DIT 345 Fundamentals of Software Architecture

Lecture 3: Utility Trees and Architectural Styles

Rebekka Wohlrab

Assignment 1 Tips: Architectural Impact

- Task: “For each constraint, write a short one-sentence description of the impact it will have on the architecture of the system.”
  - Be careful that you really describe the architectural impact. Don’t write: “We need a large market share because it is important that the company makes profit.”
  - Ok: ”Our application needs to interact with a COBOL system that is developed in-house, which impacts the architectural design of our interfaces and modifiability of the architecture.”

- Make sure to include a metric for the quality requirements

Architecturally significant requirements

- We only want to capture the most important quality attribute requirements.
- We call a subset of our QARs **architecturally significant** if they have:
  - A profound impact on the architecture.
    Which means that including this requirement will very likely result in a different architecture than if it were not included.

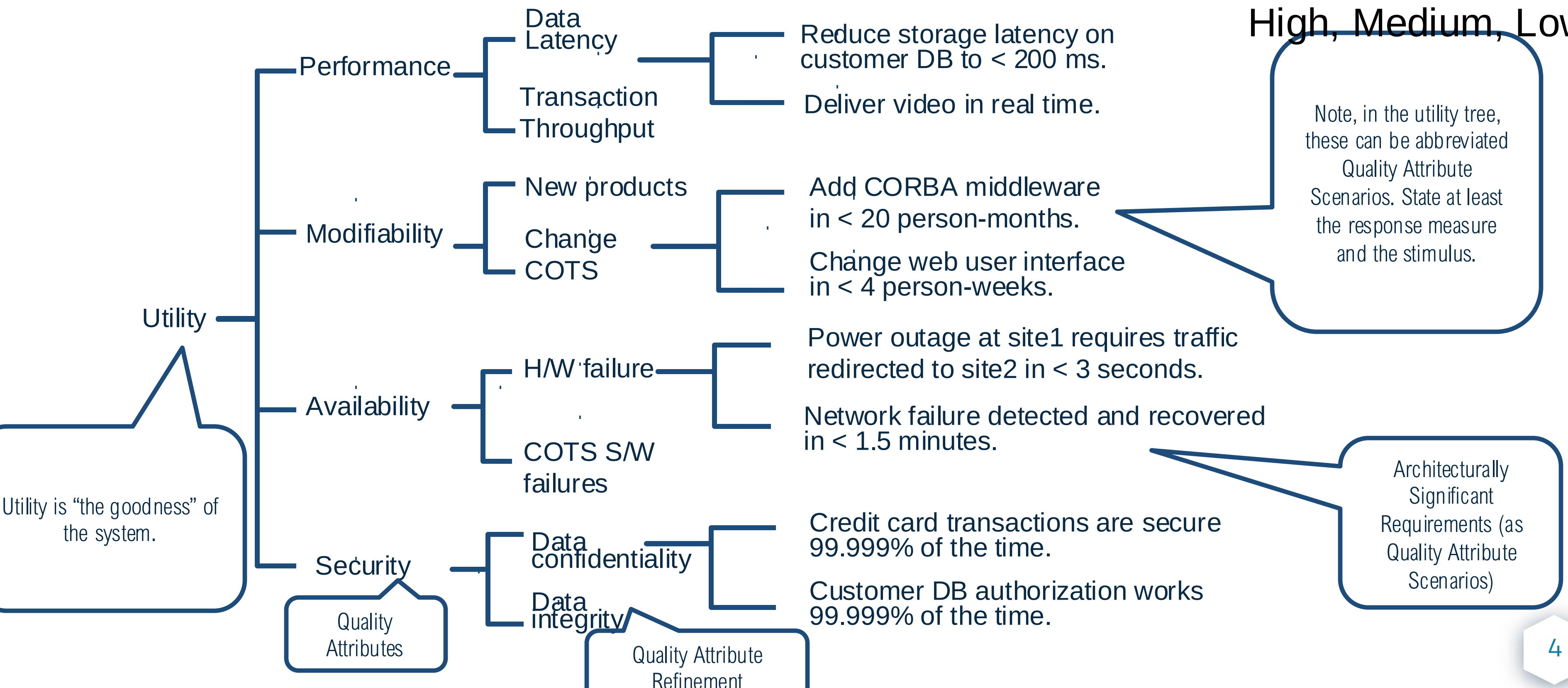
  and

  - A high business or mission value.
    Which means that if the architecture is going to satisfy this requirement—potentially at the expense of not satisfying others—it must be of high value to important stakeholders.

- Need to create a "catalog" of requirements – in the form of a utility tree!

Organizing QAs: Utility Trees

(H,M) --- business value, architectural
High, Medium, Low


Utility
Performance
Data Latency
Transaction Throughput
Reduce storage latency on customer DB to < 200 ms.
Deliver video in real time.
Modifiability
New products
Change COTS
Add CORBA middleware in < 20 person-months.
Change web user interface in < 4 person-weeks.
Availability
H/W failure
COTS S/W failures
Power outage at site1 requires traffic redirected to site2 in < 3 seconds.
Network failure detected and recovered in < 1.5 minutes.
Security
Data confidentiality
Data integrity
Credit card transactions are secure 99.999% of the time.
Customer DB authorization works 99.999% of the time.

Note, in the utility tree, these can be abbreviated Quality Attribute Scenarios. State at least the response measure and the stimulus.

Architecturally Significant Requirements (as Quality Attribute Scenarios)

Utility is "the goodness" of the system.

Quality Attributes

Quality Attribute Refinement

High, Medium, Low ranking (p. 306)

- **High** designates a must-have requirement, **Medium** is for a requirement that is important but would not lead to project failure were it omitted. **Low** describes a nice requirement to have but not something worth much effort.
- For architectural impact, **High** means that meeting this ASR will profoundly affect the architecture. **Medium** means that meeting this ASR will somewhat affect the architecture. **Low** means that meeting this candidate ASR will have little effect on the architecture.

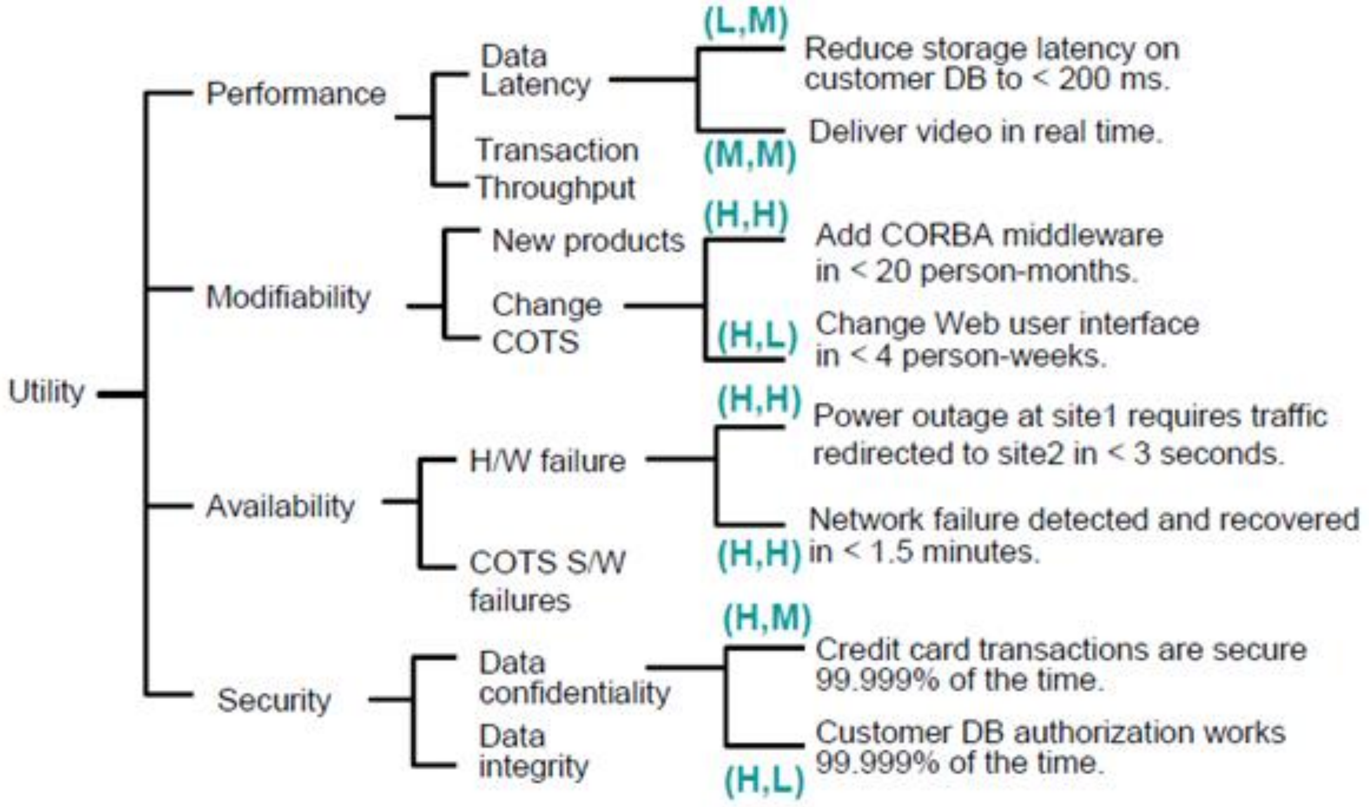
Evaluating Business Value

- Need to elicit stakeholders' preferences for quality attributes
- Need to understand business impact of requirements
- Ask questions:
  - "What could potentially occur if this requirement isn't fulfilled?"
  - "In what ways does the execution of this requirement contribute value to the project?"

- High, Medium, Low: Rank Business Value and Architectural Impact

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Organizing QAs: Utility Trees


Utility
Performance
Data Latency
Transaction Throughput
(L,M)
Reduce storage latency on customer DB to < 200 ms.
Deliver video in real time.
(M,M)
Modifiability
New products
Change COTS
(H,H)
Add CORBA middleware in < 20 person-months.
(H,L)
Change Web user interface in < 4 person-weeks.
Availability
H/W failure
COTS S/W failures
(H,H)
Power outage at site1 requires traffic redirected to site2 in < 3 seconds.
Network failure detected and recovered in < 1.5 minutes.
(H,H)
Security
Data confidentiality
Data integrity
(H,M)
Credit card transactions are secure 99.999% of the time.
Customer DB authorization works 99.999% of the time.
(H,L)

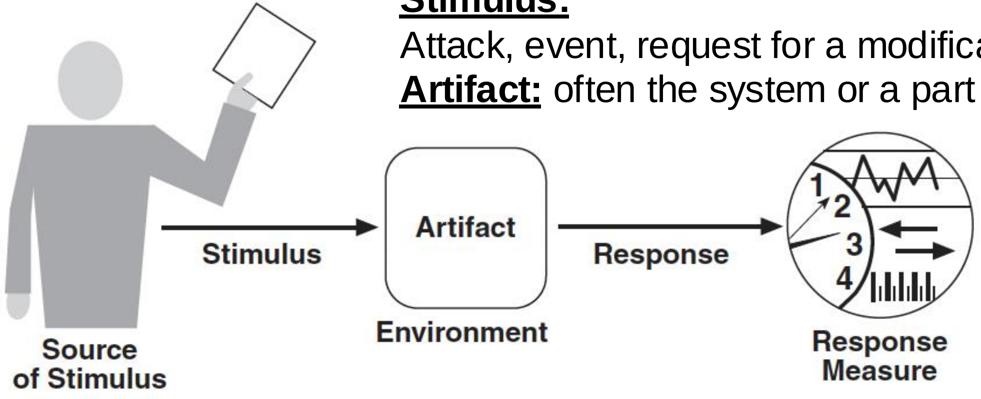
High, Medium, Low: Rank Business Value and Architectural Impact

What is the point of Quality Attribute Scenarios?

- Want to be able to test quality attribute requirements – need precise and contextual information
- Example of an imprecise requirement: “A user request should be answered within 2s.”
- Somewhat measurable - but in what context should this requirement hold?
- Some QAs are relevant at run time, others are relevant for the development of the system

Source:
Actor, part of the system, processor, system

Stimulus:
Attack, event, request for a modification, ...
Artifact: often the system or a part of a system



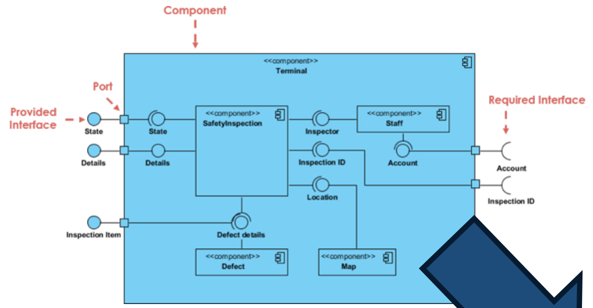
Response measure:
Latency, throughput, number of hours, ...

Now we discuss how you can actually improve QAs

Requirements

(Quality Attribute Scenarios, architecturally significant requirements

Software Architecture



Code

Architectural Styles

- Recurring forms have been widely observed in different systems.
- These forms are worth capturing because they have known properties and can be re-used.

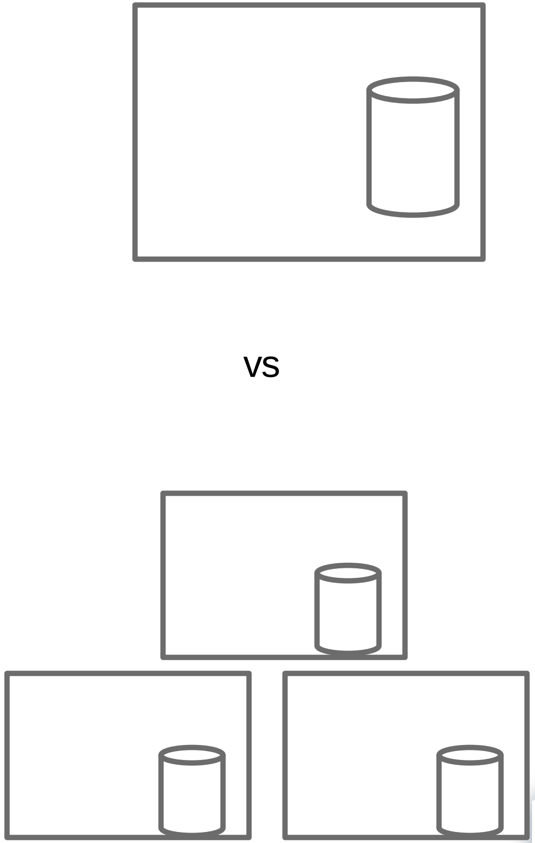
- We call these forms styles:
  An architectural style is a set of element and relation types, together with a set of constraints on how they can be used.

Architectural Styles

1. Layered style
2. Pipe-and-filter style
3. Service-oriented style
4. Publish-subscribe style
5. Peer-to-peer style
6. Microservice style

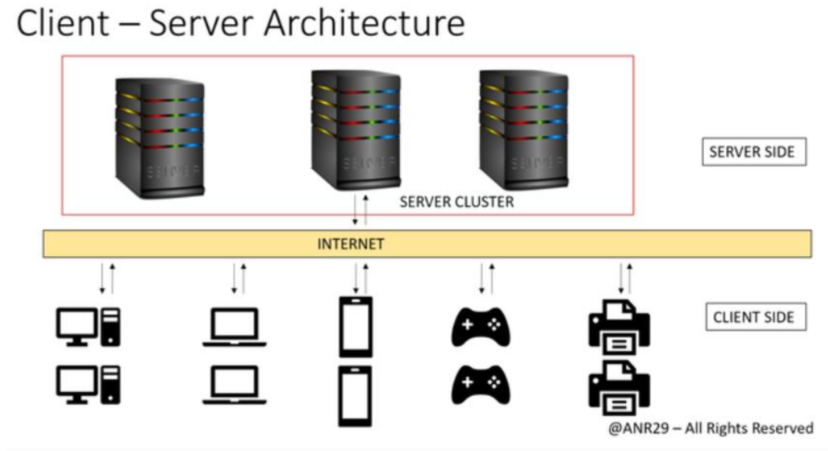
Architecture quanta

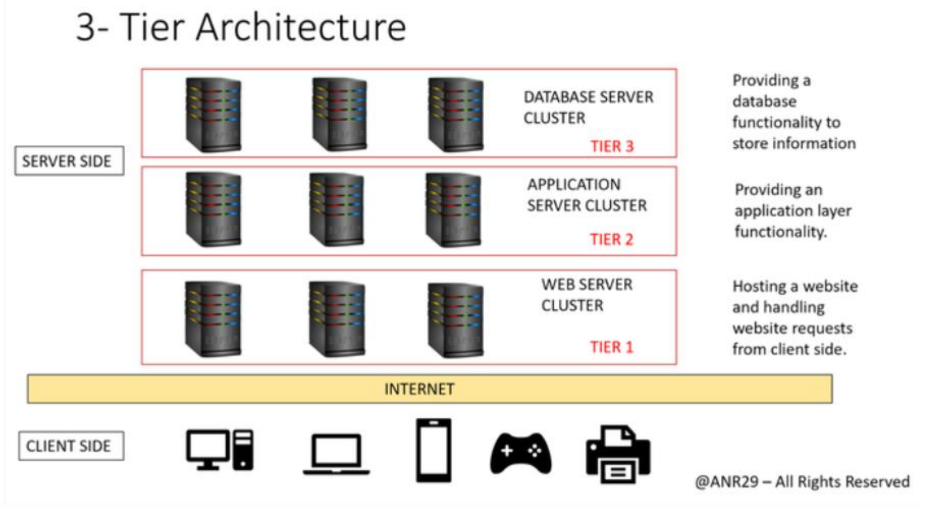
- Quantum: An independently deployable artifact with high functional cohesion
- An architecture quantum includes all the necessary components to function independently from other parts of the architecture.
    - If an application uses a database, it is part of the quantum because the system won't function without it.
- Most legacy systems deployed using a single database by definition have a quantum of one.
- However, in the distributed microservices architecture style, each service includes its own database creating multiple quanta (Q = #MS) within that architecture.



1- Layered Architecture style

Topology Flavours



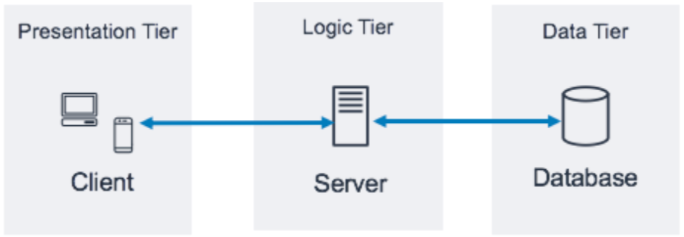

3- Tier Architecture
SERVER SIDE
DATABASE SERVER CLUSTER
TIER 3
Providing a database functionality to store information
APPLICATION SERVER CLUSTER
TIER 2
Providing an application layer functionality.
WEB SERVER CLUSTER
TIER 1
Hosting a website and handling website requests from client side.
INTERNET
CLIENT SIDE
@ANR29 – All Rights Reserved

CLIENT — SERVER ARCHITECTURE

1- Layered Architecture style

Topology

- Want: Separation of concerns, so that we have easier portability, modifiability, and reuse.
- This style of architecture is the de facto standard for most applications, primarily because of its simplicity, familiarity, and low cost.
- In most organizations there are user interface (UI) developers, backend developers, rules developers, and database experts (DBAs). These organizational layers fit nicely into the tiers of a traditional layered architecture.



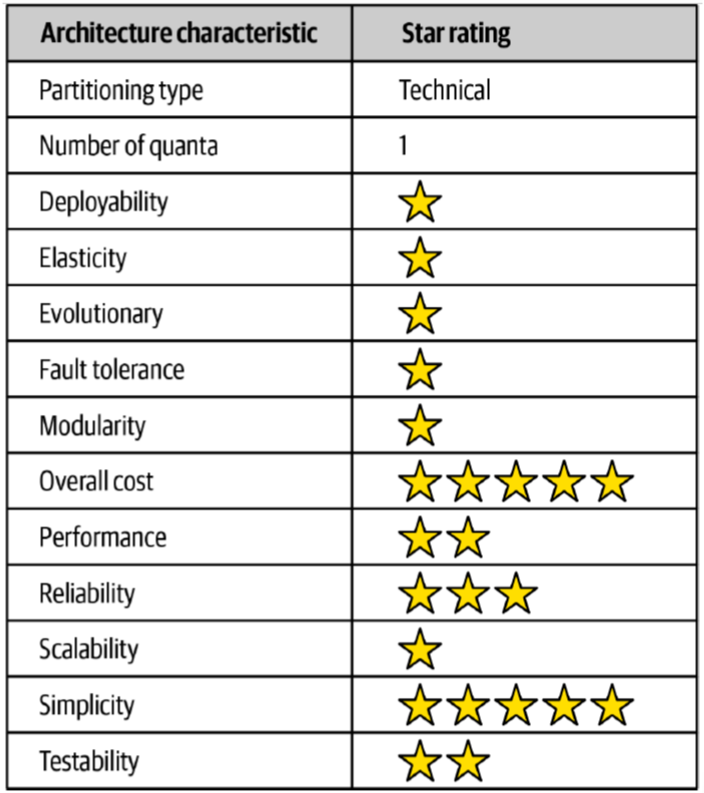
1- Layered Architecture Style

Layered Style Rating

The layered architecture style is a good choice for:

- Small, simple applications or websites.
- As a starting point, with very tight budget and time constraints. The layered architecture is perhaps one of the lowest-cost architecture styles, promoting ease of development for smaller applications.
- The layered architecture style is also a good choice when an architect is still analyzing business needs and requirements.

a five-star rating means the architecture characteristic is one of the strongest features in the architecture style.

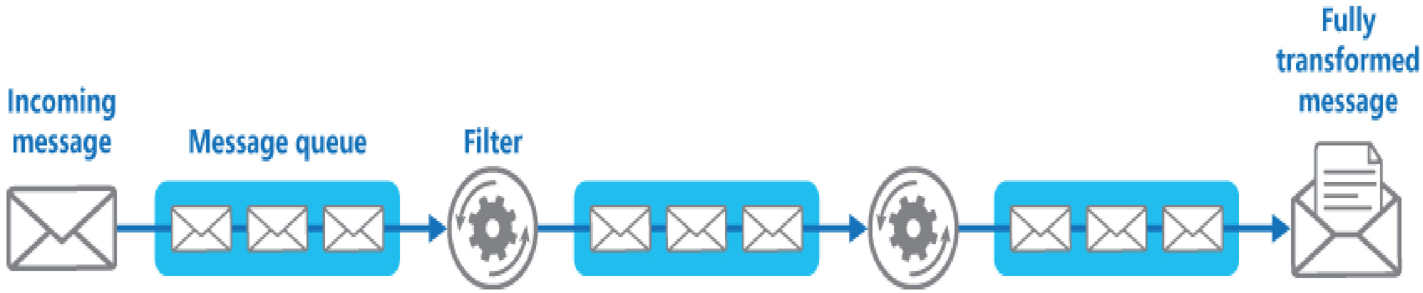


Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★
Elasticity	★
Evolutionary	★
Fault tolerance	★
Modularity	★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★

Figure 10-6. Layered architecture characteristics ratings

2- Pipeline Architecture style

- Need to process streams of discrete data items, from input to output. Many types of transformations occur repeatedly in practice, and so it is desirable to create these as independent, reusable parts.



2- Pipeline Architecture style

Pipes

It is the communication channel between filters. Each pipe is typically unidirectional and point-to-point (no broadcast) for performance reasons, accepting input from one source and always directing output to another. Architects favor smaller data pipes payload to enable high performance.

Filters

Filters are self-contained, independent from other filters, and generally stateless. Filters should perform one task only. Composite tasks should be handled by a sequence of filters rather than a single one.

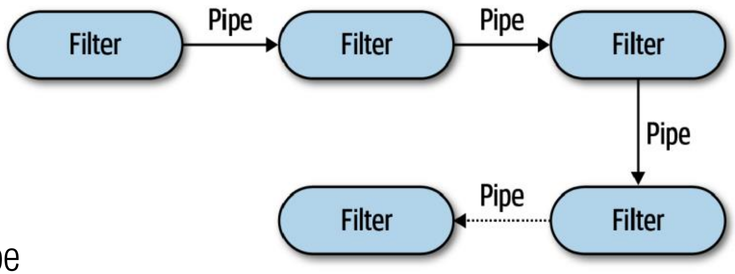
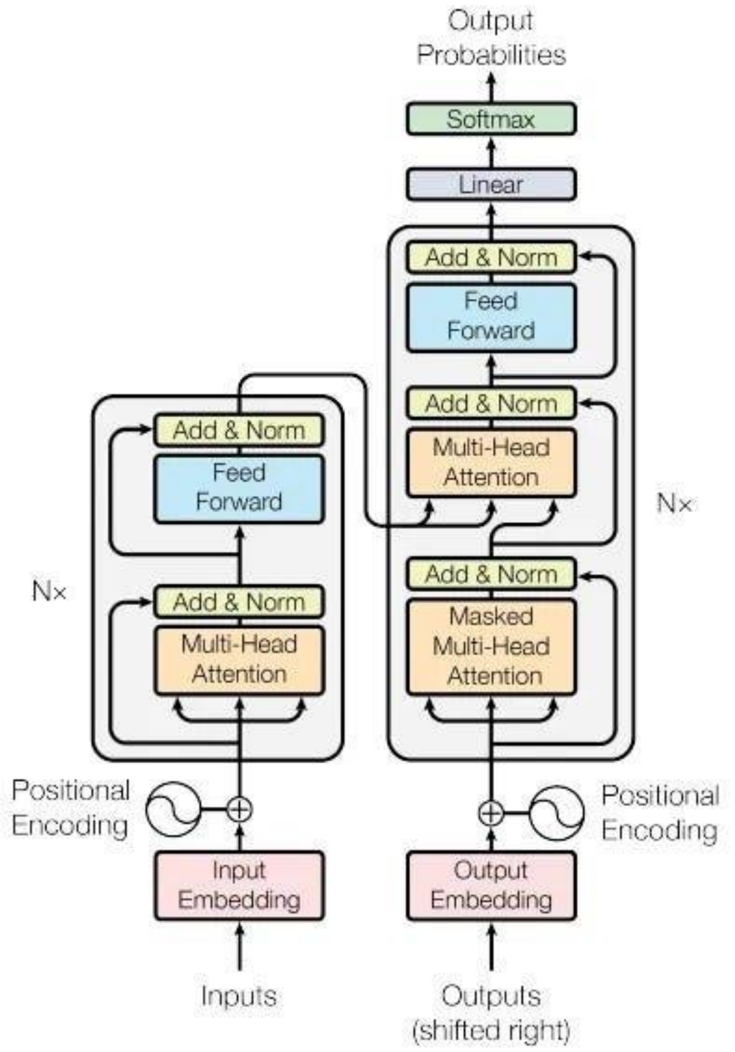


Figure 11-1. Basic topology for pipeline architecture

Transformer architecture

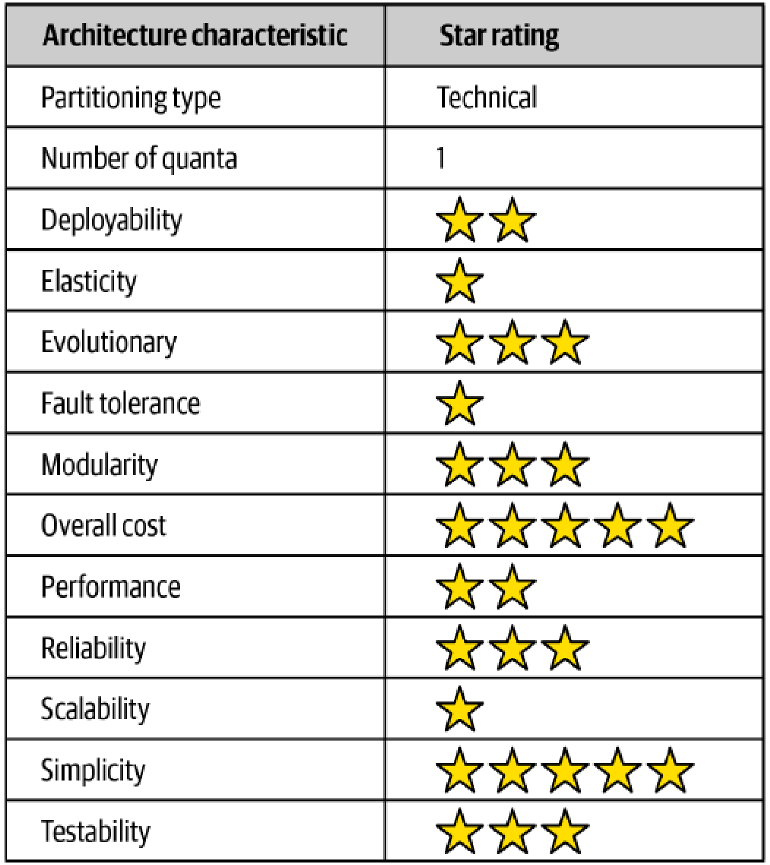


*Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Kaiser, Łukasz; Polosukhin, Illia (2017). Advances in Neural Information Processing Systems. **30**. Curran Associates, Inc.*

Pipeline Architecture Style

Pipeline Style Rating

- Because the pipeline architecture is usually implemented as a monolithic deployment, the architectural quantum is always one.
- Overall cost and simplicity combined with modularity are the primary strengths of the pipeline architecture style.
- Being monolithic in nature, pipeline architectures don’t have the complexities associated with distributed architecture styles, are simple and easy to understand, and are relatively low cost to build and maintain.



Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★★
Elasticity	★
Evolutionary	★★★
Fault tolerance	★
Modularity	★★★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★★

Figure 11-3. Pipeline architecture characteristics ratings

3- Service-Oriented Architecture style

Topology of Service-Oriented style

- **Idea: support interoperability** of distributed components running on different platforms and written in different implementation languages, provided by different organizations, and distributed across the Internet

- Solution: The service-oriented architecture (SOA) style describes a collection of distributed components that provide and/or consume services.

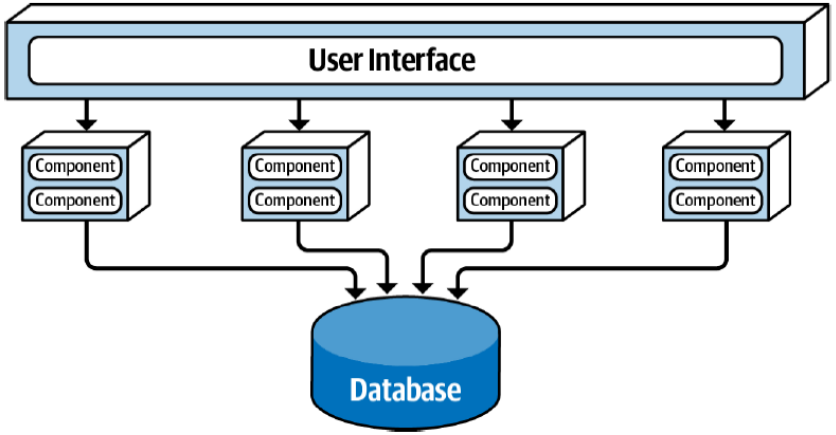


Figure 13-1. Basic topology of the service-based architecture style

Because the services typically share a single monolithic database, the number of services within an application generally range between 4 and 12 services, with the average being about 7 services.

3- Service-Oriented Architecture style

Topology of Service-Oriented style

- Services are accessed remotely from a user interface using a remote access protocol.
- Typical connectors: REST.
- Many topology variants exist within the service-oriented architecture style, making this perhaps one of the most flexible architecture styles.

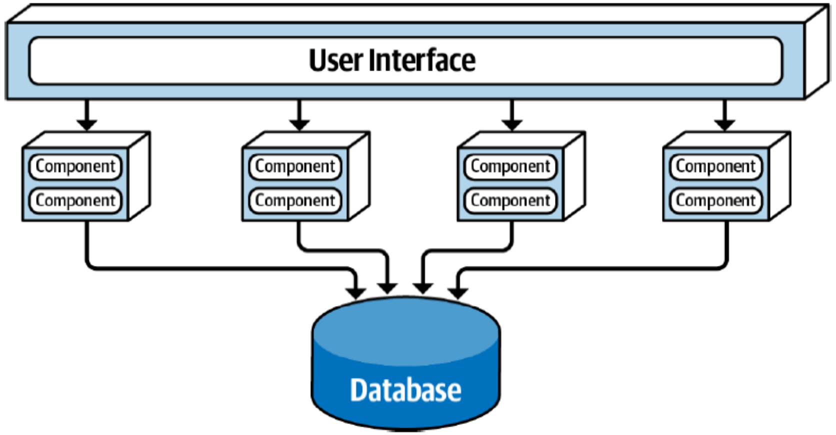


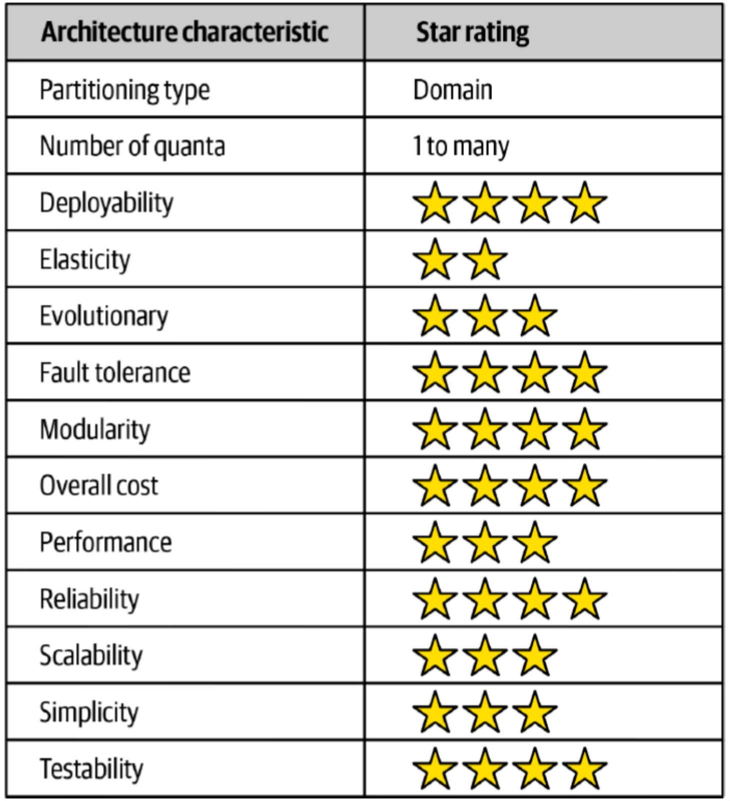
Figure 13-1. Basic topology of the service-based architecture style

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Service-Based Architecture style

Service-Based style Rating

- Service-based architecture is a domain-partitioned architecture, meaning that the structure is driven by the domain rather than a technical consideration (such as presentation logic or persistence logic).
- Being a distributed architecture, the number of quanta can be >= 1.
- Breaking apart an application into separately deployed domain services allows for fast change (agility), better test coverage due to the limited scope of the domain (testability), and more frequent deployments.
- Fault tolerance and overall application availability also rated high



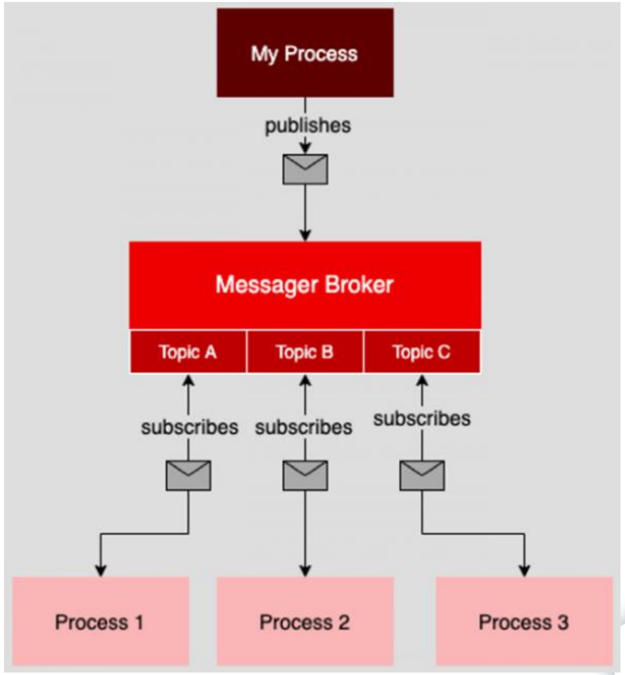
Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★
Evolutionary	★★★
Fault tolerance	★★★★
Modularity	★★★★
Overall cost	★★★★
Performance	★★★
Reliability	★★★★
Scalability	★★★
Simplicity	★★★
Testability	★★★★

Figure 13-9. Service-based architecture characteristics ratings

4- Publish-Subscribe Architecture Style

Topology

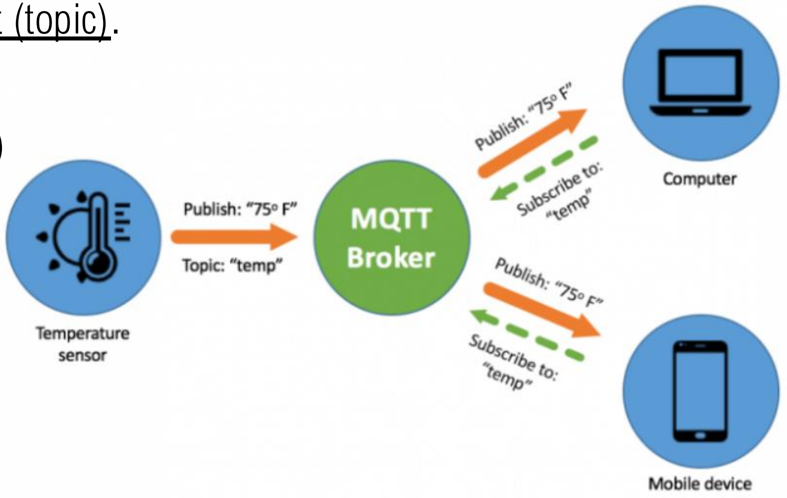
- The pub-sub style is an extensible real-time distributed style, the components interact via announced messages, or events, and components may subscribe to a set of events.
- It is the job of the publish-subscribe runtime infrastructure to make sure that each published event is delivered to subscribers of that event.
- Publisher components place events on the event bus by announcing them; the connector then delivers those events to the subscriber components that have registered an interest in the events.
- Subscribers let the publisher know they're interested, the publisher stores their addresses to know where to send which message.



Publish-Subscribe Architecture Style

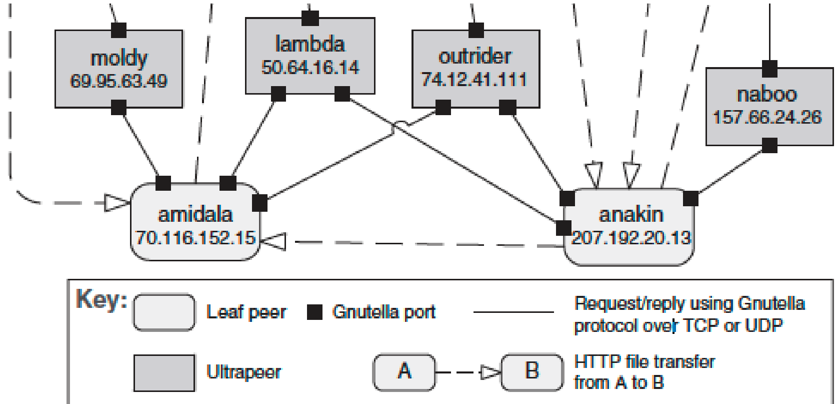
Topology

- Components interact via announced messages, or events.
  - Components may subscribe to a set of events.
  - It is the job of the publish-subscribe runtime broker to make sure that each published event is delivered to all subscribers of that event (topic).
- Advantages: loose coupling, scalability, modifiability, improved security (messages sent to subscribers only)
- Limitations: need to guarantee delivery, performance problems when overloaded with messages



5- Peer-to-Peer Style

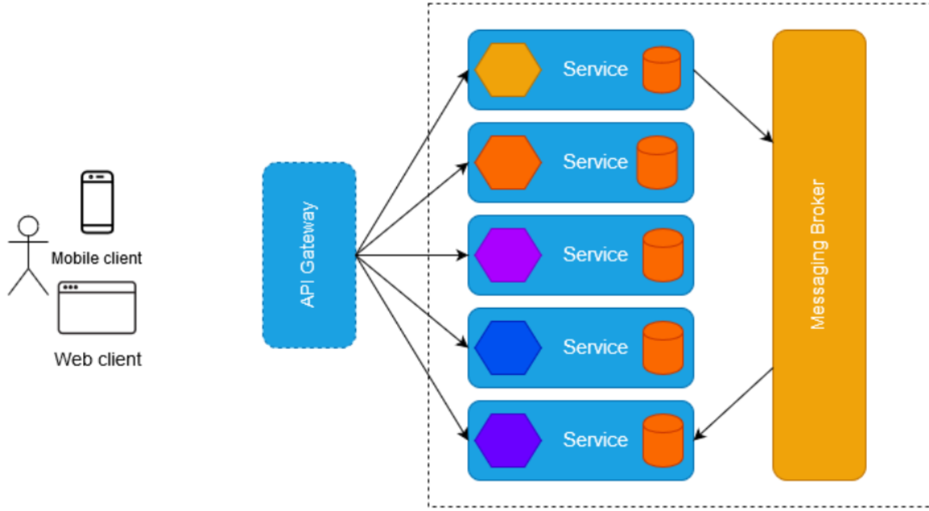
- Peers request services from and provide services to one another across a network.
  - Peer: an independent component running on a network node.
  - Request/reply connector, which is used to connect to the peer network, search for other peers, and invoke services from other peers.
- Weaknesses: Managing security, data consistency, data/service availability, backup, and recovery are all more complex.
- Small peer-to-peer systems may not be able to consistently achieve quality goals such as performance and availability.



6- Microservices Style

Topology

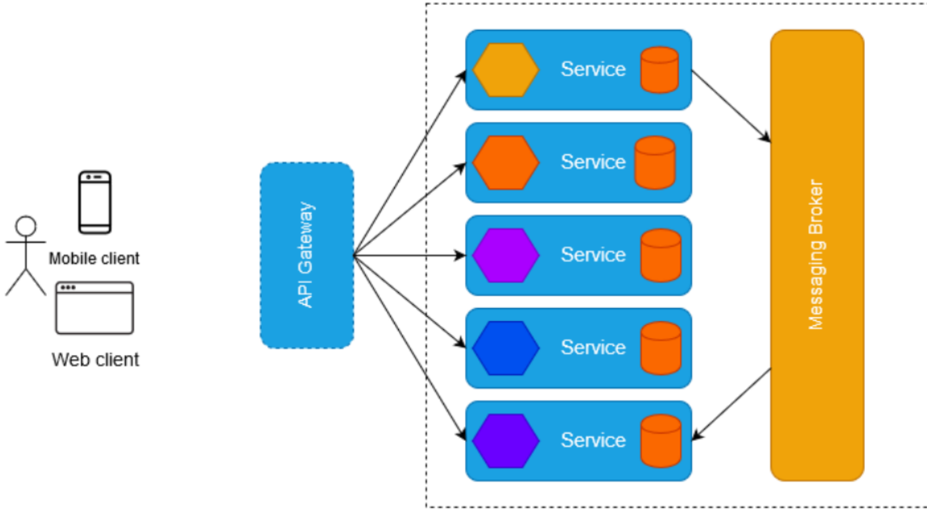
- Decoupling the services: Benefits of extreme decoupling, both at the domain and operational process level.



Microservices Style
Other Features

Data Isolation. Another requirement of microservices, driven by the bounded context concept, is data isolation. Many other architecture styles use a single database for persistence. However, microservices tries to avoid using all kinds of coupling, shared schemas and databases as integration points.

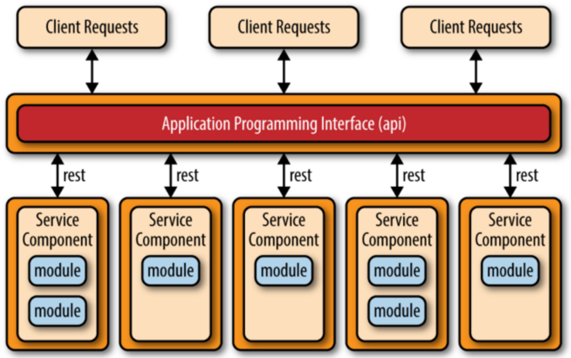
CI/CD of DevOps The evolutionary path from monolithic applications to a microservices architecture style was prompted primarily through the process of continuous integration and delivery.



Microservices Style

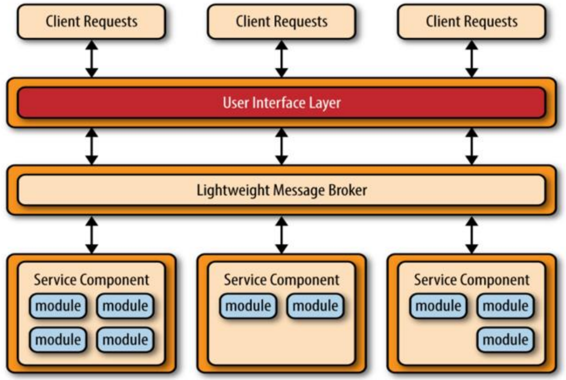
Interface

The User Interface is critical to the performance of Microservice design.
There are 2 different designs to deal with the APIs. This is suitable for remote web application environment.



REST (REpresentational State Transfer)

Traditional web-based or fat-client business application



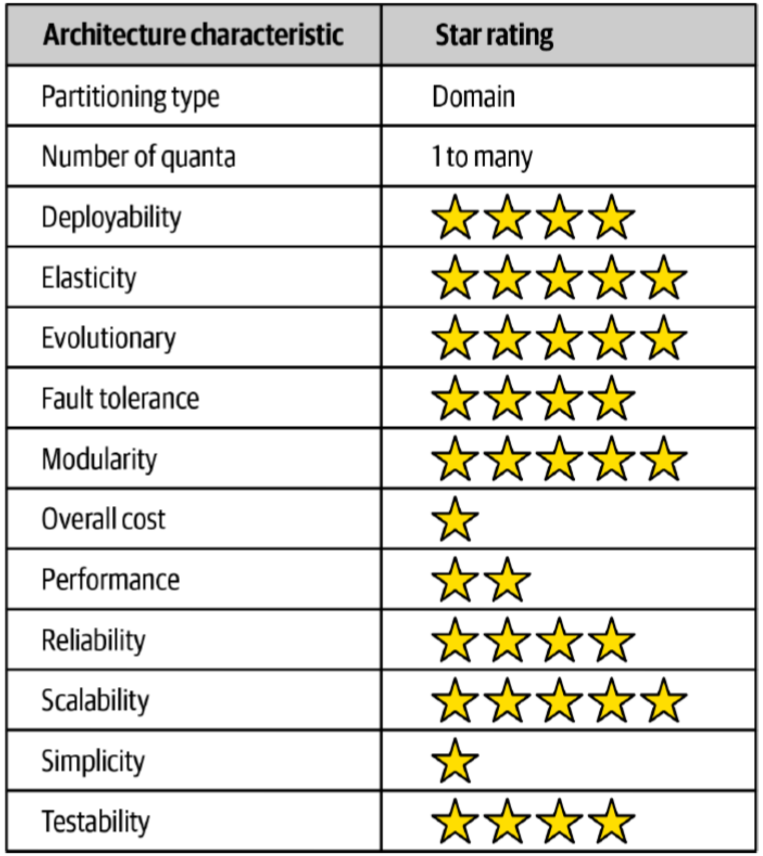
lightweight centralized message broker to access remote service components

Message brokers can validate, store, route, and deliver messages to the appropriate destinations. They serve as intermediaries between services. This facilitates decoupling of processes/services.

Microservices Style

Microservices Style Rating

- Microservices is decidedly a domain-centered architecture, where each service boundary should correspond to a domain.
- Fault tolerance and reliability are impacted when too much interservice communication is used. Independent, single-purpose services generally lead to high fault tolerance.
- High scalability, elasticity, and evolutionary systems utilized this style to great success.
- It also has the most distinct quanta of any modern architecture.



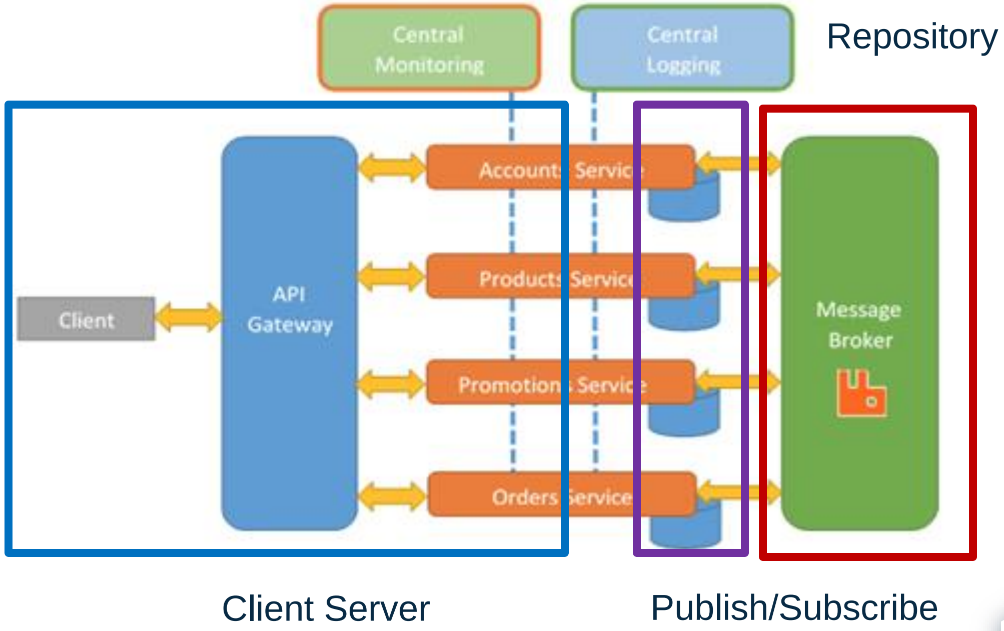
Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★★★★
Evolutionary	★★★★★
Fault tolerance	★★★★
Modularity	★★★★★
Overall cost	★
Performance	★★
Reliability	★★★★
Scalability	★★★★★
Simplicity	★
Testability	★★★★

Figure 17-13. Ratings for microservices

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

You can mix architectural styles

The microservice architectural style can be seen as a combination of client-server and publish/subscribe styles.



Central Monitoring
Central Logging
Repository
Client
API Gateway
Accounts Service
Products Service
Promotions Service
Orders Service
Message Broker
Client Server
Publish/Subscribe