Architecture Styles

Hönnun og smíði hugbúnaðar

Haust 2022

Architecture Characteristics

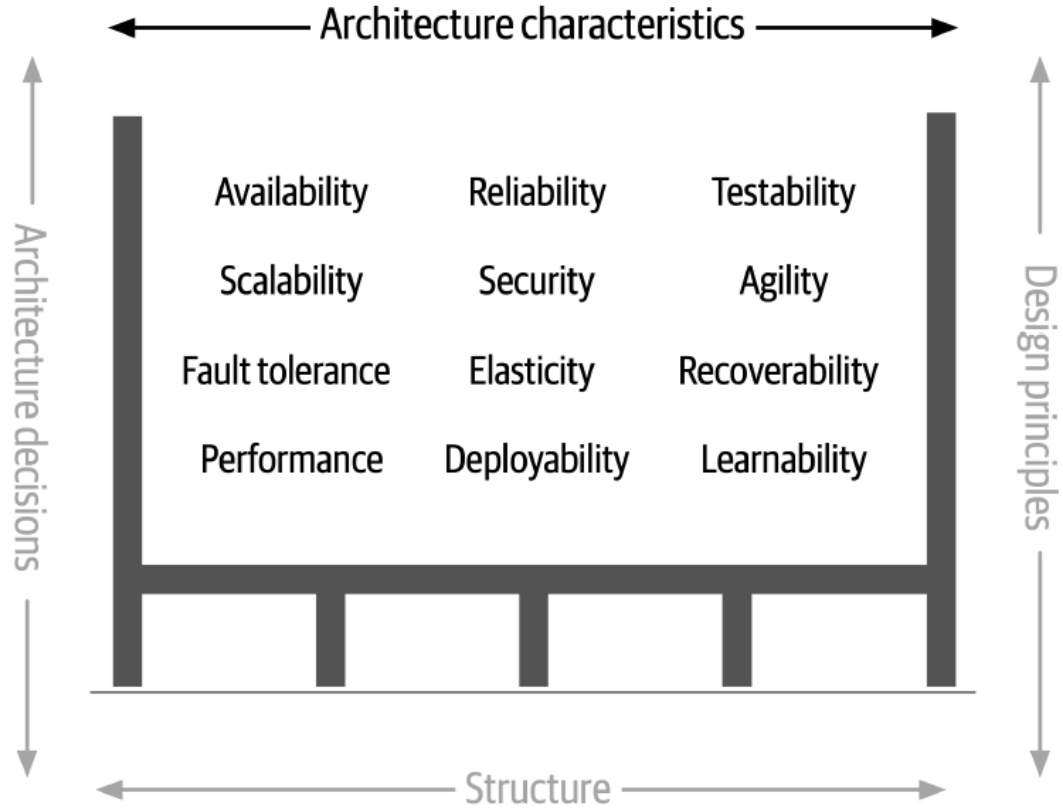


Figure 1-4. Architecture characteristics refers to the "-ilities" that the system must support

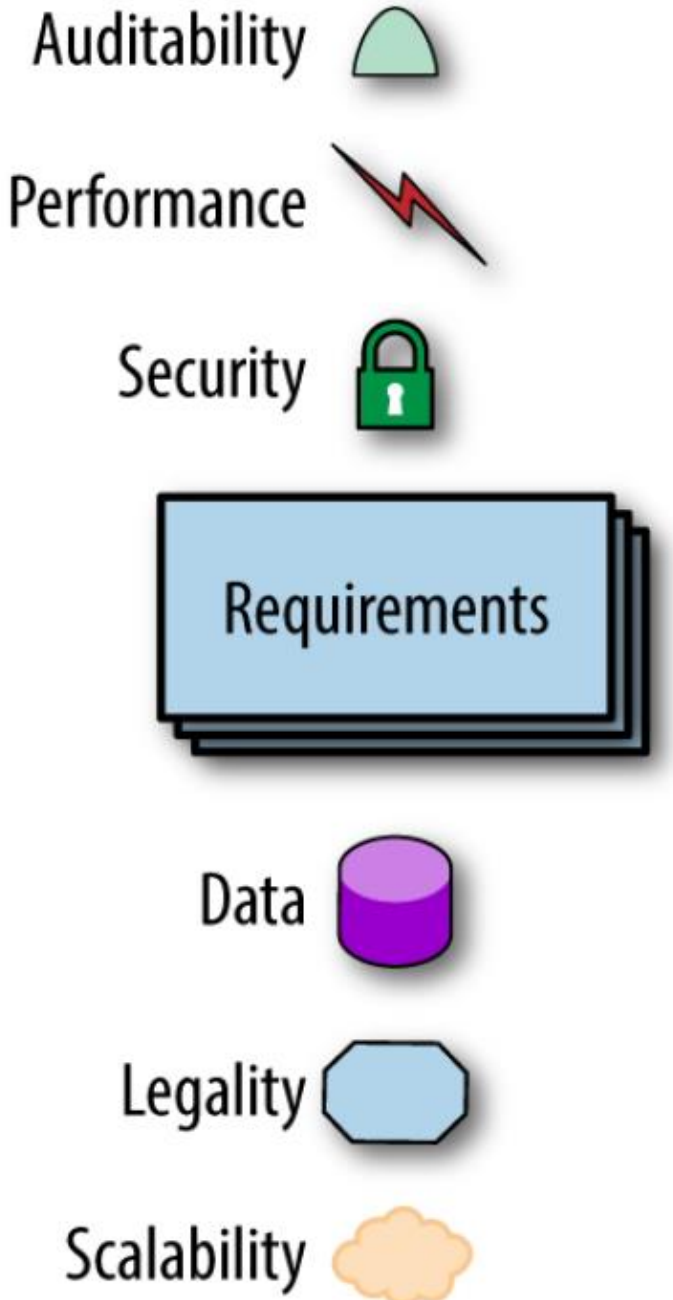


Figure 4-1. A software solution consists of both domain requirements and architectural characteristics

Hvað er Architecture Characteristic

- Samheiti
  - Nonfunctional requirements
  - Quality Attribute
- Uppfylla þrjú skilyrði
  - Tilgreinir non-domain hönnunar þátt
  - Hefur áhrif á structural þátt í hönnunninni
  - Er mikilvægt fyrir velgegni kerfisins
- Breitt róf af characteristics til
  - Nokkrar flokkanir:
    - Operational Architecture Characteristics
    - Structural Architecture Characteristics
    - Cross-Cutting Architecture Characteristics

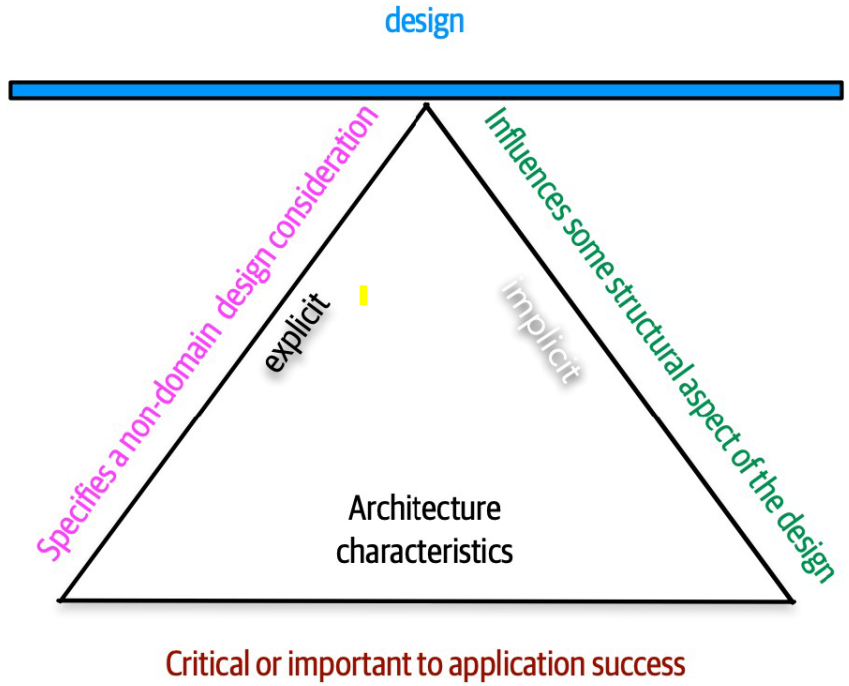


Figure 4-2. The differentiating features of architecture characteristics

Operational Architecture Characteristics

Skarast mikið við **DevOps** áhyggjuefni

Table 4-1. Common operational architecture characteristics

Term	Definition
Availability	How long the system will need to be available (if 24/7, steps need to be in place to allow the system to be up and running quickly in case of any failure).
Continuity	Disaster recovery capability.
Performance	Includes stress testing, peak analysis, analysis of the frequency of functions used, capacity required, and response times. Performance acceptance sometimes requires an exercise of its own, taking months to complete.
Recoverability	Business continuity requirements (e.g., in case of a disaster, how quickly is the system required to be on-line again?). This will affect the backup strategy and requirements for duplicated hardware.
Reliability/safety	Assess if the system needs to be fail-safe, or if it is mission critical in a way that affects lives. If it fails, will it cost the company large sums of money?
Robustness	Ability to handle error and boundary conditions while running if the internet connection goes down or if there's a power outage or hardware failure.
Scalability	Ability for the system to perform and operate as the number of users or requests increases.

Mörg characteristic overlap-a í skilgreiningunni þeirra, t.d. **Availability** og **Reliability**, dæmi um mun þeirra á milli er UDP vs TCP, kerfi getur verið Available þótt það noti UDP en það er ekki endilega Reliable

Structural Architecture Characteristics

Table 4-2. Structural architecture characteristics

Term	Definition
Configurability	Ability for the end users to easily change aspects of the software's configuration (through usable interfaces).
Extensibility	How important it is to plug new pieces of functionality in.
Installability	Ease of system installation on all necessary platforms.
Leverageability/reuse	Ability to leverage common components across multiple products.
Localization	Support for multiple languages on entry/query screens in data fields; on reports, multibyte character requirements and units of measure or currencies.
Maintainability	How easy it is to apply changes and enhance the system?
Portability	Does the system need to run on more than one platform? (For example, does the frontend need to run against Oracle as well as SAP DB?
Supportability	What level of technical support is needed by the application? What level of logging and other facilities are required to debug errors in the system?
Upgradeability	Ability to easily/quickly upgrade from a previous version of this application/solution to a newer version on servers and clients.

Cross-Cutting Architecture Characteristics

Misc flokkur

Table 4-3. Cross-cutting architecture characteristics

Term	Definition
Accessibility	Access to all your users, including those with disabilities like colorblindness or hearing loss.
Archivability	Will the data need to be archived or deleted after a period of time? (For example, customer accounts are to be deleted after three months or marked as obsolete and archived to a secondary database for future access.)
Authentication	Security requirements to ensure users are who they say they are.
Authorization	Security requirements to ensure users can access only certain functions within the application (by use case, subsystem, webpage, business rule, field level, etc.).
Legal	What legislative constraints is the system operating in (data protection, Sarbanes Oxley, GDPR, etc.)? What reservation rights does the company require? Any regulations regarding the way the application is to be built or deployed?
Privacy	Ability to hide transactions from internal company employees (encrypted transactions so even DBAs and network architects cannot see them).
Security	Does the data need to be encrypted in the database? Encrypted for network communication between internal systems? What type of authentication needs to be in place for remote user access?
Supportability	What level of technical support is needed by the application? What level of logging and other facilities are required to debug errors in the system?
Usability/achievability	Level of training required for users to achieve their goals with the application/solution. Usability requirements need to be treated as seriously as any other architectural issue.

Architecture Characteristics Trade-Offs

- Kerfi geta aðeins stutt við nokkur characteristic
  - Characteristic þurfa design effort og structural support
    - Þetta er dýrt/erfitt og tímafrekt
  - Characteristic fara oft á móti hvort öðru
    - Það eru trade-offs við hvaða characteristic er valið
    - T.d. Meira security fer á móti performance
  - Of mörg characteristic leyða til of generic og flókna lausn
    - KISS
    - *„Never shoot for the best architecture, but rather the least worst architecture.“*

Identifying Architectural Characteristics

- Þrjár aðal leiðir til að finna mikilvægustu architectural characteristics
  - *Extracting from domain concerns*
  - *Extracting from Requirements*
  - *Extracting from Implicit domain knowledge*

Extracting from domain concerns meira

- Tala við domain stakeholder-ana
- *„það er nauðsynlegt að allar uppgjörs millifærslur verða kláraðar fyrir enda dagsins“* -> performance, reliability, fault tolerance, scalability etc.

Table 5-1. Translation of domain concerns to architecture characteristics

Domain concern	Architecture characteristics
Mergers and acquisitions	Interoperability, scalability, adaptability, extensibility
Time to market	Agility, testability, deployability
User satisfaction	Performance, availability, fault tolerance, testability, deployability, agility, security
Competitive advantage	Agility, testability, deployability, scalability, availability, fault tolerance
Time and budget	Simplicity, feasibility

Metrics for Architectural Characteristics

- Sum characteristic eru með augljósar mælingar
  - Performance
    - T.d. Mæla response time fyrir requestur
  - Scalability
    - T.d. Mæla response time eftir mismunandi álagi
  - Testability
    - T.d. Code coverage
- Annað er erfiðara að mæla
  - Complexity
    - Ein leið til að mæla t.d. Code complexity væri með *Cyclometic Complexity*
  - Deployability
    - T.d. Hversu lengi hvert deployment tekur
    - Hversu mörg fail vs success build verða
    - Hlutfall villna sem verða við deployment

Scope of Architectural Characteristics

Connascence

- Áður en við tölum um scope þurfum við að tala um connascence

- Connascence er tegund af coupling
- Tveir components eru *connascent* ef breyting í öðrum er líkleg til að leiða til breytingu í hinum
- Tvær tegundir
  - *Static* connascence er hægt að finna við static code analysis
  - *Dynamic* connascence finnst við runtime
    - *Synchronous* connascence
      - Synchronous köll á milli tveggja distributed service
      - Kerfi sem eru synchronously connascent þurfa að hafa sömu operational architecture characteristics **t.d. Ef service *a* kallar á service *b* og *b* er með lélegt scaleability þá er *a* líka með lélegt scaleability**
    - *Asynchronous* connascence
      - Leyfir fire-and-forget í event-driven architecture og leyfir service-um því að vera öðruvísi í operational architecture characteristics

Architecture Quantum

- Almenna skilgreiningin á Quantum
  - *„the minimum amount of any physical entity involved in an interaction“*
- Skilgreiningin á *Architecture Quantum*
  - ***„An independently deployable artifact with high functional cohesion and synchrounous connascence“***
- Architecture Quantum gefur **scope-ið** fyrir architecture characteristics
  - Characteristics skilgreind á *quantum* level-i
  - Characteristic á því við um ákveðið quantum en ekki allt kerfið

Architecture Styles

Architecture Style?

- highest level architecture á kerfinu
- Heildarstrúktur og highest-level yfirlitsmynd
- T.d. 3-Tier / vertical slicing í monolith eða microservices í distributed kerfi

Monolithic vs Distributed Architectures

Architecture styles er hægt að classify-a í tvo hópa, *monolithic* og *distributed*

Monolith

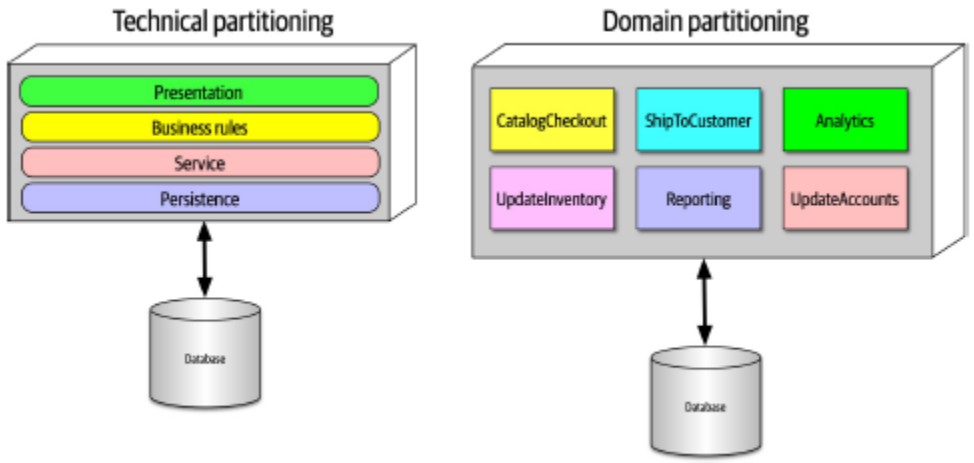


Figure 8-4. Two types of top-level partitioning in architecture

vs

Distributed

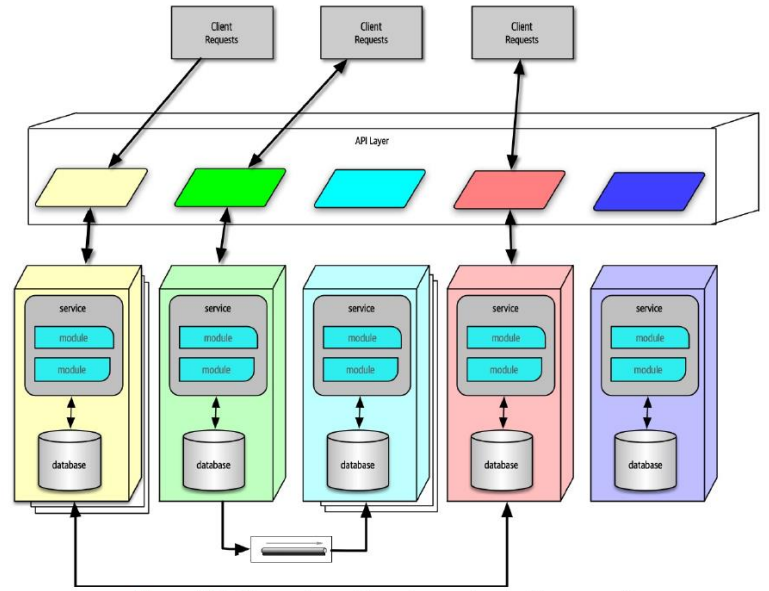


Figure 17-1. The topology of the microservices architecture style

Monolithic vs Distributed

- Hvað er monolith?
  - Í stuttu máli er monolithic kerfi eitthvað kerfi sem er single deployable unit
  - Monolith er eitt *Architectural Quantum*
- Kostir
  - Einfaldleiki
  - Skiljanleiki
  - Getur stundum verið einfalt að scale-a -> duplicate-a kerfið og nota load balancer
  - **Ættir næstum alltaf að byrja með Monolith**
- Gallar
  - Ekki jafn flexible og distributed
  - Stór monolith geta verið lengi að builda og deploy-a (hægja einnig á IDE)
  - Þarft að deploy-a öllu kerfinu við hverja breytingu
  - Læsir technology stack-inum
  - Getur verið erfitt að scale-a
  - Mun meiri coupling
  - Maintenance verður erfiðara þegar kerfið stækkar
  - Teymi/forritarar flækjast fyrir hvort öðrum

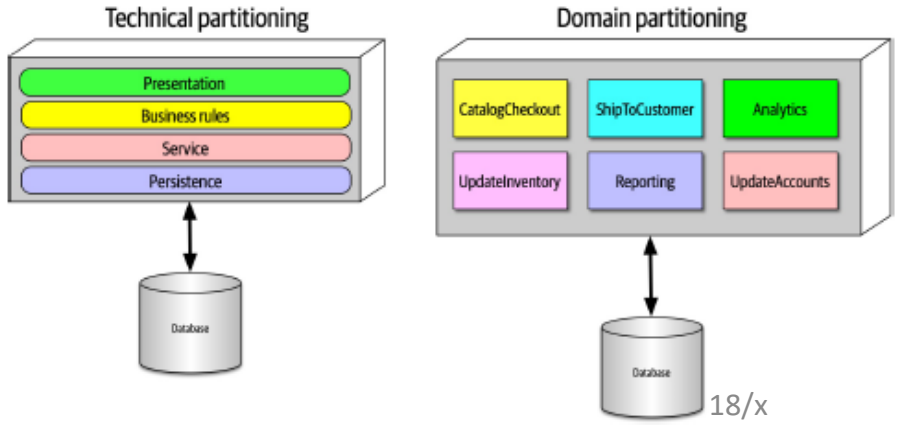


Figure 8-4. Two types of top-level partitioning in architecture

Monolithic vs **Distributed**

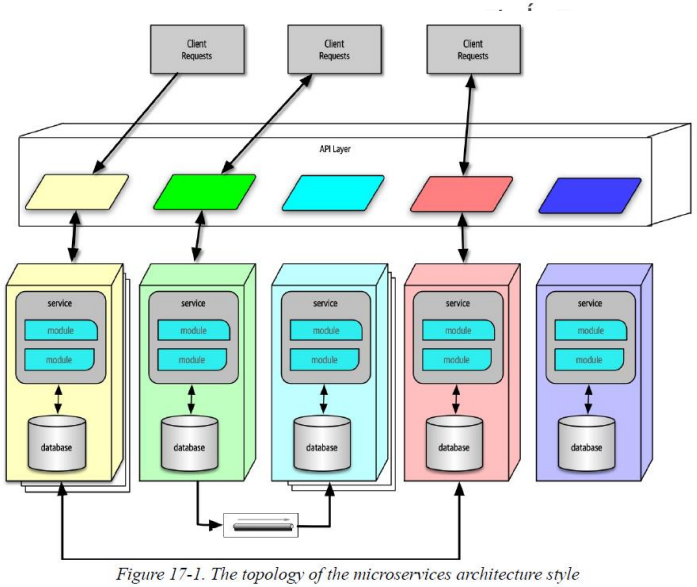


Figure 17-1. The topology of the microservices architecture style

- Kerfi sem er skipt upp í margar einingar
- Keyrandi í sínum eigin umhverfum
- Tala í gegnum networking protocols(http/s, events etc.)
- Vinsælt og mögulegt með framkomu container-a, kubernetes og skýjaþjónusta (t.d. AWS)
- Getur verið mörg Architecture Quantum

Monolithic vs **Distributed**

- Kostir
  - Hvert service getur verið með sín eigin *Architectrual Characteristics*
  - performance, scalability, availability og elasticity
  - Flexibility og adaptability
  - Robustness
  - Deployability -> Hvert service independently deployable
  - Hvert service getur verið með sinn eigin technology stack
  - Auðvelt að skipta út, fjarlægja eða bæta við service
  - Relatively lág coupling á milli componenta
- Gallar
  - Testing erfiðara
  - Flækjustig
  - Erfiðara
  - Dýrara í rekstri
  - Tracability, monitoring, debugging erfiðara
  - Distributed transactions er erfitt
  - *The Fallacies of Distributed Computing*

The Fallacies of Distributed Computing

"A fallacy is something that is believed or assumed to be true but is not."

- Fallacy 1: The Network Is Reliable
- Fallacy 2: Latency is Zero
- Fallacy 3: Bandwith is infinite
- Fallacy 4: The network is secure
- Fallacy 5: The topology never changes
- Fallacy 6: There is only one administrator
- Fallacy 7: Transport cost is zero
- Fallacy 8: The network is homogeneous

Fallacy #1: The Network Is Reliable

Þó netið hefur orðið áreiðanlegra með árunum þá er það ekki þar með sagt það er áreiðanlegt, mikilvægt fyrir distributed kerfi því öll samskipti á milli distributed service-a fara í gegnum netið.

Figure 9-2. The network is not reliable

Fallacy #2: Latency Is Zero

Latency(biðtími) er ekki núll, tíminn sem það tekur distributed service *a* að tala við distributed service *b* mun alltaf vera hærri en ef *a* og *b* væru keyrandi í sama minni og þyrftu ekki að tala saman í gegnum netið, þ.e.a.s $T_{remote} > T_{local}$

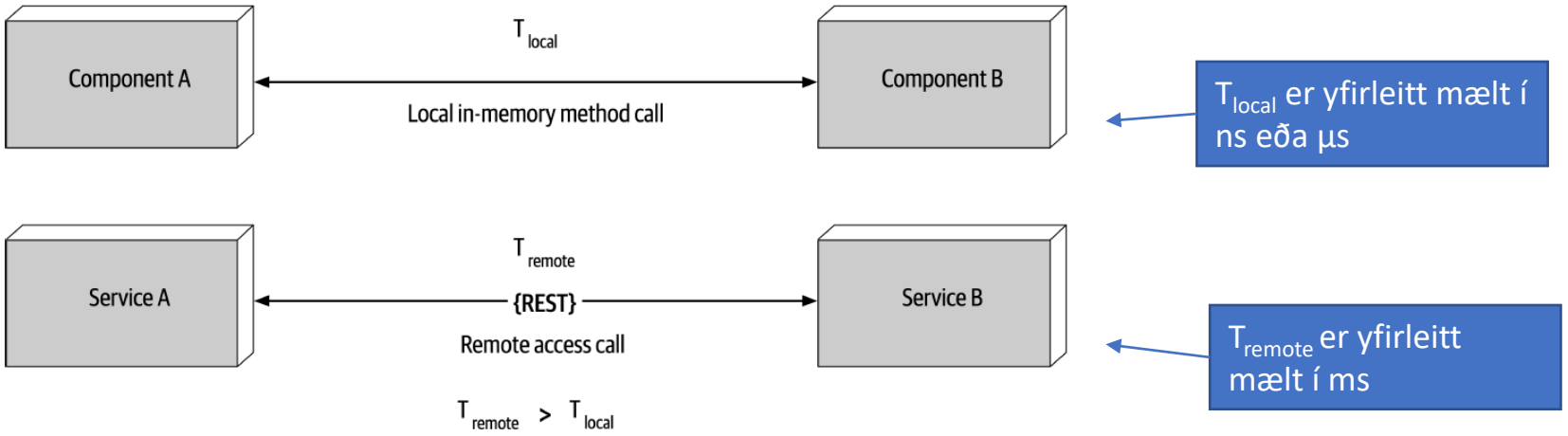


Figure 9-3. Latency is not zero

T_{local} er yfirleitt mælt í ns eða µs

T_{remote} er yfirleitt mælt í ms

Fallacy #3: Bandwidth Is Infinite

Bandwidth(mælt í bits per seconds) hefur mikil áhrif á distributed kerfi því öll samskipti á milli service-a fara í gegnum netið, mikil notkun hefur því bæði áhrif á latency(fallacy 2) og reliability(fallacy 1)

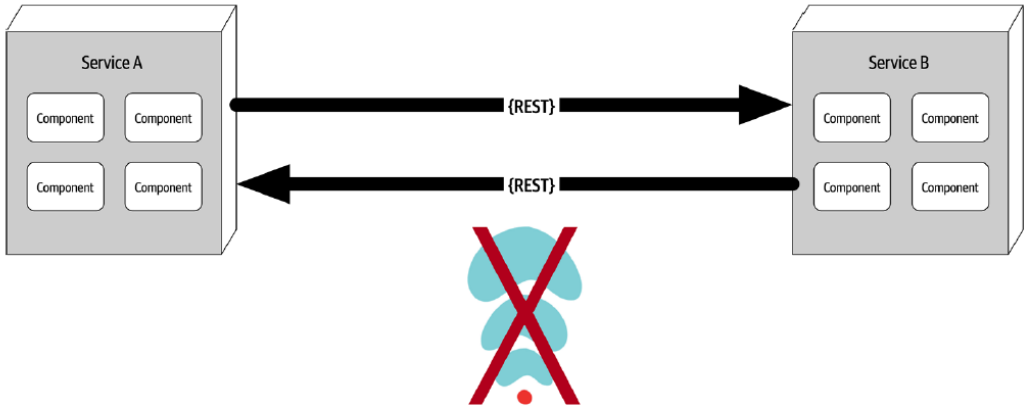


Figure 9-4. Bandwidth is not infinite

Fallacy #4: The Network Is Secure

Surface area-ið til að gera árás verður töluvert stærra í distributed kerfi

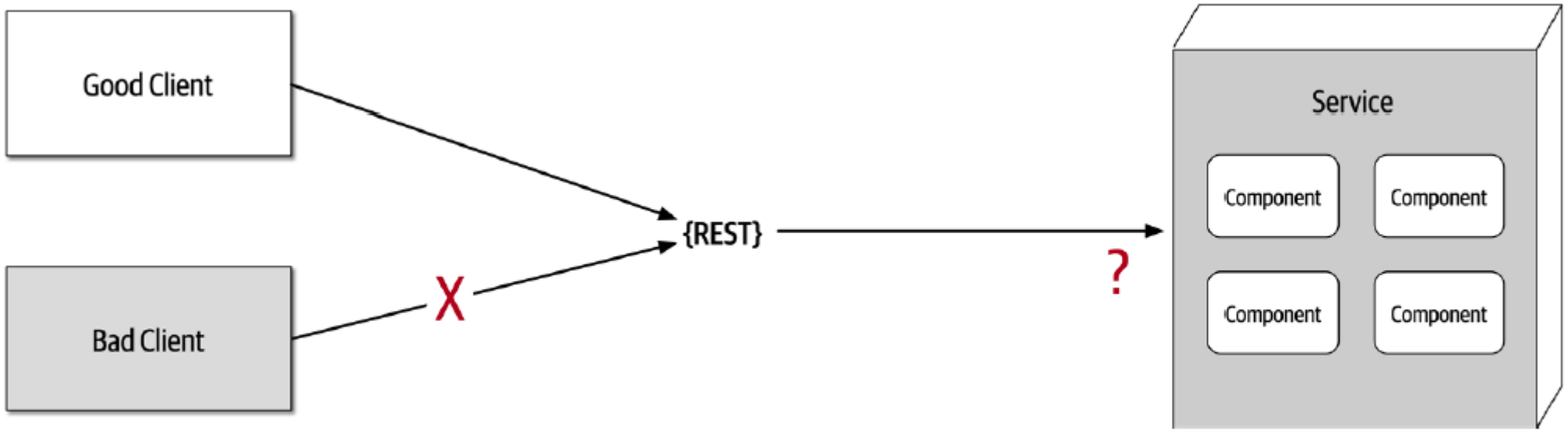


Figure 9-5. The network is not secure

Fallacy #5: The Topology Never Changes

Það sem er átt við með Network Topology er í raun networkið sjálft og allir componentar í því t.d. Routers, hubs, switches, firewalls etc.

T.d. Network update sem trigger-ar timeouts myndi mögulega brjóta kerfið

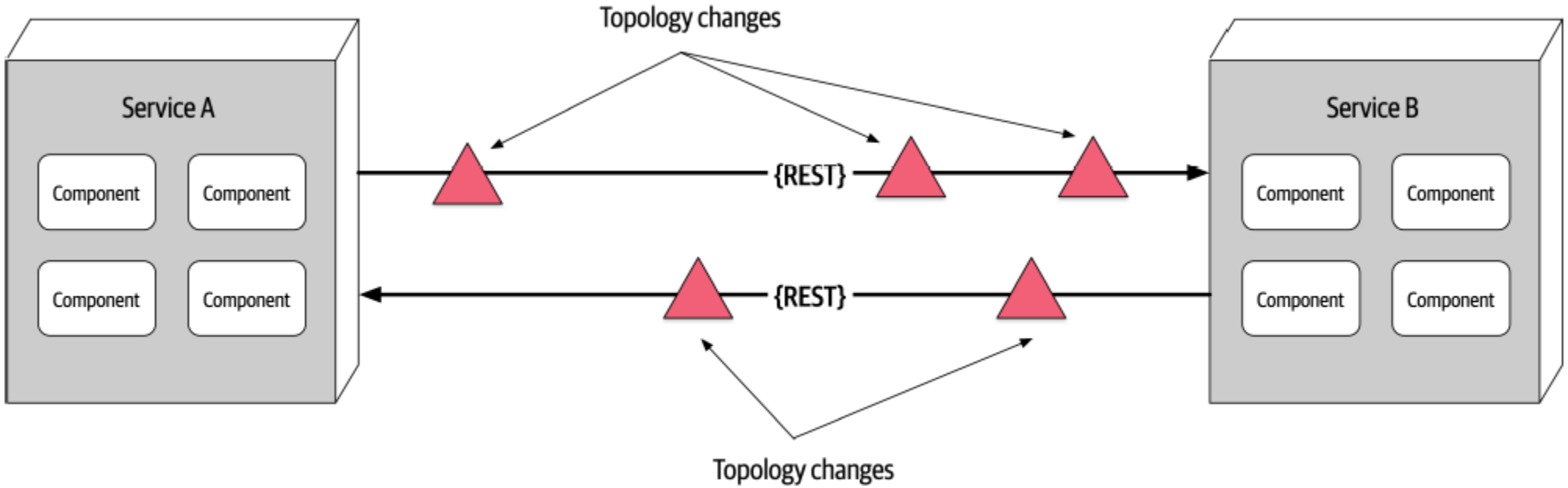


Figure 9-6. The network topology always changes

Fallacy #6: There Is Only One Administrator

Það geta verið margir network administrators í stóru fyrirtæki, þetta fallacy getur því leitt af sér fallacy 5 þar sem architect gæti bara verið í samskiptum við einn admin á meðan er eitthver annar admin að krukka í networkinu og gera breytingar

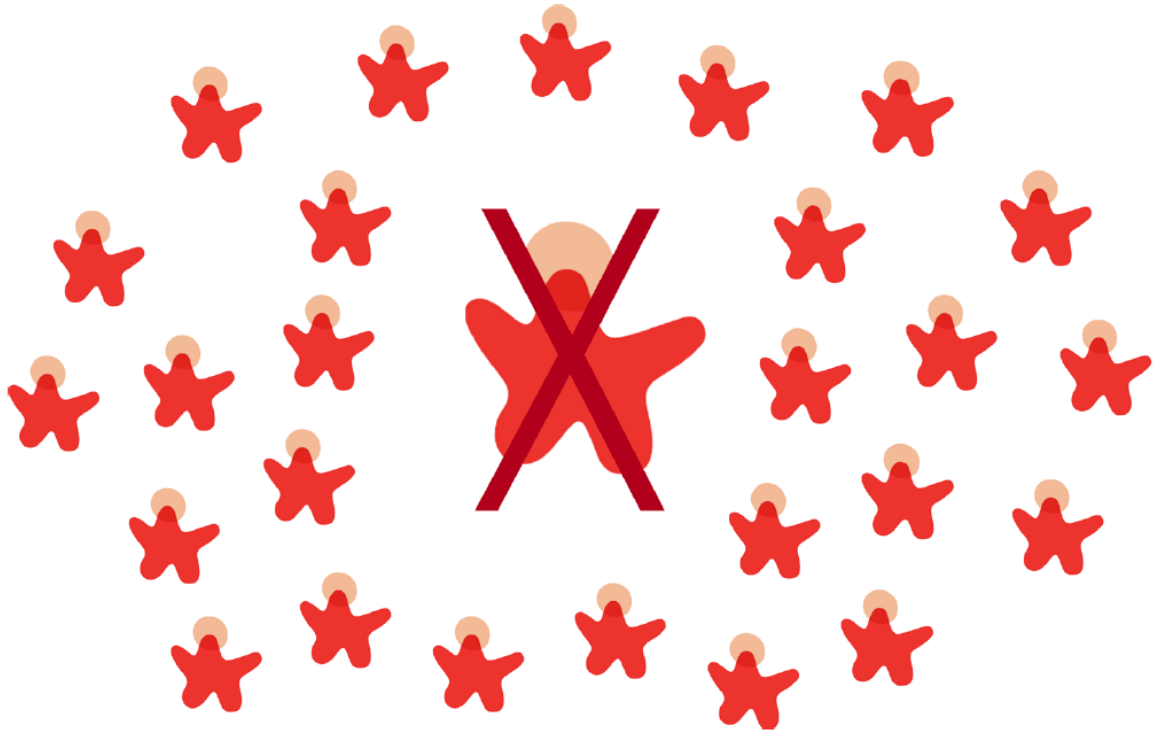


Figure 9-7. There are many network administrators, not just one

Fallacy #7: Transport Cost Is Zero

Distributed kerfi kosta yfirleitt meira en monoliths, þarft fleiri resources, gateways etc.

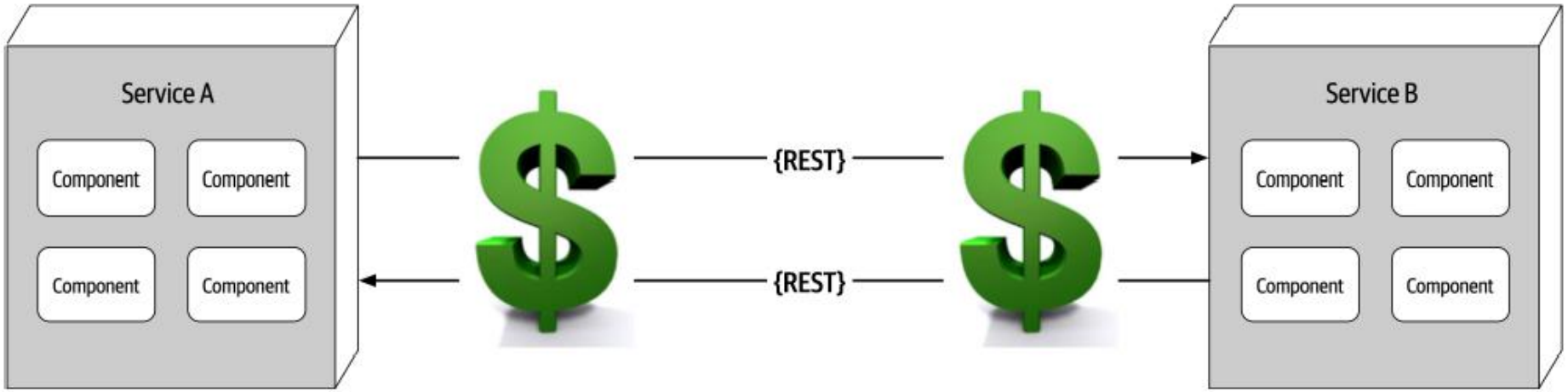


Figure 9-8. Remote access costs money

Fallacy #8: The Network Is Homogeneous

Networkið getur verið samansett af mörgum hardware söluaðilum,

„does Juniper hardware seamlessly integrate with Cisco hardware? “

Þetta vandamál er minna vandamál í dag en áður

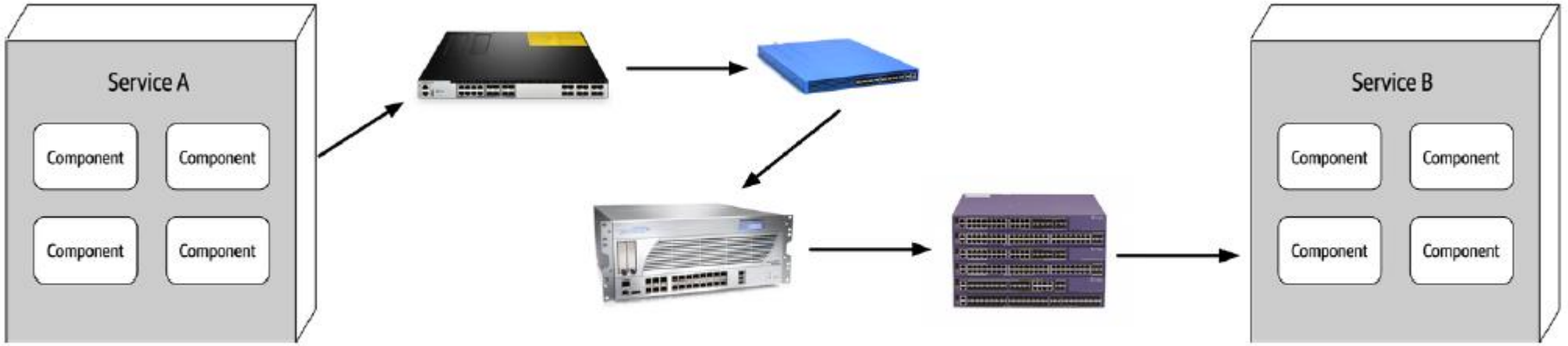


Figure 9-9. The network is not homogeneous

Back to styles -> Monolithic vs Distributed

Monolithic styles

- N-tier architecture
- Vertical slicing
- Onion architecture
- Pipeline architecture

Búin að skoða þetta

Distributed styles

- Service-based architecture
- ~~Service-oriented architecture~~
- **Event-driven architecture**
- **Microservices architecture**

Eigum eftir að skoða þetta og **munum að mestu leyti skoða microservices og event driven architecture** en gott að hafa í huga að það eru aðrir stílar til.

Service-Based Style

- Service-Based Architecture
  - Independently deployable user interface
  - independently deployable domain services
  - Sameiginlegur gagnagrunnur
  - Domain service-in tala ekki við hvort annað
- Transactions
  - Service nægilega umfangsmikil til að geta notað ACID transactions
- Domain partitioned
  - Hvert service sér um eitt domain
- Service-in eru fá
  - Á bilinu 4-12
  - Ekki hægt að hafa mikið fleiri því þau tala öll við sama gagnagrunn
  - Ef service-in eru mun fleiri en 12 þá hentar microservice architecture örugglega betur
- Eitt Architecture Quantum
  - Sameiginlegur gagnagrunnur -> Eitt Qunatum

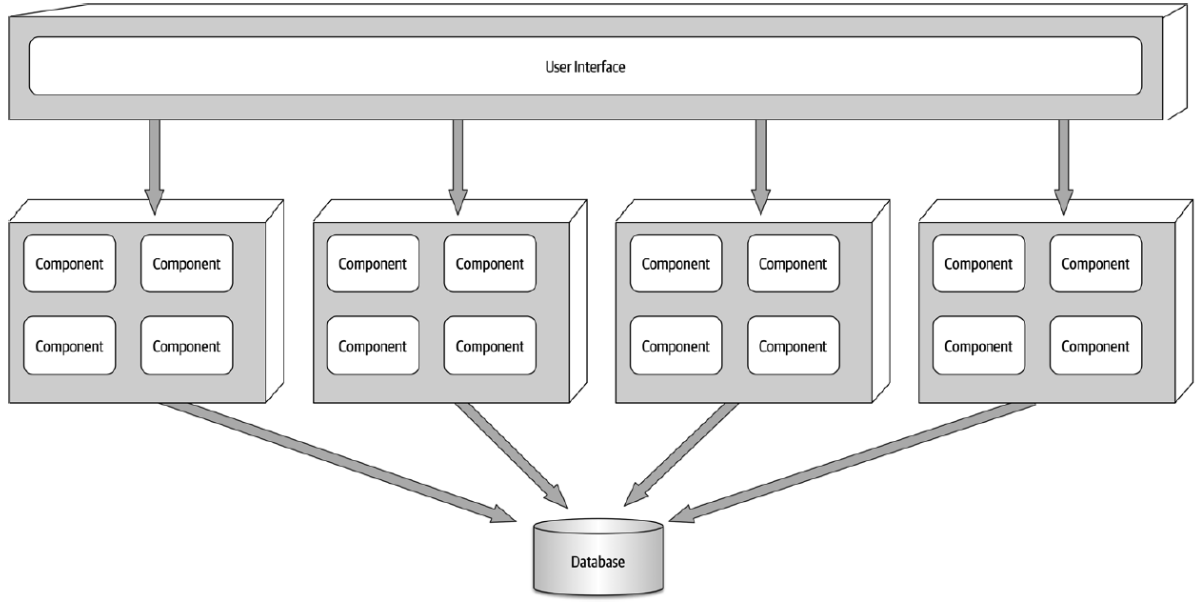
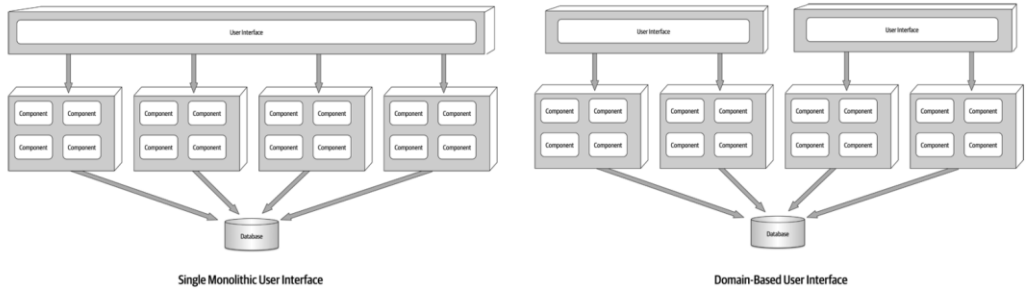


Figure 13-1. Basic topology of the service-based architecture style

Service-Based Architecture variations



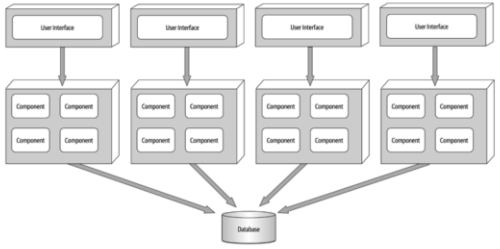
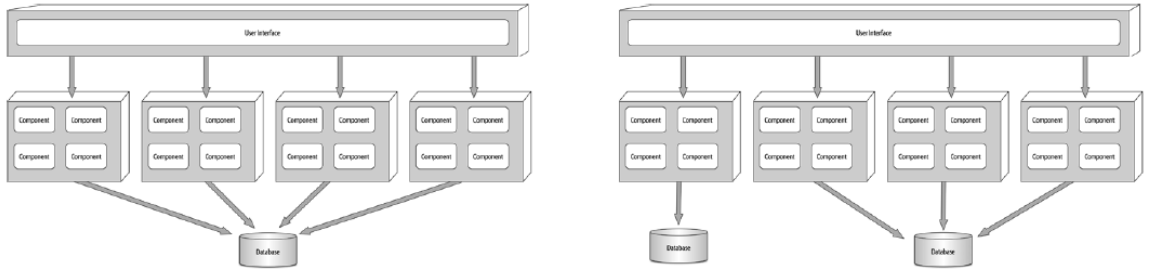


Figure 13-2. User interface variants



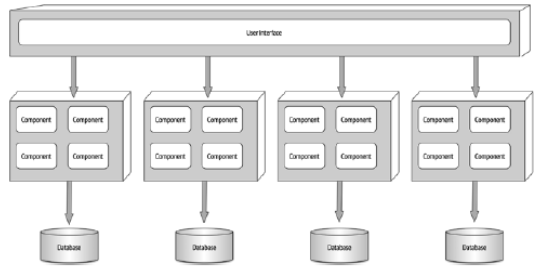


Figure 13-3. Database variants

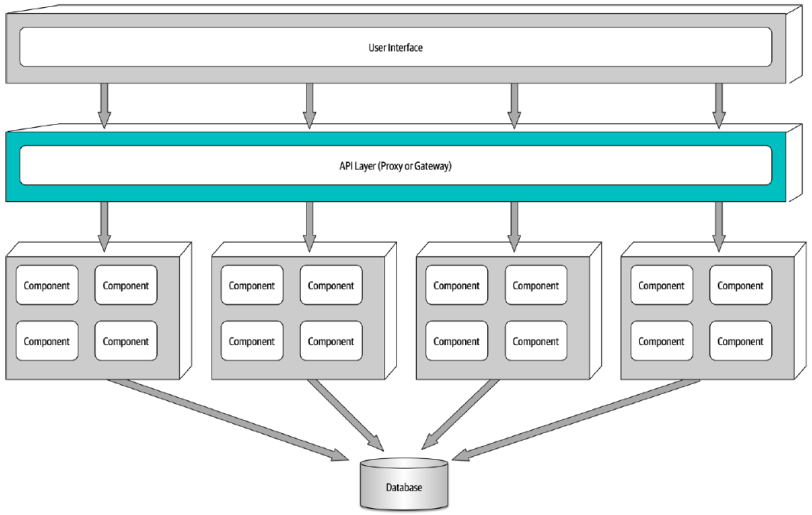


Figure 13-4. Adding an API layer between the user interface and domain services

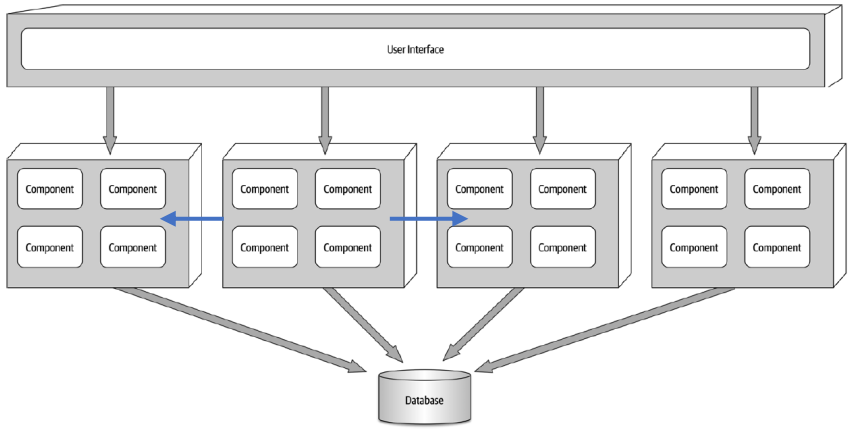


Figure 13-1. Basic topology of the service-based architecture style

Database Partitioning for Service-Based Architecture

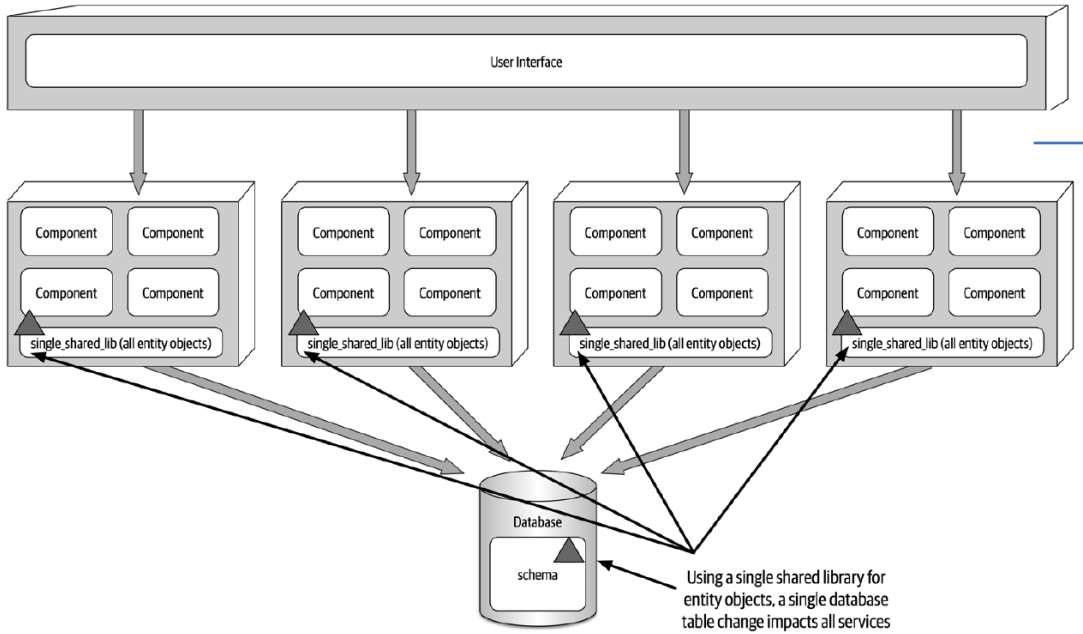


Figure 13-6. Using a single shared library for database entity objects

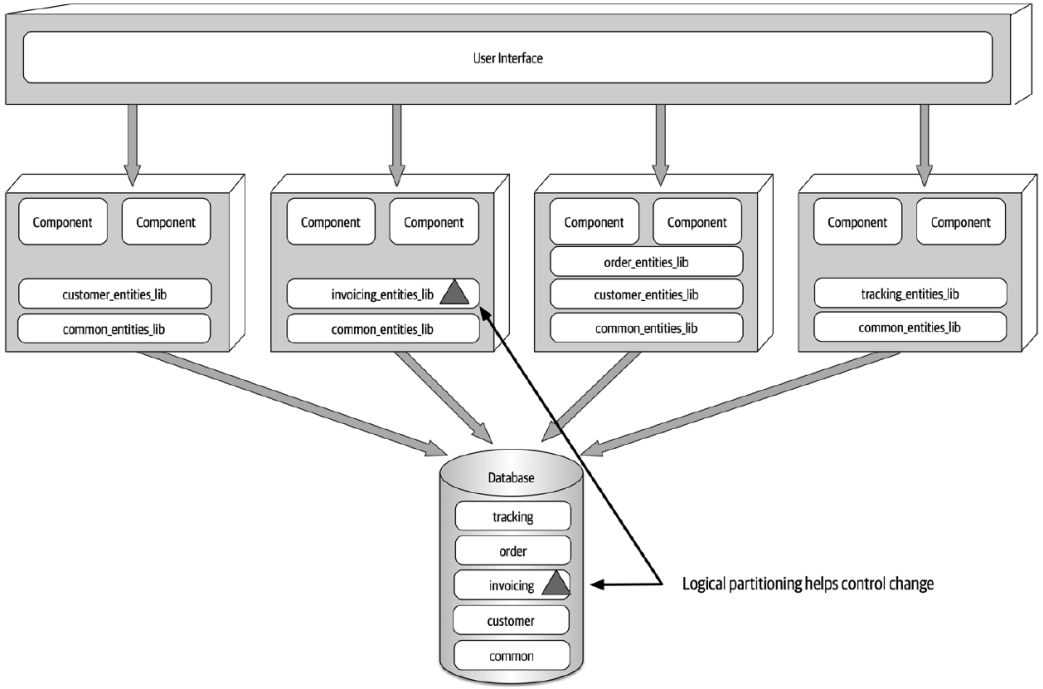


Figure 13-7. Using multiple shared libraries for database entity objects