Microservices

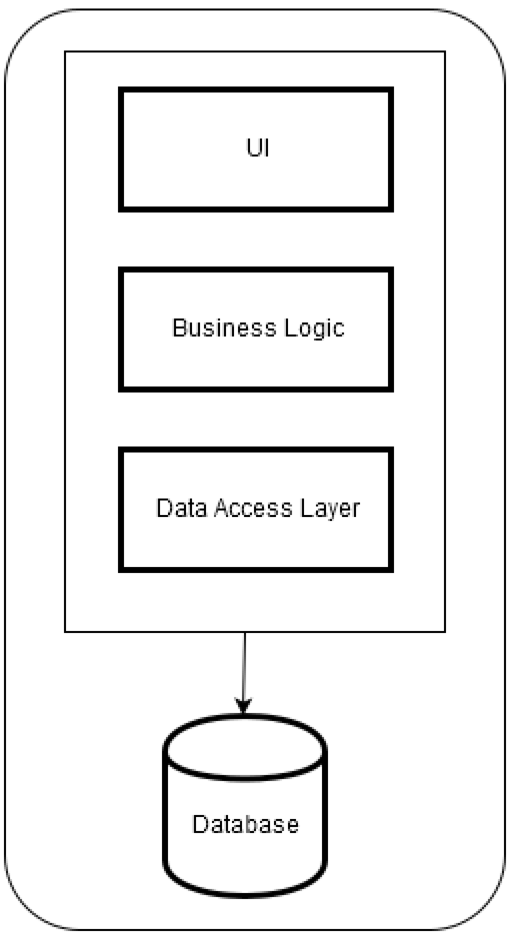
The three principals of the microservices architecture style.

* Microservices are ideal for big systems
* Microservice architecture is goal-oriented not solution-oriented
* Microservices are focused on replaceability

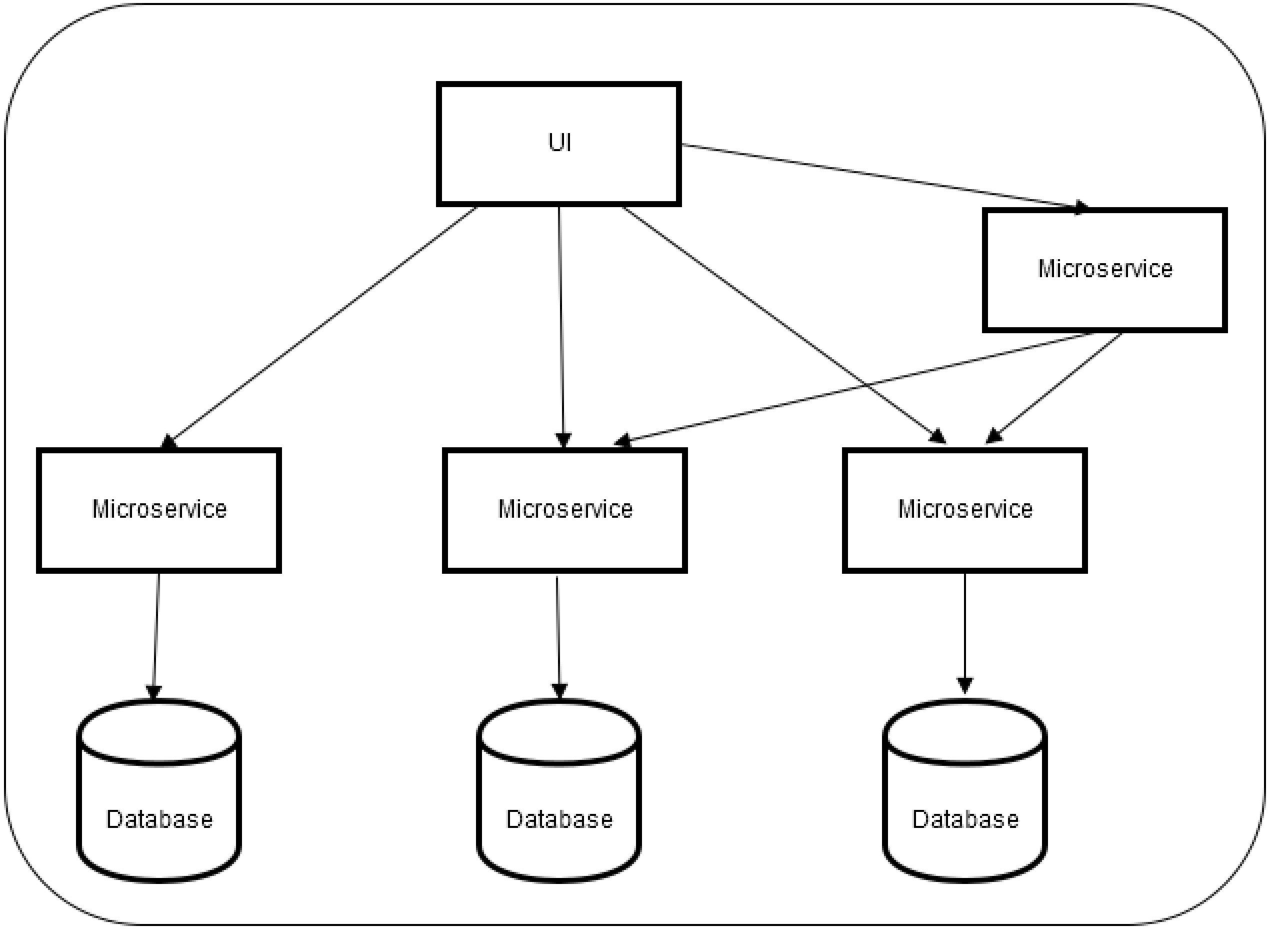
microservice applications share some important characteristics

- Small in size
- Messaging enabled
- Bounded by contexts
- Autonomously developed
- Independently deployable
- Decentralized
- Built and released with automated processes

Web Service vs Microservice



Monolithic Architecture



Microservices Architecture

Web Service

Web Service is a way to expose the functionality of an application to other application, without a user interface. It is a service which exposes an API over HTTP.

Web Services allow applications developed in different technologies to communicate with each other through a common format like XML, Json, etc.

Microservices

Micro Service is independently deployable service modeled around a business domain. It is a method of breaking large software applications into loosely coupled modules, in which each service runs a unique process and communicates through APIs. It can be developed using messaging or event-driven APIs, or using non-HTTP backed RPC mechanisms.

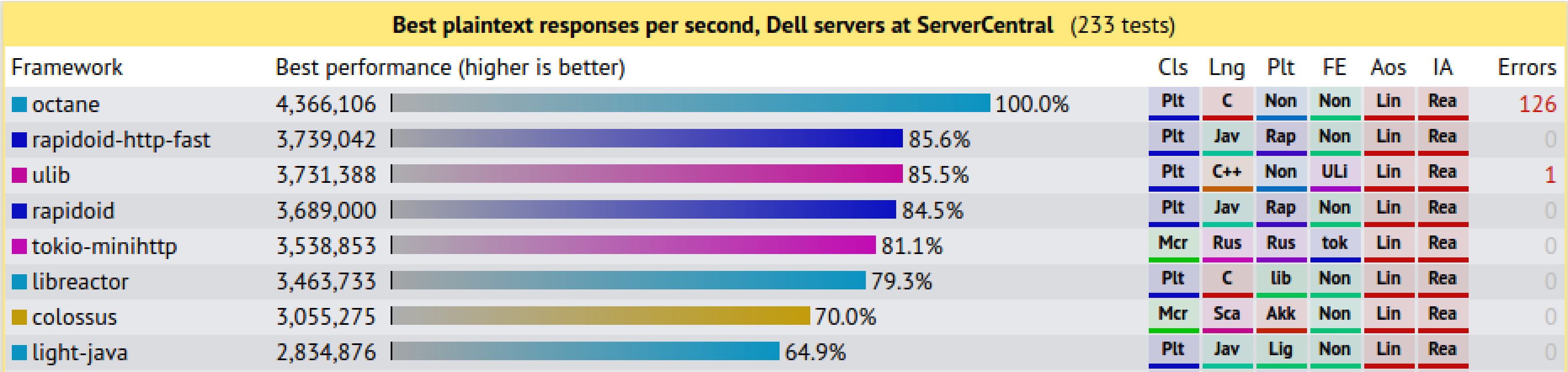
Difference

Microservice: Non-Java EE based that can address cross-cutting concerns in request/response chain.

Webservice: Java EE based that cannot manipute immutable request and response. All cross-cutting concerns must be addressed at network level. Commercial gateway.

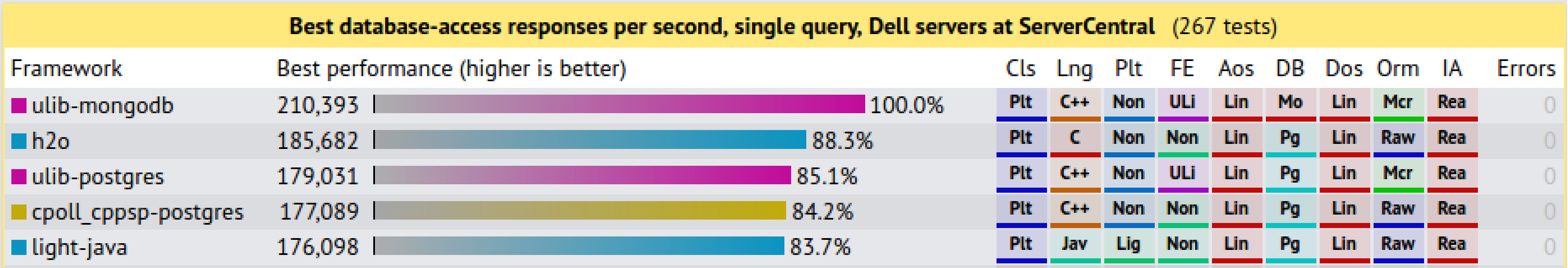
Difference

- Microservice: low latency, high throughput and small memory footprint.
- Webservice: high latency, low throughput and big memory footprint.



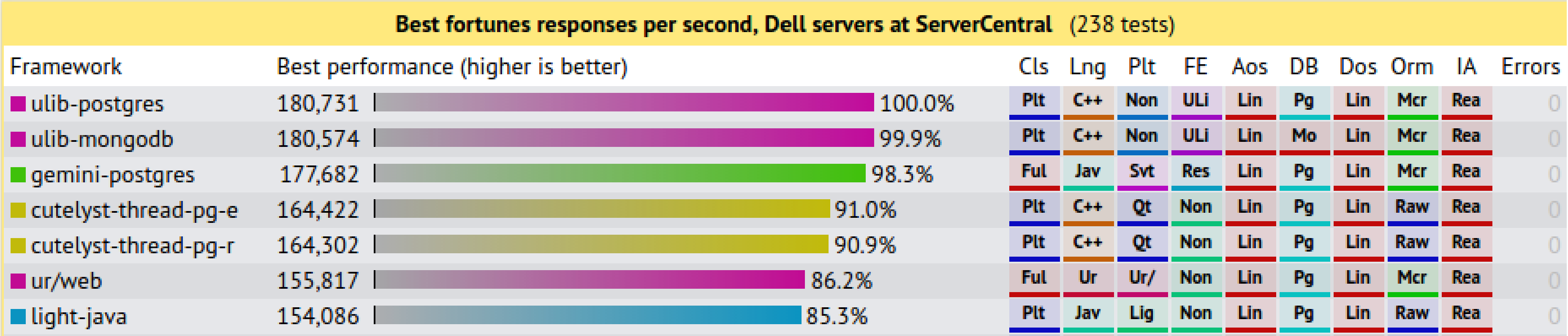
Best plaintext responses per second, Dell servers at ServerCentral (233 tests)

Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	IA	Errors
octane	4,366,106	100.0%	Plt	C	Non	Non	Lin	Rea	126
rapidoid-http-fast	3,739,042	85.6%	Plt	Jav	Rap	Non	Lin	Rea	0
ulib	3,731,388	85.5%	Plt	C++	Non	ULi	Lin	Rea	1
rapidoid	3,689,000	84.5%	Plt	Jav	Rap	Non	Lin	Rea	0
tokio-minihttp	3,538,853	81.1%	Mcr	Rus	Rus	tok	Lin	Rea	0
libreactor	3,463,733	79.3%	Plt	C	lib	Non	Lin	Rea	0
colossus	3,055,275	70.0%	Mcr	Sca	Akk	Non	Lin	Rea	0
light-java	2,834,876	64.9%	Plt	Jav	Lig	Non	Lin	Rea	0



Best database-access responses per second, single query, Dell servers at ServerCentral (267 tests)

Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA	Errors
ulib-mongodb	210,393	100.0%	Plt	C++	Non	ULi	Lin	Mo	Lin	Mcr	Rea	0
h2o	185,682	88.3%	Plt	C	Non	Non	Lin	Pg	Lin	Raw	Rea	0
ulib-postgres	179,031	85.1%	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea	0
cpoll_cppsp-postgres	177,089	84.2%	Plt	C++	Non	Non	Lin	Pg	Lin	Raw	Rea	0
light-java	176,098	83.7%	Plt	Jav	Lig	Non	Lin	Pg	Lin	Raw	Rea	0



Best fortunes responses per second, Dell servers at ServerCentral (238 tests)

Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA	Errors
ulib-postgres	180,731	100.0%	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea	0
ulib-mongodb	180,574	99.9%	Plt	C++	Non	ULi	Lin	Mo	Lin	Mcr	Rea	0
gemini-postgres	177,682	98.3%	Ful	Jav	Svt	Res	Lin	Pg	Lin	Mcr	Rea	0
cutelyst-thread-pg-e	164,422	91.0%	Plt	C++	Qt	Non	Lin	Pg	Lin	Raw	Rea	0
cutelyst-thread-pg-r	164,302	90.9%	Plt	C++	Qt	Non	Lin	Pg	Lin	Raw	Rea	0
ur/web	155,817	86.2%	Ful	Ur	Ur/	Non	Lin	Pg	Lin	Mcr	Rea	0
light-java	154,086	85.3%	Plt	Jav	Lig	Non	Lin	Pg	Lin	Raw	Rea	0

Microservice Security Is Harder

- Win

Every service only has access to what it needs to perform its function

- Lose

Much larger attack surface(internal threats)

How do other services know who's accessing them?

How can other services trust each other?

OAuth2

Delegated Authorization

- A protocol for conveying authorization decisions via a token
- Standard means of obtaining a token (aka 4 OAuth2 grant types)

Authorization Code

Resource Owner Password Grant

Implicit

Client Credentials

- Users and Clients are separate entities

I am authorizing this app to perform these actions on my behalf

What is OAuth2 Not

OAuth2 is not Authentication

- The user must be authenticated to obtain a token
- How the user is authenticated is outside of the spec
- How the token is validated is outside the spec
- What the token contains is outside the spec.

What is OpenID Connect

Delegated Authentication

- A protocol for conveying user identity via a signed JWT
- Built on top of OAuth2
- Standard means of obtaining an ID token
- Standard means of verifying ID token
- Steve is authorizing this app to perform these actions on his behalf and here is his email and role in case you need it

Beyond OAuth2: End to End Microservice Security

- Token Propagation
- ID token and Client Credentials token
- New Tokens via Token Exchange
- Data Integratity
- Data Confidentiality

Token Propagation

- The token is too powerful
- Can be used to do anything to the system as that user until it expires
- Token leakage is a big deal
- Internal fraud is easy

ID token and CC token

- A program fooled into misusing its authority

e.g. when one app fully trusts another by virtue of the app's identity

- A and B fully trust Resource

Where's the proof that Resource is acting faithfully?

How can A and B know if the User is actually authorized?

- If Resource is compromised, A and B will be compromised

Total trust in the bank

- If apps fully trust one another, do teams as well?

Transitively, perhaps

Do banks really work that way?

- What happens when you have no trust boundaries?

You don't check the other person's work

You allow a single person to perform multiple critical tasks

You have no separation of duties

Insiders

- Most attackers are insiders. Over 60%.

Confused deputy mitigations

- Authorize based on more than just caller's identity

The user and their scopes

- Send both client and user's token

Still vulnerable, the combination is not integrity protected

- Authorize based on a composite token
- Authorize based on a call stack

No information loss

Specify allowable behaviours with high precision

New Tokens via Token Exchange

- Given Actor + Subject + Audience, get a new token

Policy decision given caller, user and intent

New token expresses caller, user and intent

- Given Actor + previous token + Audience, get a new token

Policy decision based on delegation chain(call stack)

Policy decision based on aggregated trust

Token Exchange

- Pros

User, client and call stack

Narrow audience and scope

Trust boundaries are unambiguous

Centralized policy management and de-centralized policy enforcement

- Cons

Network and AS overhead

Security vs Performance

Policy Management vs Agility

HTTPS doesn't solve message security

- Does Cart really need to see the CC info?

Payment is the only service that really use it

- Only Point to Point confidentiality and integrity are assured
- Rearrange services so Payment can be directly called by Shop

Limits architectural choise

Overcomplicates Shop

End to End Message Security

- Cannot be solved at the network layer
- Necessarily an application layer concern
- Stop assuming trust based on solely on caller's identity
- Limit the effect of token leakage and misuse
- Break out of the performance vs security tradeoff

JOSE for Message Security

- Use public/private key pairs
- Sender signs message with private key

Integrity and non-repudiation

- Sender encrypts signed messages with public key of recipient

Confidentiality

- Recipient decrypts with its private key, verifies with sender's public key

Key Pairs

- Public Key Infrastructure(PKI)
- Certificate Authority
- OAuth2 Server

Oauth2 and Serivce Discovery

- Oauth2 server generates key pairs
- Service first time start to download key pair with client_id and client_secret
- Service register host, port and public key to service discovery (consul)
- Other services can find public key of other service from discovery

Sign and Verify Data

- The sender signs with its private key

HTTP request, response, headers

Messaging body

- Receiver users the registry to get sender's public key
- Use the public key to verify payloads
- Payloads are traceable to the individual service that registered

Encrypt and Decrypt Data

- Sender uses the registry to get receiver's public key
- Use public key to encrypt payloads

sign, then encrypt

multiple receivers possible via JWE JSON Serialization

encrypt entire bodies, objects or signle field

- Use it when needed

Distributed OAuth2

- End to End message confidentiality with JWE
- Message integrity via JWS

Service authentication for free

- Still need to deal with authorization

Assert and authorize call stack

Limit the effect of token leakage and misuse

Performance vs security tradeoff

Self Issued JWT

- Services authenticate with JWTs they create and sign themselves

JWT is just a specific use case of JWS

Services authenticate by verifying with sender's public key

- Create as many as you need without network overhead

Signle Use JWT

- Short expiry
- JTI can prevent replay attacks
- Unambiguous intent

Express the intented recipient and operation to be performed

- Very limited power

Can only be used for intended purpose and only once

- Services only accept these JWTs

Reject the bear user token as it has no power on its own

Nested, Self Issued JWT

- Incoming JWTs can be nested in another JWT for use downstream

Call stack expressed as nested JWTs

- Verifiable chain of custody

Like an audit trail

Call stack verified by verifying each JWT recursively

- Unbreakable chain of custody

Sender and receiver is encoded in the chain

Chains are wrapped in another JWS, with doesn't get propagated

Chain truncation can be detected

Extenalized Policy, Embedded Decision Making

- Services make authorization decisions on their own
- Policy as externalized configuation

Not hard-coded into services

- No additional calls required for authorization

Good performance

- Policy can be flexible

Allow services to evolve