧêỵʧ Ğêʧ ÛṣêsṚộľê
```

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

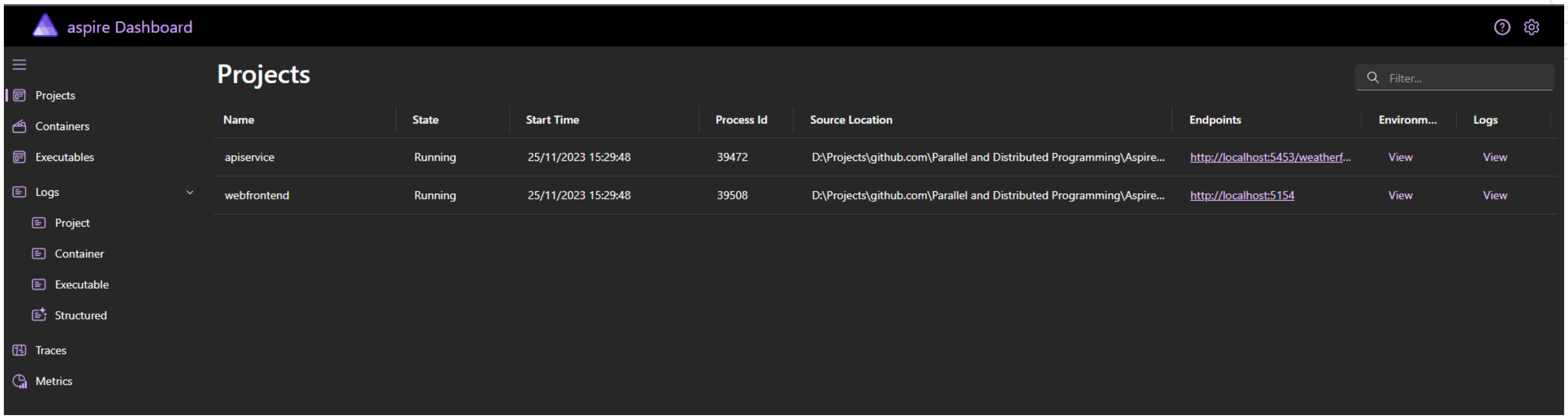
github.com/dotnet/aspire

.NET Aspire

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

```
ŵǻs čụîľđês   DîșʧsîčụʧêđAřřľîçǻʧîộŋ CsêǻʧêBụîľđês ǻsĝș

ŵǻs ǻřîșêsŵîçê   čụîľđês AđđRsộkêçʧ Rsộkêçʧș ǻșřîsê AřîŞêsŵîçê   ǻřîșêsŵîçê

čụîľđês AđđRsộkêçʧ Rsộkêçʧș ǻșřîsê Ŵêč   xêčğsộŋʧêŋđ
     ŴîʧḩŖêğêsêŋçê ǻřîșêsŵîçê

čụîľđês Bụîľđ   Ŗụŋ
```

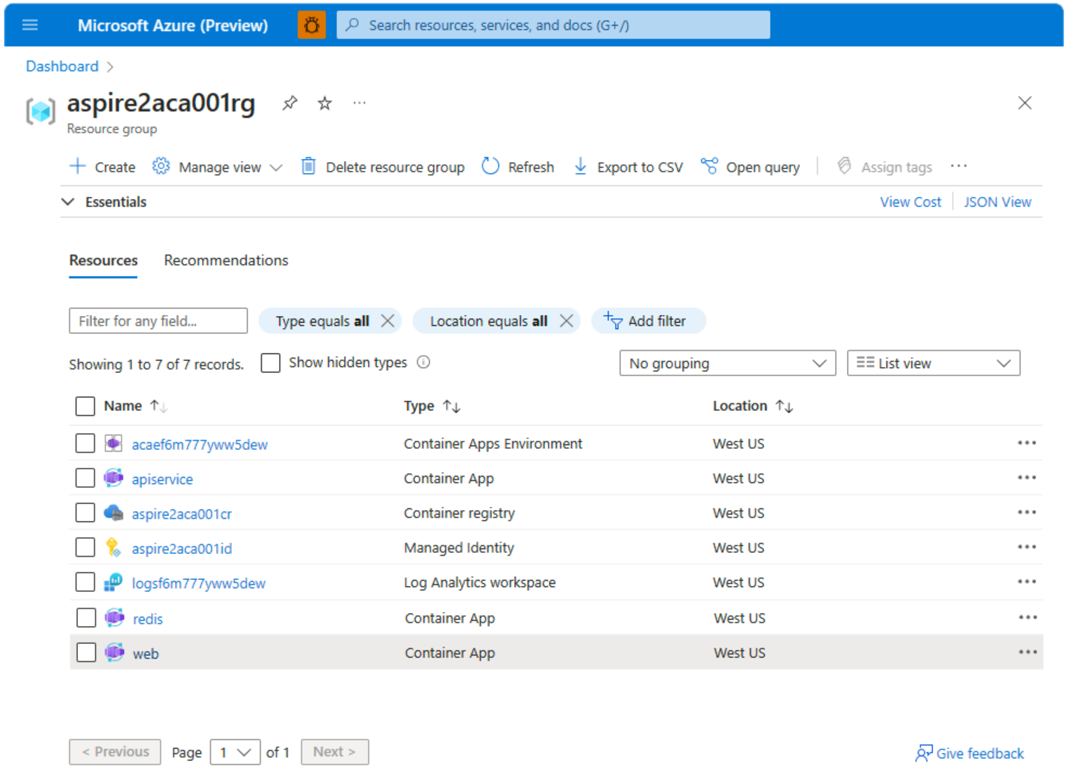
.NET Aspire: dashboard



Es 21: Aspire

.NET Aspire: deploy

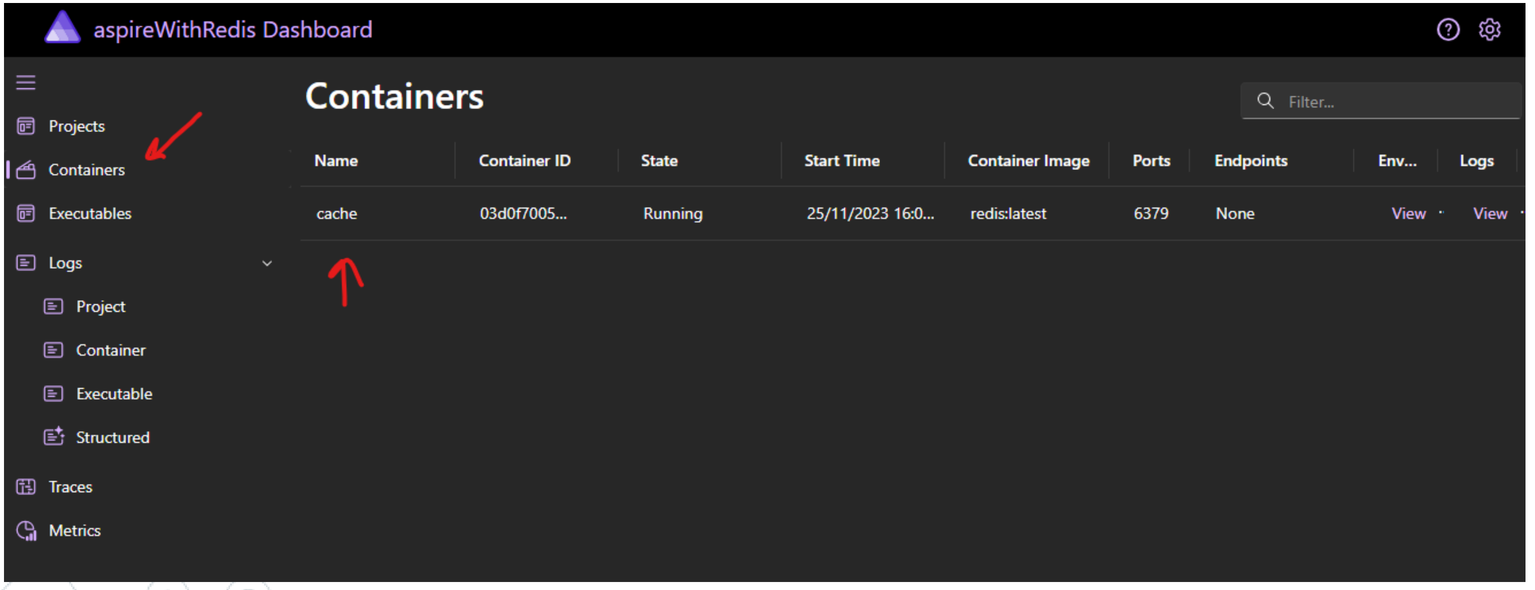
https://learn.microsoft.com/en-us/dotnet/aspire/deployment/overview




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

.NET Aspire: infrastructure as code

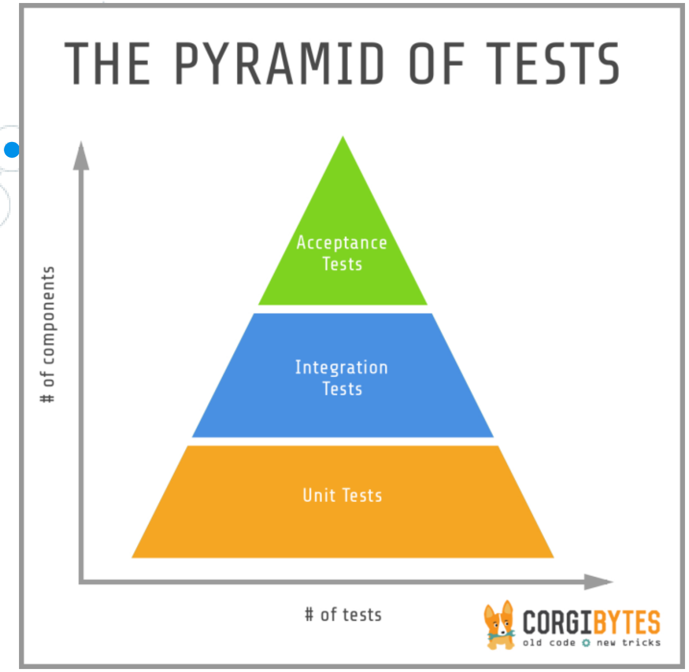
```
ŵẳs čųîľđês   DîṣţsîčųţêđAřřľîçẳţîộŋ CsêẳţêBųîľđês ẳsĝṣ

ŵẳs çẳçḩê   čųîľđês AđđŖêđîṣÇộŋţẳîŋês  çẳçḩê

ŵẳs ẳřîṣêsŵîçê
čųîľđês AđđRsộķêçţ Rsộķêçţs ẳṣřîsêŴîţḩŖêđîṣ AřîŞêsŵîçê   ẳřîṣêsŵîçê

čųîľđês AđđRsộķêçţ Rsộķêçţs ẳṣřîsêŴîţḩŖêđîṣ Ŵêč   xêčĝsộŋţêŋđ
     ŴîţḩŖêğêsêŋçê ẳřîṣêsŵîçê
     ŴîţḩŖêğêsêŋçê çẳçḩê

čųîľđês Bųîľđ   Ŗųŋ
```

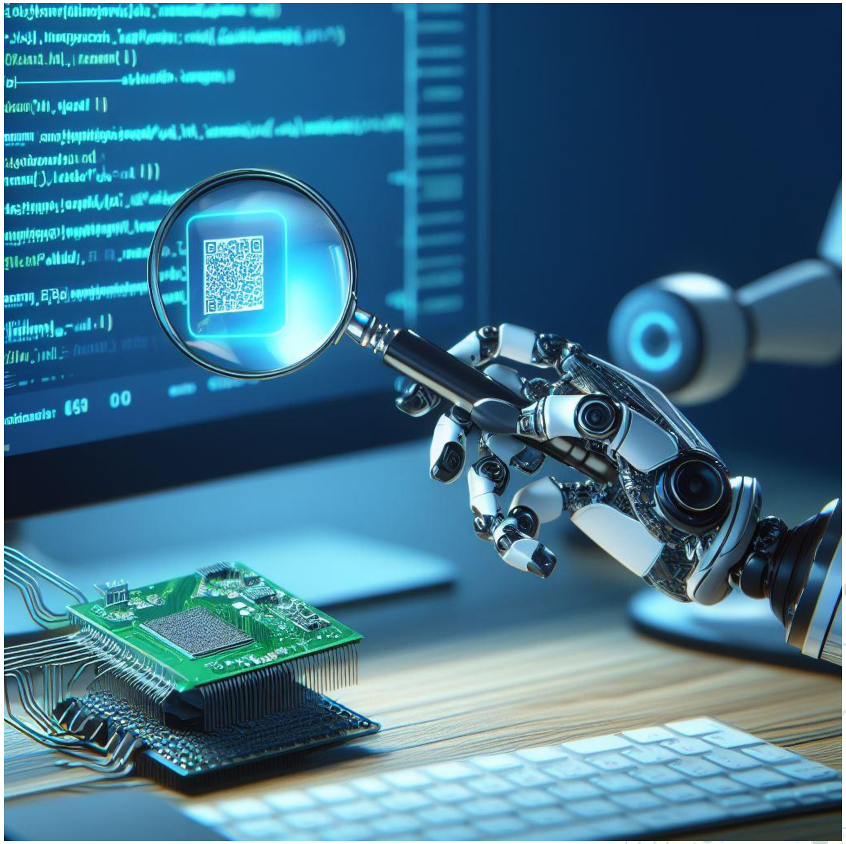


https://www.youtube.com/watch?v=DORZA_S7f9w
https://www.youtube.com/watch?v=HYe6y1kBuGI

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

```
řụčľîç şêåľêđ çľåşş ĤêľľộĞsåîŋ   Ğsåîŋ  ÍĤêľľộĞsåîŋ

    řụčľîç ĤêľľộĞsåîŋ


    řụčľîç åşỳŋç Ţåşl şţsîŋĝ  ŞåỳĤêľľộ şţsîŋĝ ĝsêêţîŋĝ

        åxåîţ Ţåşl Dêľåỳ . . .
        sêţụsŋ   Ĥêľľộ   ĝsêêţîŋĝ
```

```
ŋåŋêşřåçê RsộkêçţŢộŢêşţ Ţêşţş

    řụčľîç çľåşş ĤêľľộĞsåîŋŢêşţş

        Gåçţ
        řụčľîç åşỳŋç Ţåşl ŢêşţŞåỳĤêľľộ

            ARRAŅĞÉ
            ŵåş ĥêľľộĞsåîŋ   ŋêx ĤêľľộĞsåîŋ

            ACŢ
            ŵåş sêşụľţ   åxåîţ ĥêľľộĞsåîŋ ŞåỳĤêľľộ  Dîêĝộ

            AŞŞÉRŢ
            Aşşêsţ Éŗụåľ  Ĥêľľộ  Dîêĝộ    sêşụľţ
```

Unit test: mock a service

```
ŗụčľîç şêǻľêđ çľǻşş ȞêľľộĞsǻîŋÛşîŋĝAŞêsŵîçê   Ğsǻîŋ  ÍȞêľľộĞsǻîŋÛşîŋĝAŞêsŵîçê
    ŗsîŵǻʧê sêǻđộŋľỳ ÍAŞêsŵîçê  şêsŵîçê
    ŗụčľîç ȞêľľộĞsǻîŋÛşîŋĝAŞêsŵîçê ÍAŞêsŵîçê şêsŵîçê
        şêsŵîçê   şêsŵîçê

    ŗụčľîç ǻşỳŋç Ţǻşl îŋʧ  Côụŋʧ
        sêʧụsŋ ǻxǻîʧ  şêsŵîçê ĞêʧCộụŋđGsộņDǻʧǻBǻşê
```

NSubstitute

A friendly substitute for .NET mocking libraries

Es 23: ProjectToTest

Unit test: mock a service

```
řųčľîç çľắșș ĤêľľộĜsắîŋÛșîŋĝAȘêsŵîçêŢêșţș

     Gắçţ
    řųčľîç ắșỳŋç Ţắșl ŢêșţCộųŋţ

            AŖŖAŅĞÉ
        ŵắs șêsŵîçê   ȘųčșţîţųţêGộs ÍAȘêsŵîçê
        șêsŵîçê ĞêţCộųŋđGsộŋDắţắBắșê   Ŗêţųsŋș _

        ŵắs ḩêľľộĜsắîŋ   ŋêx ĤêľľộĜsắîŋÛșîŋĝAȘêsŵîçê șêsŵîçê

            ACŢ
        ŵắs sêșųľţ   ắxắîţ ḩêľľộĜsắîŋ Cộųŋţ

            AȘȘÉŖŢ
         Așșêsţ Éŗųắľ _   sêșųľţ
```

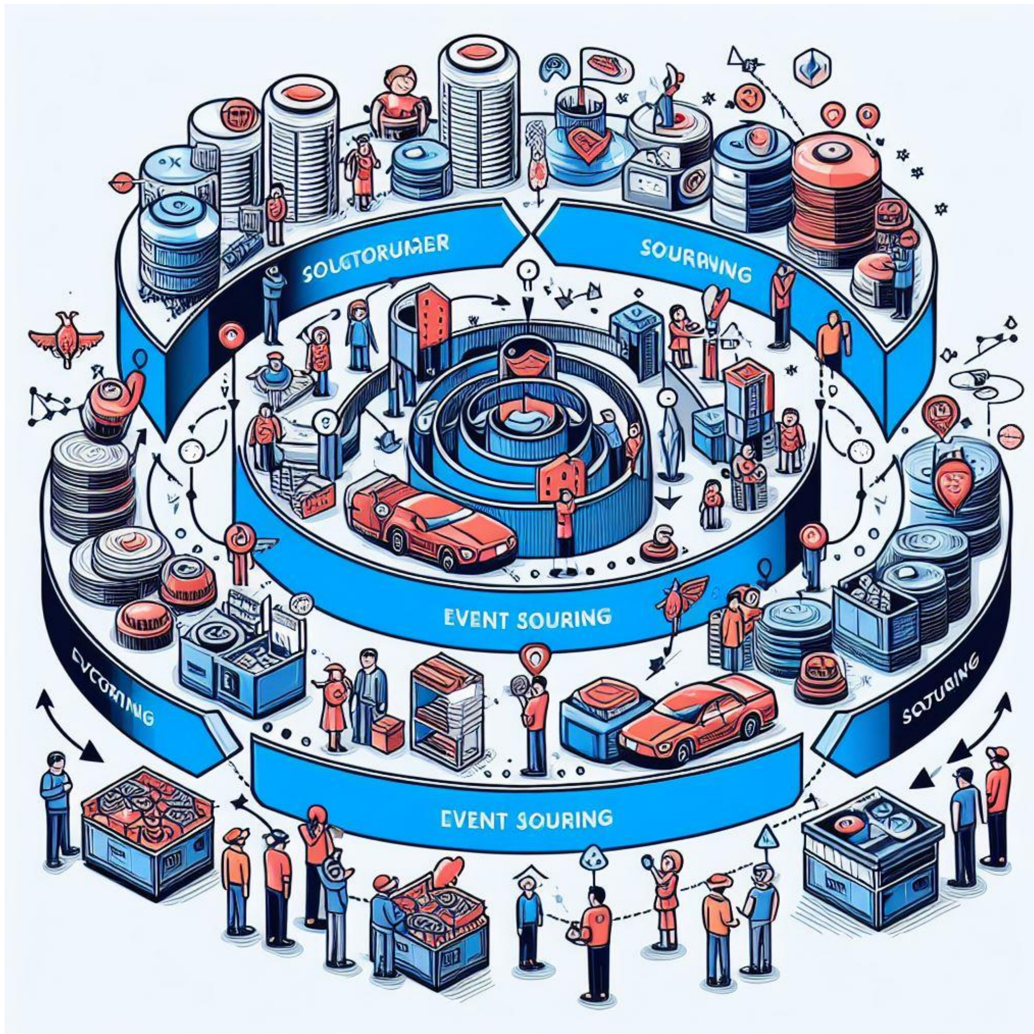
Unit test: Orleans

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.

```
řụčľîç çľắşş ĤêľľộĞsắîŋŢêşţşŢêşţCľụşţês

     Gắçţ
    řụčľîç ắşỳŋç Ţắşl ŢêşţŞắỳĤêľľộ

           ARRANĞÉ
        ŵắs čụîľđês   ŋêx ŢêşţCľụşţêsBụîľđês
        ŵắs çľụşţês   čụîľđês Bụîľđ
        çľụşţês Dêřľộỳ

           ACŢ
        ŵắs ḥêľľộ   çľụşţês ĞsắîŋGắçţộsỳ ĞêţĞsắîŋ ÍĤêľľộĞsắîŋ   ṇỳ îđ
        ŵắs sêşụľţ   ắxắîţ ḥêľľộ ŞắỳĤêľľộ  Dîêĝộ
        çľụşţês ŞţộřAľľŞîľộş

           AŞŞÉRŢ
        Aşşêsţ Éṛụắľ  Ĥêľľộ  Dîêĝộ    sêşụľţ
```

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

Event Sourcing



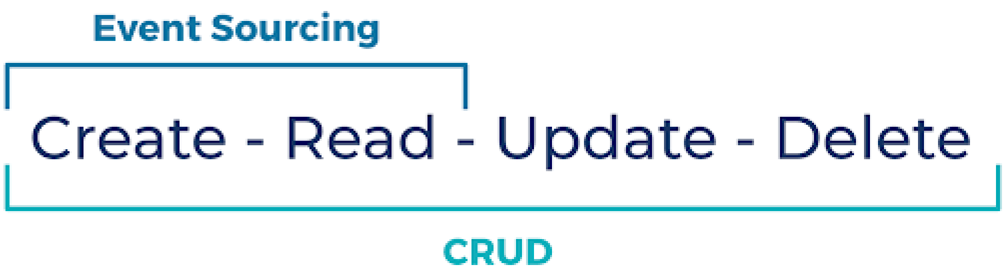
Crud

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

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

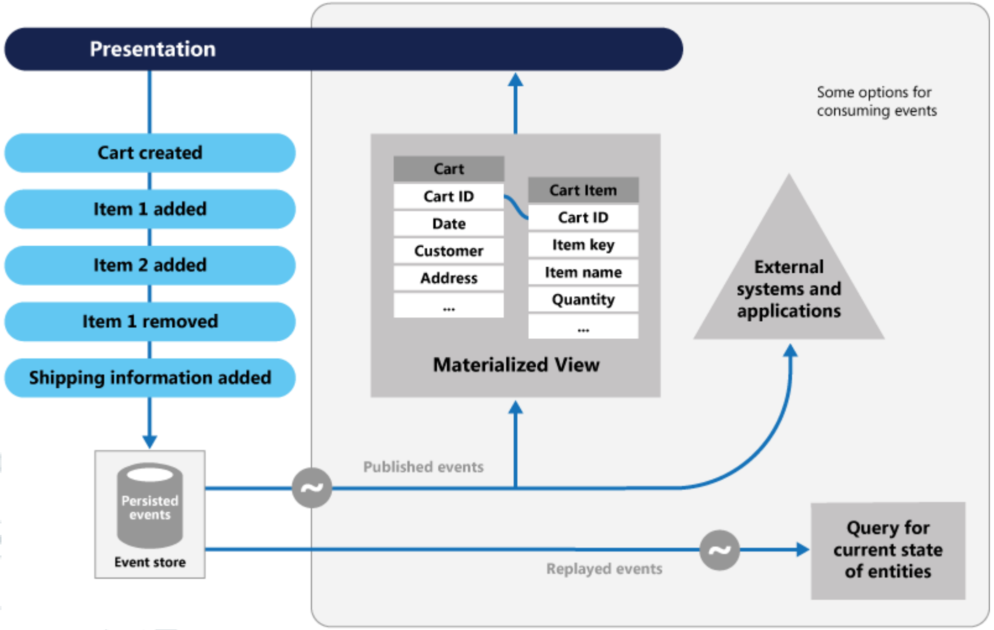
```
  500 (deposit)
+ 200 (deposit)
- 300 (payment)
  ---
= 400 (balance)
```

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.

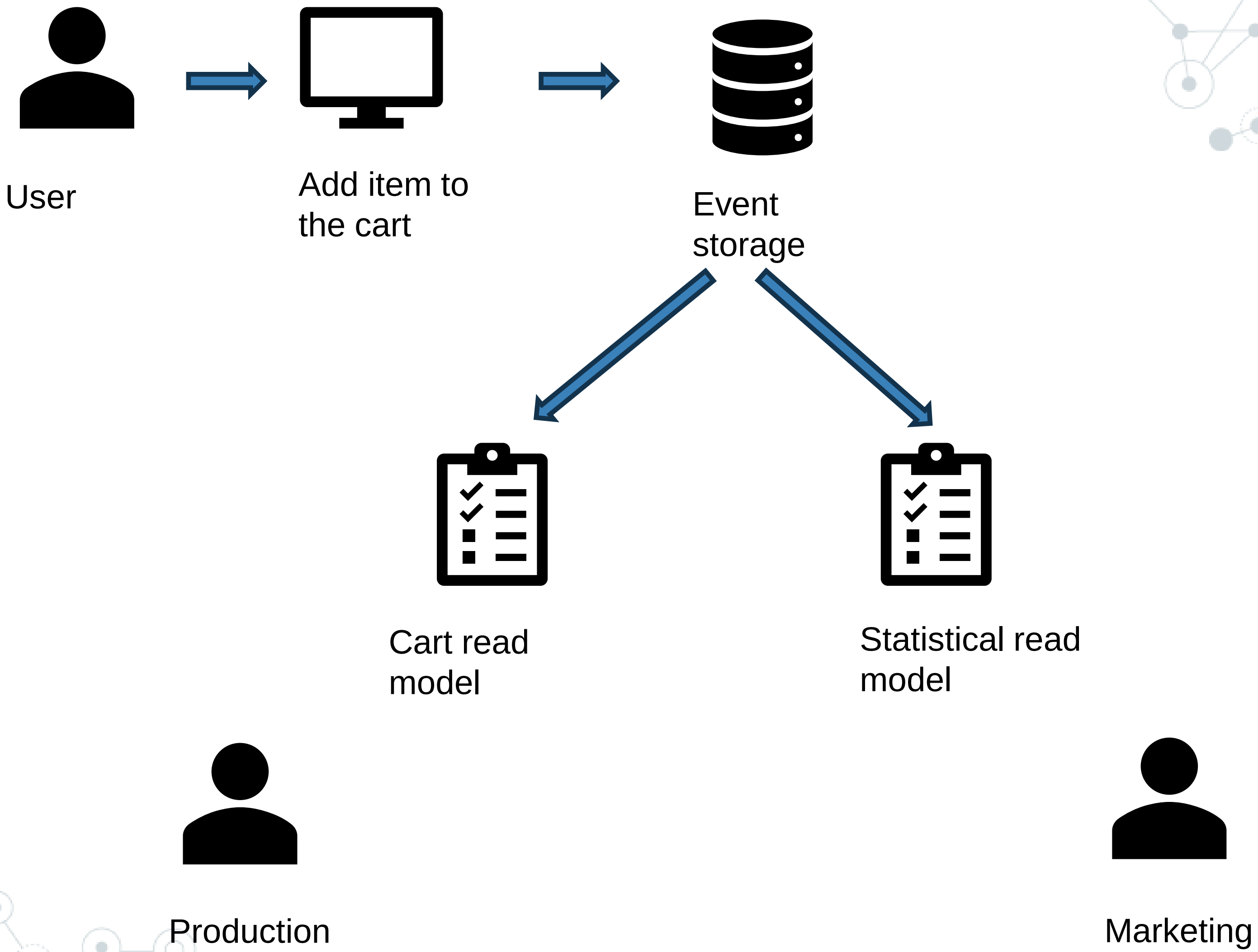
Performance

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/