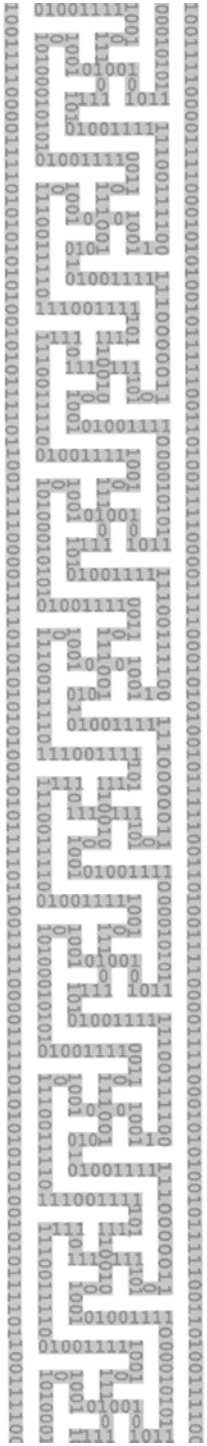


АРХИТЕКТУРУУДЫН ХАРЬЦУУЛАЛТ

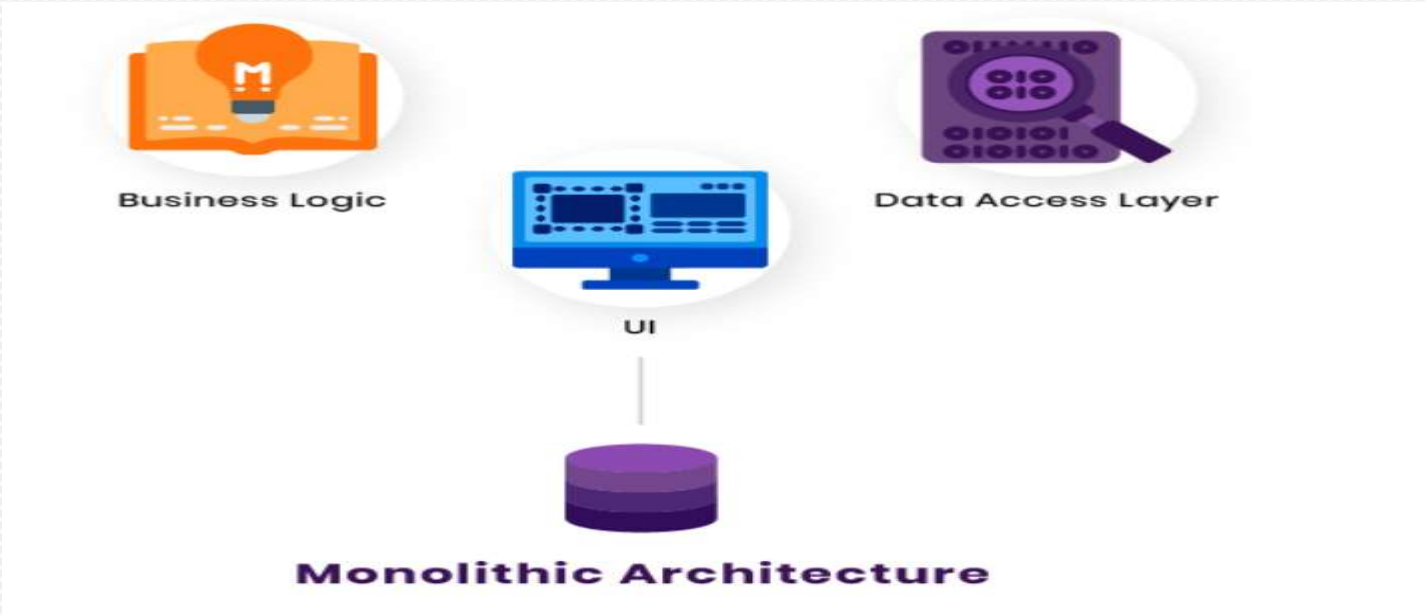
Доктор Намсрайдоржийн МӨНХЦЭЦЭГ

Архитектуруудын харьцуулалт

- Ямар нэгэн шинэ програм хангамжийн бүтээгдэхүүнийг амжилттай хийхэд системийн архитектурыг зөв сонгох нь маш чухал
- Хэрвээ monolithic, service-oriented, microservice, болон serverless architecture-уудын ялгааг сайн мэдэж байвал зөв сонголтыг хийхэд тус нэмэр болно.

Monolithic architecture

- Монолит бол асар том ганц чулууг хэлдэг эртний үг юм.
- Хэдийгээр энэ нэр томъёо өнөөдөр өргөн хэрэглэгддэг боловч зураг нь бүх салбарт ижил хэвээр байна.
- Програм хангамжийн инженерчлэлд цул буюу нэг бүхэл загвар гэдэг нь нэг хуваагдашгүй нэгжийг хэлдэг.
- Цул програм хангамжийн тухай ойлголт нь програмын өөр өөр бүрэлдэхүүн хэсгүүдийг нэг платформ дээр нэг програм болгон нэгтгэхэд оршино.
- Ихэвчлэн цул апп нь мэдээллийн сан, үйлчлүүлэгч талын хэрэглэгчийн интерфэйс, сервер талын програмаас бүрдэнэ.



Монолит архитектур

- Монолит архитектур нь цөөн тооны хөгжүүлэгчтэй, стартап програм хангамжийн компаниудад их зохимжтой.
- Монолит програм хангамжийн архитектур нь бүрэлдэхүүн хэсгүүд нь хоорондоо уялдаатай, бие биенээсээ хамааралтай байдаг бөгөөд энэ нь програм хангамжийг бие дааж ажиллахад тусалдаг.
- Энэхүү архитектур нь програм бүтээх уламжлалт шийдэл боловч зарим хөгжүүлэгчид үүнийг хуучирсан гэж үздэг.
- Гэсэн хэдий ч монолит архитектур нь зарим төрлийн програм хангамжийн төгс шийдэл болдог байна.

Монолитийн давуу тал

- Хөгжүүлэлт нь хялбар
  - Хөгжүүлэлтийг хялбаршуулах олон хэрэгсэл байдаг.
  - Бүх үйлдлийг нэг удирдлагаар гүйцэтгэдэг бөгөөд энэ нь хаана, юу орж байгааг удирдахад хялбар
  - Монолит нь хөгжүүлэгчид өөрчлөлт, шинэчлэлтийг тус тусдаа хийх хэрэггүй нэг дор хийдэг учраас маш их цаг хэмнэдэг.
- Гүйцэтгэл хурдан
  - Микросервис архитектур бүхий апликэйшинг бодвол бага хэмжээний апп учраас гүйцэтгэл хурдан байдаг. Жнь: Микросервис архитектурын хувьд дор хаяж 40 сервис, тус бүрдээ хэрэглэгчийн интерфэйстэйй байдаг.

Монолитийн дутагдалтай тал

- Хугацааны уртад код ихсэнэ.
  - Хэрэв таны аппликейшнд шинэ технологи нэмэх шаардлагатай бол хөгжүүлэгчид нэвтрүүлэхэд саад бэрхшээл тулгарч магадгүй юм. Шинэ технологи нэмнэ гэдэг нь өртөг өндөртэй, цаг хугацаа шаардсан програмыг бүхэлд нь дахин бичих гэсэн үг юм.
- Шинэ технологийг нэвтрүүлэхэд хэцүү байдаг
  - Цаг хугацаа өнгөрөхөд ихэнх бүтээгдэхүүн хөгжиж, цар хүрээ нь өргөжиж, бүтэц нь бүдгэрдэг. Кодын хэмжээ их болж ялангуяа шинэ хөгжүүлэгчдийн хувьд ойлгох, өөрчлөхөд хэцүү болдог. Үүнээс улбаалан алдаа, хамаарлыг олоход хэцүү болдог. Код өсөн нэмэгдэж байгаатай холбогдуулан кодын зохиомжийн чанар буурч, хөгжүүлэхэд хэт ачаалалтай болдог.
- Agile байдал дутмаг
  - Ямар нэг өөрчлөлтийг хийхэд системийг бүхэлд нь харах, бодолцох ёстой учраас бусад хөгжүүлэгчид ажиллахад төвөг болдог

Гэсэн хэдийн монолит архитектур бүхий програм хангамж хэрэглээнээс гараагүй бөгөөд өөрийн тохирсон систе болон хэрвээ хөгжүүлэгчид микросервис архитектурын шийдлийг сайн хийж чадахгүй байгаа тохиолдолд сайн шийдэл болдог.

A SERVICE-ORIENTED ARCHITECTURE (SOA)

- Үйлчилгээнд чиглэсэн архитектур (SOA) нь шаардлагатай функцийг гүйцэтгэх үүргээр нь салгасан, програм хангамжийн агентуудыг хооронд нь чөлөөтэйгөөр холбосон програм хангамжийн архитектурын хэв маяг юм.
- SOA нь үйлчилгээ үзүүлэгч ба үйлчилгээний хэрэглэгч гэсэн хоёр үндсэн үүрэгтэй. Эдгээр хоёр үүргийг програм хангамжийн төлөөлөгч гүйцэтгэж болно.
- Аппликейшн нь түүний модулиудыг хооронд нь нэгдмэл байдлаар нэгтгэж, дахин ашиглахад хялбар байдлаар зохион байгуулдаг.

SOA-гийн давуу тал

- **Үйлчилгээг дахин ашиглах боломжтой байдал**

Үйлчилгээнд чиглэсэн хэрэглээний функциональ бүрэлдэхүүн хэсгүүдийн бие даасан, чөлөөтэй хосолсон шинж чанараас шалтгаалан эдгээр бүрэлдэхүүн хэсгүүдийг бусад үйлчилгээнд нөлөөлөхгүйгээр олон програмд дахин ашиглах боломжтой.

- **Сайн арчлах боломжтой**

Програм хангамжийн үйлчилгээ бүр бие даасан нэгж тул бусад үйлчилгээг гэмтээхгүйгээр шинэчлэх, засварлахад хялбар байдаг. Жишээлбэл, томоохон аж ахуйн нэгжийн програмуудыг үйлчилгээнд хуваахад илүү хялбар удирдах боломжтой.

- **Найдвартай байдал өндөр**

Үйлчилгээ нь монолитийг бодвол кодыг хэмжээ нь олон хэсэг салсан байдаг учраас илүү дибаг хийх, шалгахад илүү хялбар байдаг.

- **Зэрэгцээгээр хөгжүүлэх боломжтой**

Үйлчилгээнд чиглэсэн архитектур нь давхаргаас бүрддэг тул хөгжлийн явцад параллелизмыг дэмждэг. Бие даасан үйлчилгээг зэрэг хөгжүүлж, нэгэн зэрэг дуусгах боломжтой.

SOA-гийн дутагдалтай тал

Цогц менежмент

Үйлчилгээнд чиглэсэн архитектурын гол сул тал бол түүний нарийн төвөгтэй байдал юм. Үйлчилгээ бүр нь мессежийг цаг тухайд нь хүргэх ёстой. Эдгээр зурвасын тоо нэг удаад сая гаруй байж болох бөгөөд энэ нь бүх үйлчилгээг удирдахад маш том сорилт болдог.

Хөрөнгө оруулалтын зардал өндөр

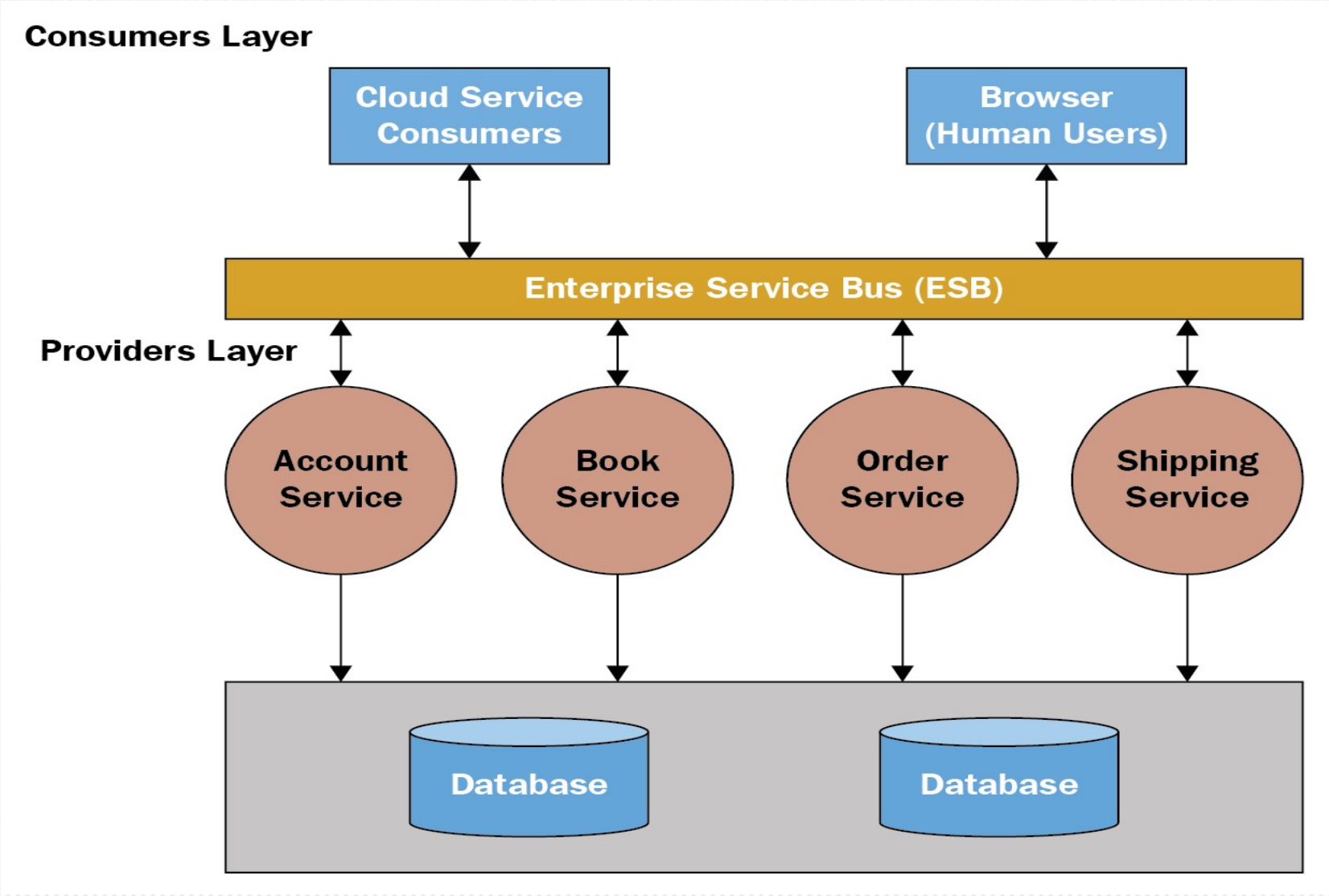
SOA -ийн хөгжил нь хүний нөөц, технологи, хөгжлийн томоохон хөрөнгө оруулалтыг шаарддаг.

Нэмэлт хэт ачаалал

SOA -д нэг үйлчилгээ нөгөө үйлчилгээтэй харьцахаас өмнө бүх оролтыг баталгаажуулдаг. Олон үйлчилгээг ашиглах үед энэ нь хариу өгөх хугацааг нэмэгдүүлж, ерөнхий гүйцэтгэлийг бууруулдаг.

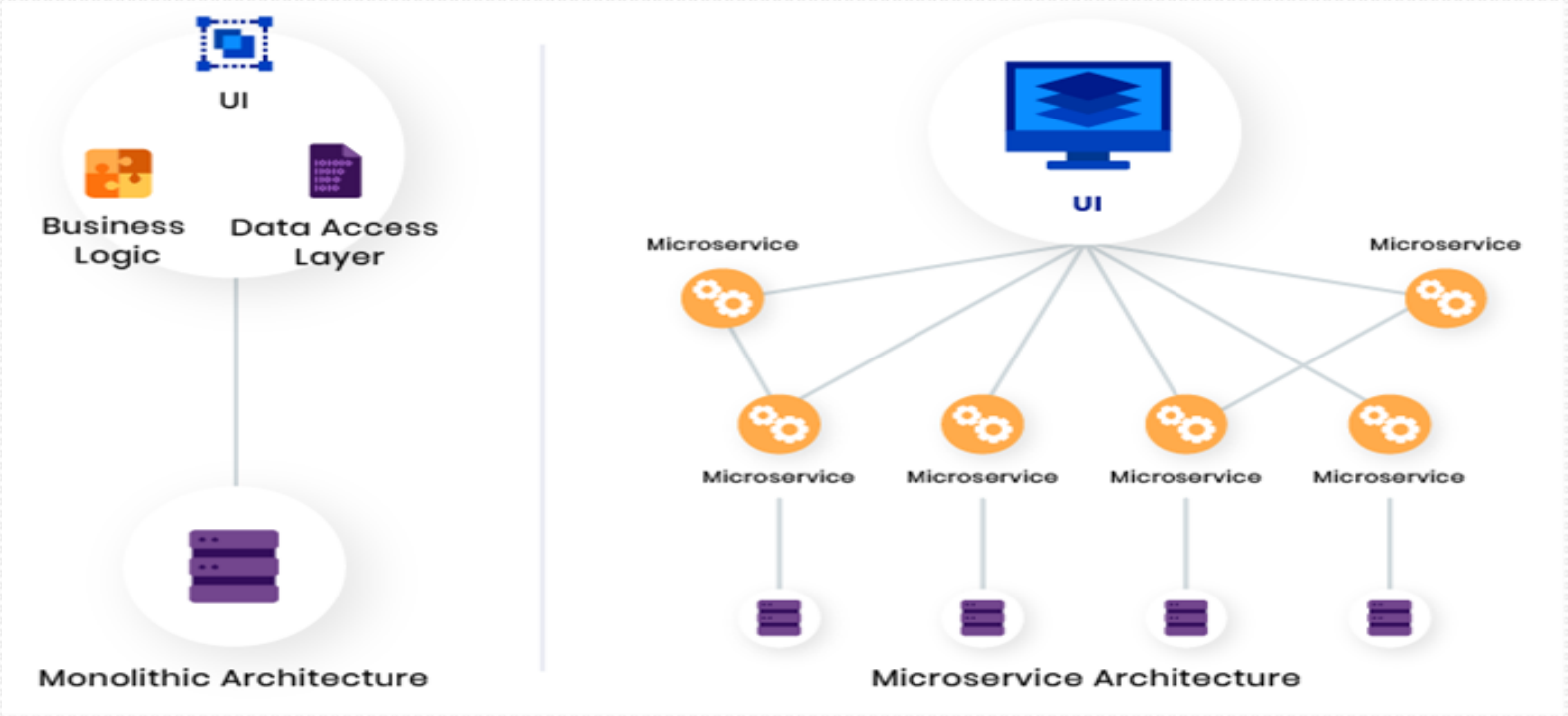
SOA архитектурыг банкны систем гэх мэт аж ахуйн нэгжийн нарийн төвөгтэй системд хамгийн тохиромжтой байдаг. Банкны системийг бичил үйлчилгээнд хуваахад маш хэцүү байдаг. Гэхдээ монолит архитектур нь банкны системийн хувьд сайн биш, учир нь нэг хэсэг нь бүхэл бүтэн програмыг гэмтээж болзошгүй юм. Хамгийн сайн шийдэл бол SOA аргыг ашиглах, нарийн төвөгтэй програмуудад салган бие даасан үйлчилгээ болгон зохион байгуулах явдал юм.

SOA


Consumers Layer
Cloud Service Consumers
Browser (Human Users)
Enterprise Service Bus (ESB)
Providers Layer
Account Service
Book Service
Order Service
Shipping Service
Database
Database

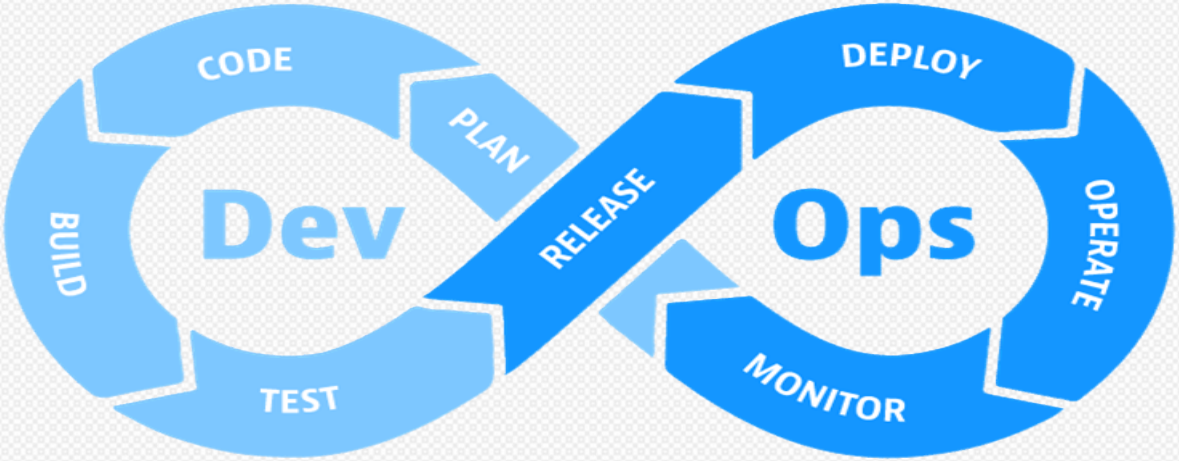
MICROSERVICE ARCHITECTURE

- Microservice нь програмыг бүрдүүлдэг хэд хэдэн бие даасан компонентуудаас бүрдсэн үйлчилгээнд суурилсан програм хангамжийн архитектурын нэг төрөл юм.
- Нэг хуваагдашгүй нэгж болгон бүтээсэн монолит архитектур бүхий программаас ялгаатай нь олон жижиг бие даан ажиллах компонентууд нь өөрийн API-тай байна.



Микросервис архитектур

- Микросервис архитектур нь бизнесийн логикт анхаарал хандуулдаг бол монолит нь технологийн давхарга болох интерфейс, программын удирдлага, өгөгдлийн сан зэрэгт анхаарал хандуулдаг.
- Сүүлийн жилүүдэд олон аж ахуйн нэгжүүд уян хатан болж, DevOps руу шилжихийн хэрээр microservice архитектурын шийдлийг их сонгож байна.



- **DevOps** is a set of practices that combines software development (*Dev*) and IT operations (*Ops*). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.[1] DevOps is complementary with Agile software development; several DevOps aspects came from the Agile methodology.

- Микросервис архитектур нь систем дэх нарийн төвөгтэй байдлыг хялбарчлах өвөрмөц үнэ цэнийг нэмж өгдөг учраас чухал ач холбогдолтой юм.
- Систем эсвэл програмаа олон жижиг хэсгүүдэд задалснаар давхардлыг бууруулах, уялдаа холбоог нэмэгдүүлэх, хэсгүүдийн хоорондох холбоо харуулснаар системийн ерөнхий хэсгүүдийг ойлгоход хялбар, өргөтгөх боломжтой, өөрчлөхөд хялбар болно.
- Монолит хандлагаас татгалзан Микросервис архитектурыг сонгосон олон компаниуд бий. Хамгийн алдартай нь Netflix, Amazon, Twitter, eBay, PayPal юм.

Микросервис давуу тал

- Хөгжүүлэх, турших, байршуулахад хялбар
- Микро сервисийн бусад архитектураас ялгагдах хамгийн том давуу тал нь биеэ даасан жижиг үйлчилгээг бие даан бүтээх, турших, байршуулах боломжтой юм.
- Программыг deploy хийх хурд маш бага байдаг нь хөгжүүлэгчид болон realtime ажиллаж байгаа системд өөрчлөлт хийх, туршилт явуулахад маш хялбар боломжийг олгодог. Хөгжүүлэгчид програмыг бүхэлд нь биш харин програм хангамжийн хэсгийг deploy хийснээр deploy хийх эрсдэл буурдаг.
- Agile буюу уян хатан байдал нэмэгдсэн
- Микро сервис архитектураар хэд хэдэн баг өөрсдийн үйлчилгээг бий бүтээхдээ бие даан, хурдан ажиллах боломжтой. Микро сервис архитектураар хэсгүүдийг салгаснаар програмын хэсэг бүрийг бие даан бүтээх боломжтой.
- Хэвтээ чиглэлд scale хийх чадвар
- Хэвтээ чиглэлд (ижил санд илүү олон үйлчилгээ бий болгох) нь хязгаарлагдахгүй бөгөөд бичил үйлчилгээгээр динамикаар

Микросервис дутагдалтай тал

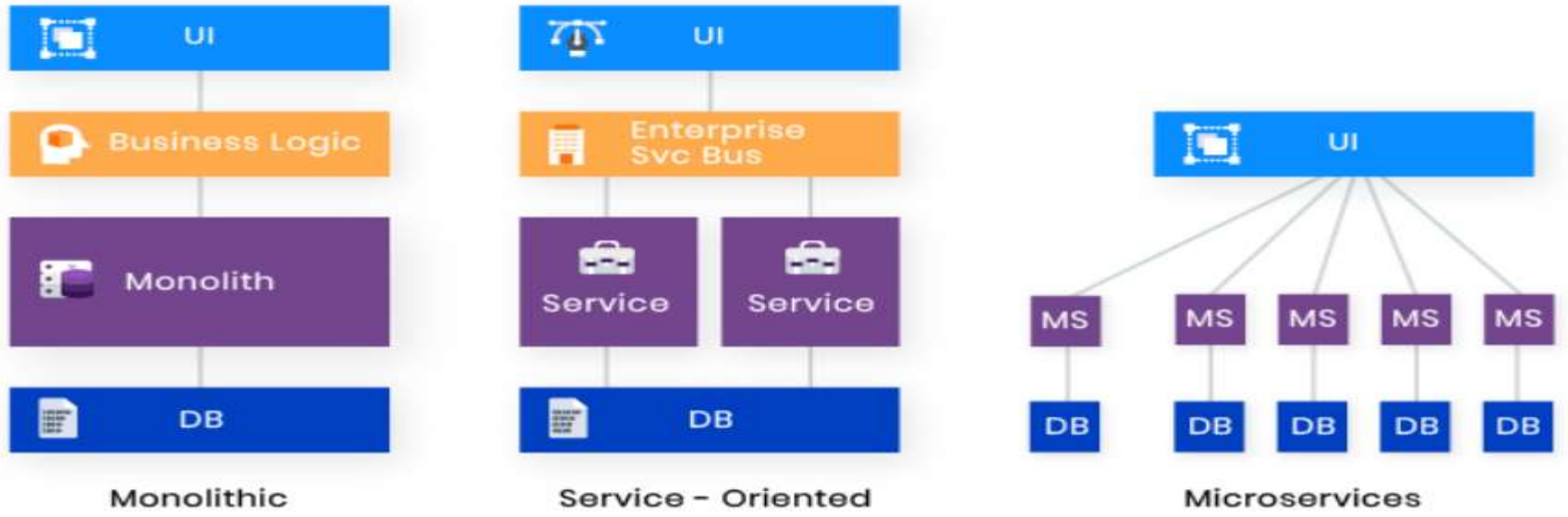
- **Нарийн төвөгтэй байдал**
- Микросервисийн хамгийн том сул тал бол түүний нарийн төвөгтэй байдал юм. Аппликешныг бие даасан Микросервис болгон хуваах нь илүү олон зүйлүүдийг удирдах шаардлагатай болдог. Энэ төрлийн архитектур нь нарийн төлөвлөлт, асар их хүчин чармайлт, багийн нөөц, ур чадвар шаарддаг. Нарийн төвөгтэй байдлын шалтгаан нь дараах зүйлүүдээс шалтгаалдаг. Үүнд:
- **Үйлчилгээ бүрийг туршиж, хянаж байх ёстой тул автоматжуулалтын эрэлт нэмэгдсэн.**
- **Ашиглагддаг хэрэгсэл болон түүл нь үйлчилгээ бүрт тохируулахад хэцүү.**
- **Мэдээллийн найдвартай байдал**, транзакшин менежмент нь үйлчилгээ тус бүр мэдээллийн сантай болсноор илүү хэцүү болдог.
- **Аюулгүй байдлын асуудал**
- Микросервис аппликейшн дээр API -ээр дамжуулан гаднаас харилцах функц бүр халдлагад өртөх магадлалыг нэмэгдүүлдэг.
- **Програмчлалын хэлний ялгаа**

Микросервис

- Микросервис нь сайн боловч бүх төрлийн аппликейшнд тохиромжгүй байдаг.
- Энэ загвар нь нарийн төвөгтэй системүүдийн хөгжүүлэлтэнд маш сайн ажилладаг.
- Олон туршлагатай хөгжүүлэгчдийн багтай мөн том хэмжээний олон үйлчилгээ бүхий системийн хувьд микросервис архитектурыг сонгоход тохиромжтой.
- Аппликешн нь том хэмжээтэй бөгөөд уян хатан, өргөтгөх боломжтой байх үед микросервис ашигтай байдаг.

Харьцуулалт

- Монолит програмууд нь бие биенээсээ хамааралтай, хуваагддаггүй нэгжүүдээс бүрдэх бөгөөд хөгжлийн хурд маш бага байдаг.
- SOA нь жижиг болон дунд зэрэг хоорондоо холбогдсон үйлчилгээнд хуваагддаг бөгөөд хөгжүүлэлтийн хурд удаавтар гэж үзэж болно.
- Микросервисийн хувьд маш жижиг, бие биентэйгээ чөлөөтэй холбогдсон үйлчилгээ бөгөөд тасралтгүй хөгжиж байдаг.



SERVERLESS ARCHITECTURE

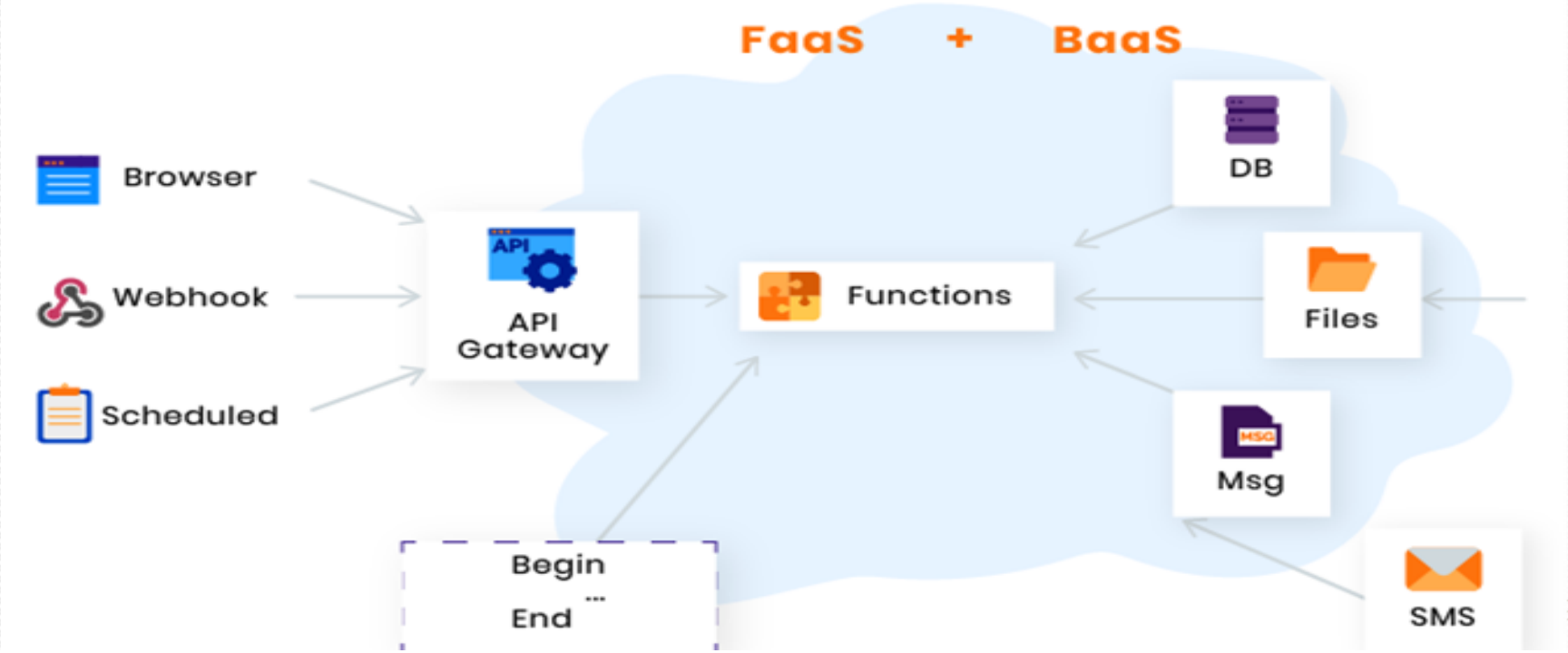
- Serverless computing гэдэг бол сервергүй биш харин хөгжүүлэгчид серверийн дэд бүтцэд санаа зовохгүйгээр гуравдагч үүлэн тооцоолол үзүүлэгч компаниудаас дэд бүтцийг асуудлыг шийдүүлсэн шийдлийг хэлнэ.
- Гол гурван том компани бол

  - Amazon Web Services (AWS)
  - Google Cloud Function
  -Microsoft Azure serverless

,

Serverless архитектурт 2 үндсэн ойлголт байдаг. Үүнд:

- **FaaS (Function as a Service)** – a cloud computing model which allows developers to upload pieces of functionality to the cloud and let these pieces be executed independently
- **BaaS (Backend as a Service)** – a cloud computing model which allows developers to outsource backend aspects (database management, cloud storage, hosting, user authentication, etc.) and write and maintain only the frontend part



SERVERLESS ДАВУУ ТАЛ

- **Байршуулахад буюу deploy хийхэд хялбар**

Сервергүй програмуудад хөгжүүлэгчид дэд бүтцийн талаар санаа зовох шаардлагагүй болно. Энэ нь тэдэнд код дээр анхаарлаа төвлөрүүлэх боломжийг олгодог. Сервергүй архитектур нь програмыг маш хурдан ажилд оруулах боломжийг олгодог, учир нь байршуулалт буюу deploy хэдхэн цаг, хэдэн өдөр болдог (уламжлалт арга барилтай харьцуулахад).

- **Бага зардал**

Сервергүй ажиллах нь зардлыг бууруулдаг. Мэдээллийн сан, зарим логик, серверүүдтэй ажиллах шаардлагагүй тул илүү өндөр чанартай код үүсгэхээс гадна зардлаа бууруулах боломжтой болно. Сервергүй загварыг ашиглахдаа зөвхөн ашигладаг CPU болон санах ойн ашиглалтанд төлбөрийг төлдөг.

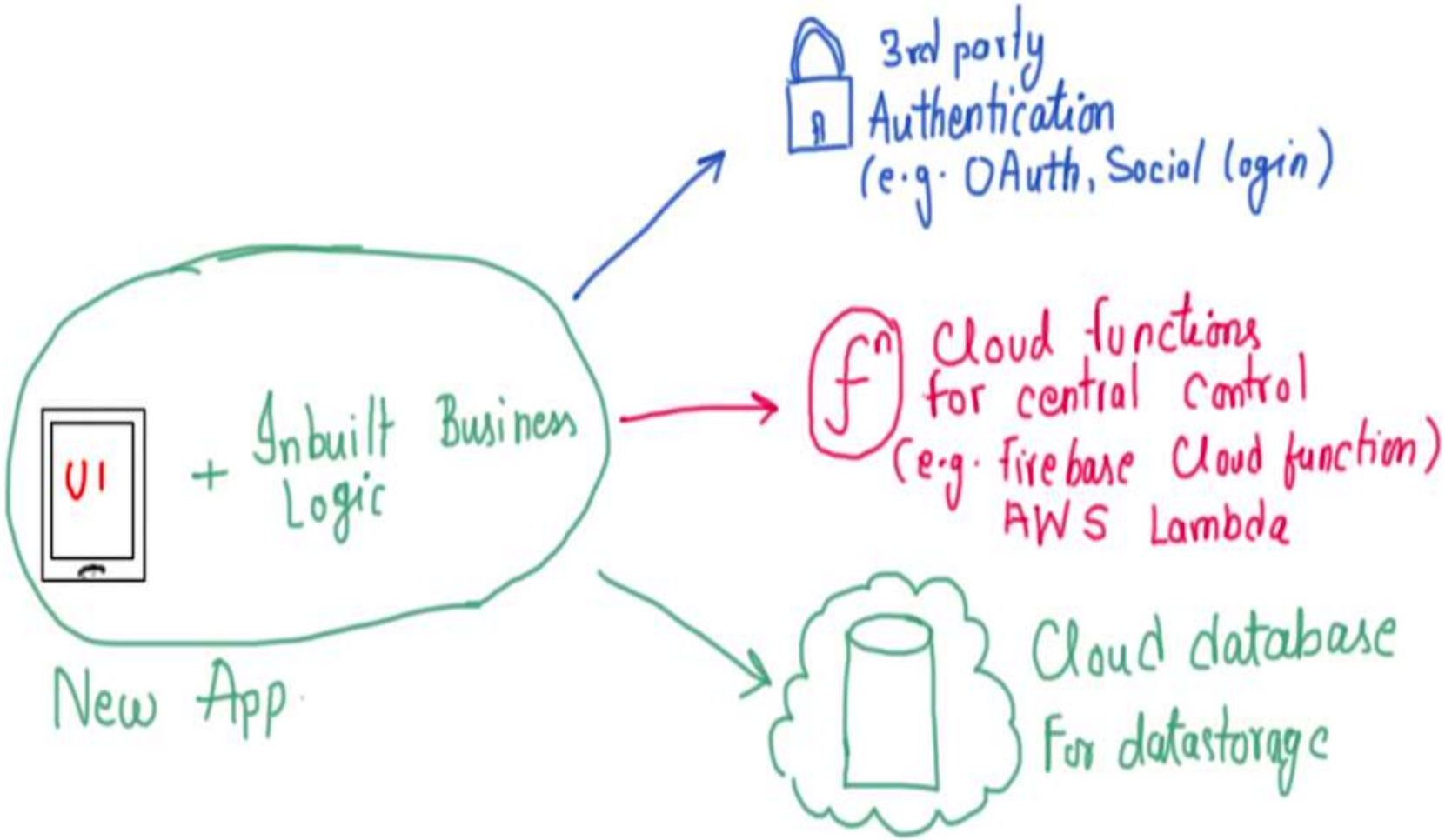
- **Өргөтгөх боломжтой.**

Олон бизнес эрхлэгчид өөрсдийн апп -уудыг Google эсвэл Facebook шиг чөлөөтэй, өргөтгөх чадвартай болгохыг хүсдэг. Сервергүй тооцоолол нь ачаалал ихсэхэд чөлөөтэйгөөр өргөтгөх боломжийг олгодог дэд бүтцийн шийдэлтэй байдаг. Уламжлалт апп-аас ялгаатай нь олон хүсэлтийг даах чадвартай байдаг байна.

SERVERLESS ДУТАГДАЛТАЙ ТАЛ

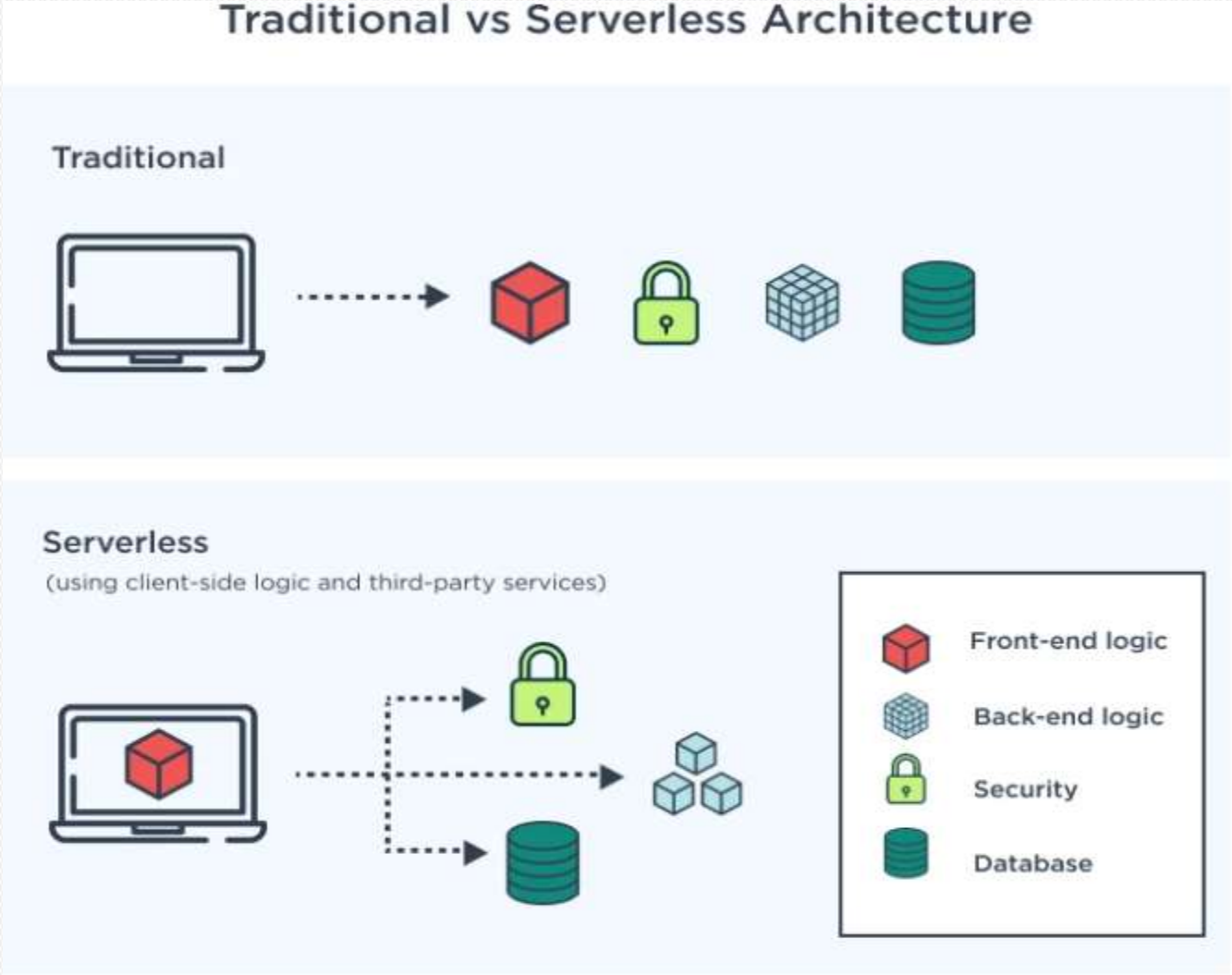
- Vendor lock-in
- Дэд бүтцээр хангагчаас хамааралтай. Өөрийн үйлчилгээг бүрэн хянахад бага зэрэг хүндрэлтэй. Нэг борлуулагчаас нөгөө рүү шилжихэд хүндрэлтэй байх талтай.
- Удаан deploy хийх апп-уудад тохиромжгүй.
- Хэрвээ 5 минутаас удаан deploy хийх функц байвал жижиг жижиг функцуудад хуваах шаардлагатай.
- Сервергүй програм хангамжийн архитектур нь нэг удаагийн даалгавар болон туслах процессыг гүйцэтгэхэд ашигтай байдаг. Энэ нь хурдацтай хөгжиж буй, хязгааргүй масштабтай байх шаардлагатай үйлчлүүлэгчдийн ачаалал ихтэй маш сайн ажилладаг.

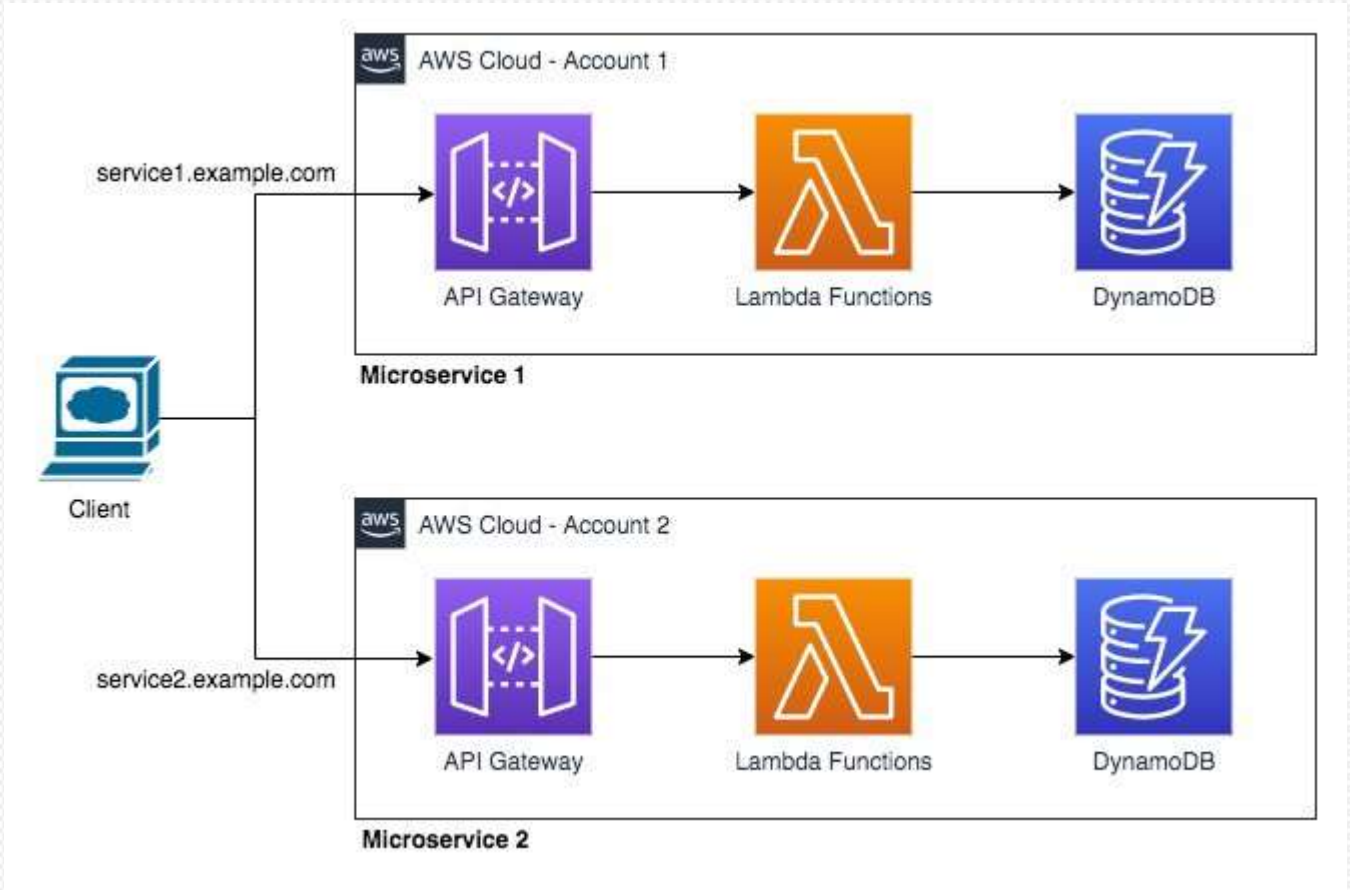
Serverless Architecture:

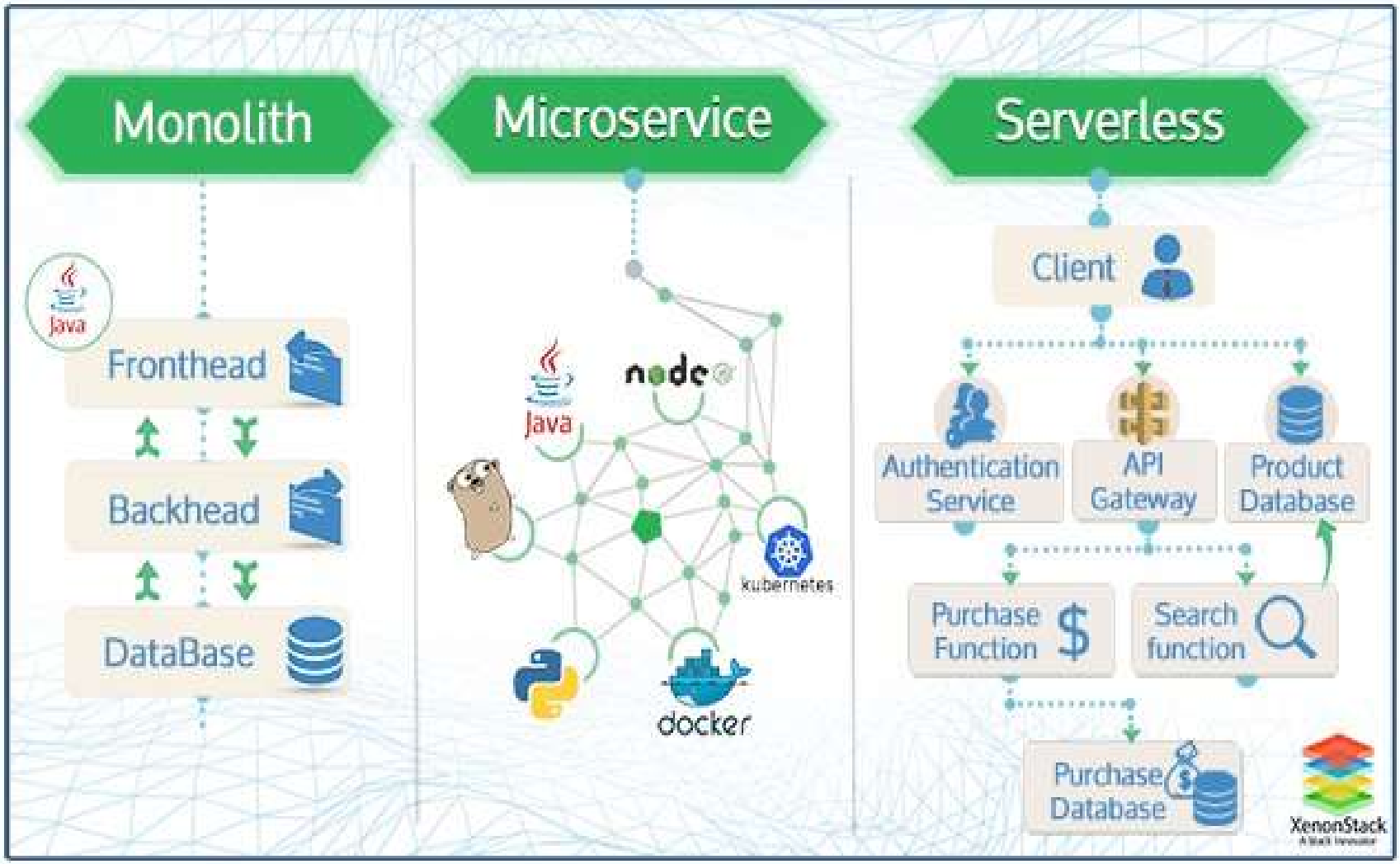


SERVERLESS ARCHITECTURES

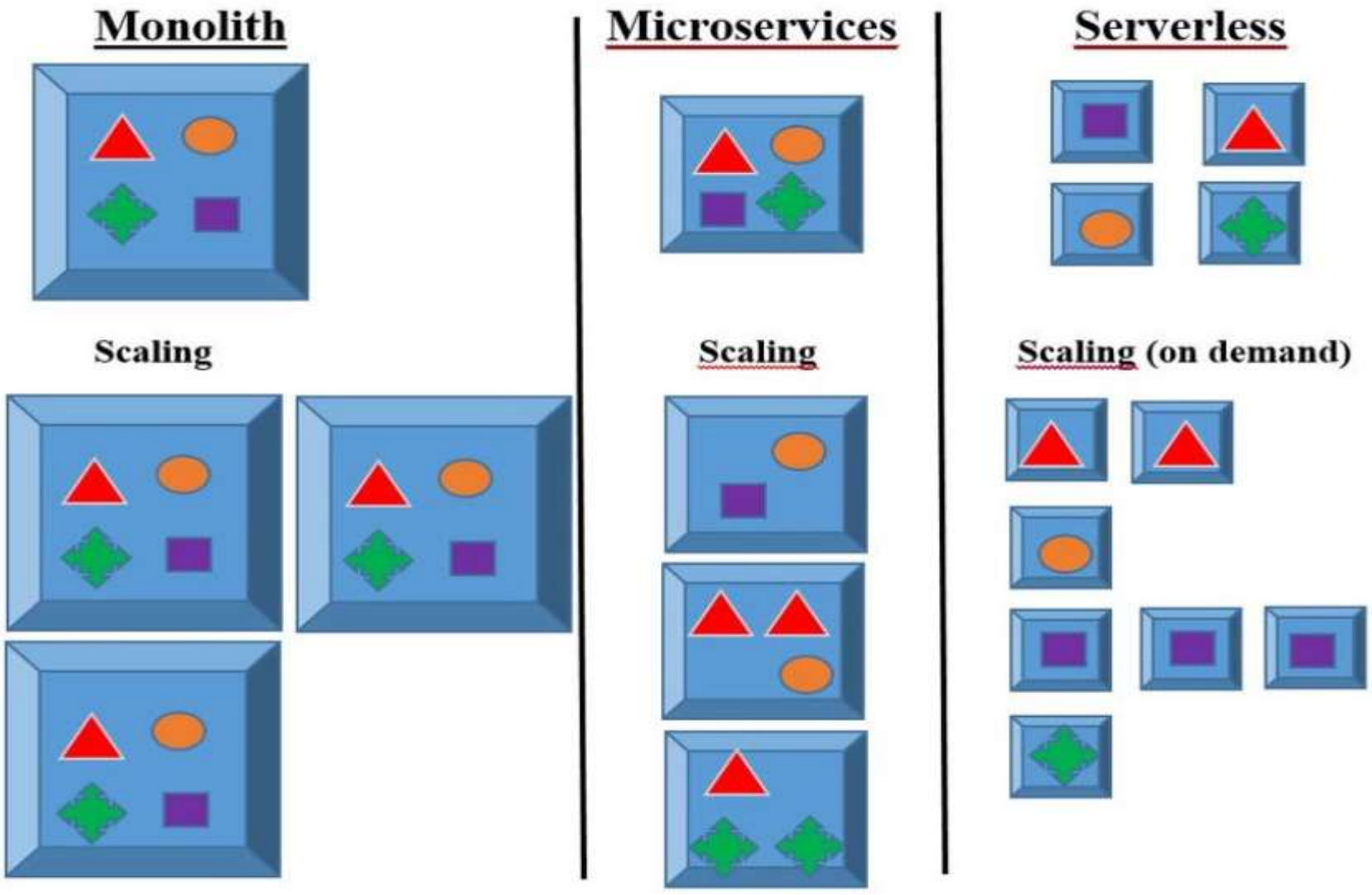
- https://martinfowler.com/articles/serverless.html




AWS Cloud - Account 1
service1.example.com
API Gateway
Lambda Functions
DynamoDB
Microservice 1
Client
AWS Cloud - Account 2
service2.example.com
API Gateway
Lambda Functions
DynamoDB
Microservice 2

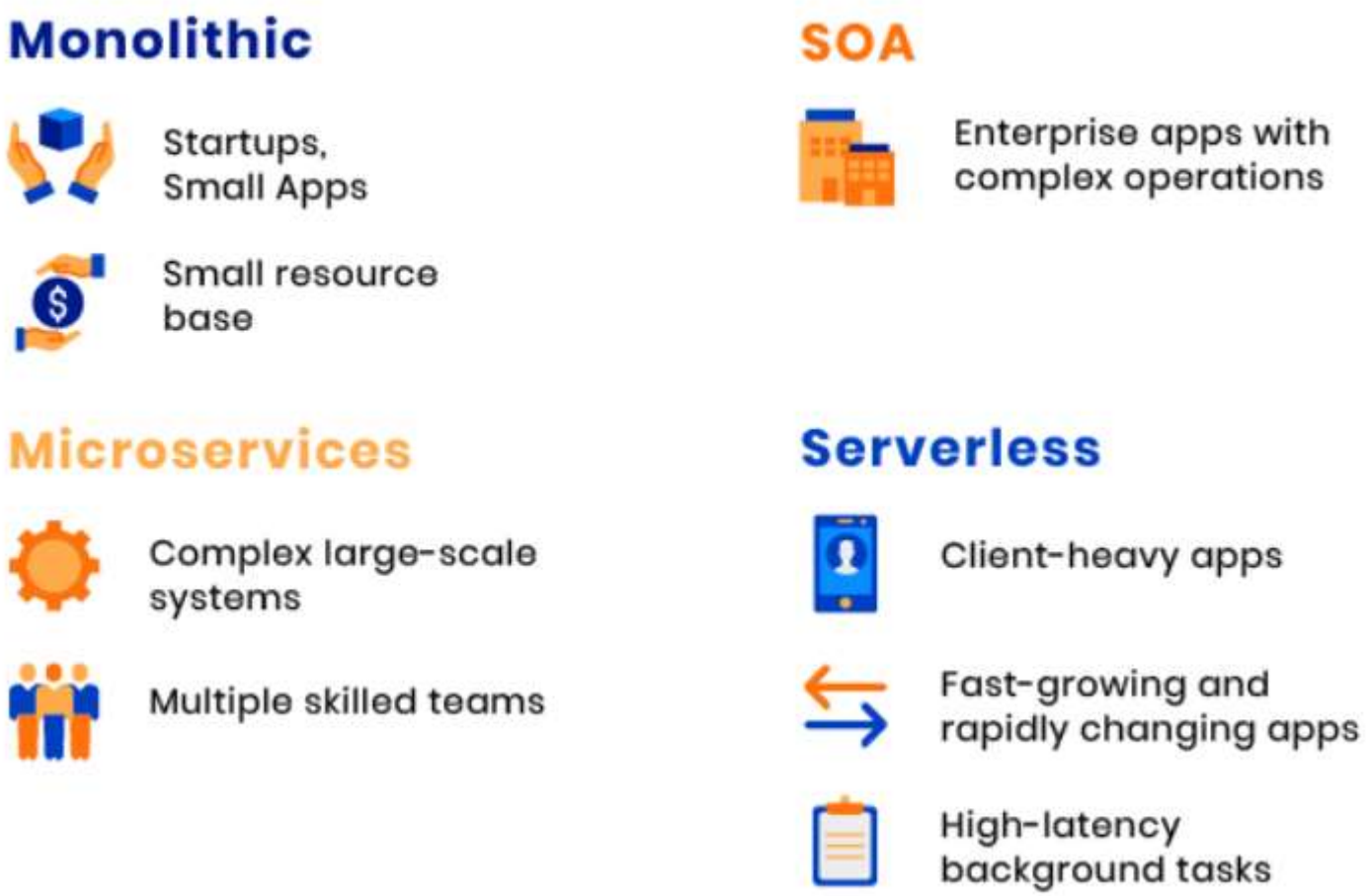

Monolith
Microservice
Serverless
Java
Fronthead
Backhead
DataBase
Java
node
kubernetes
docker
Client
Authentication Service
API Gateway
Product Database
Purchase Function
Search function
Purchase Database
XenonStack

ARCHITECTURES



Архитектуруудын харьцуулалт

- https://rubygarage.org/blog/monolith-soa-microservices-serverless



Архитектурын процесс

- Програм хангамжийн танилцуулга
- Шаардлага тодорхойлох (Functional болон Non functional шаардлага)
- Компонентуудыг тодорхойлох
- Технологийн сонголт
- Архитектурын зохиомжийг гаргах
- Архитектурын баримт бичгийг боловсруулах

Системийн шаардлагыг тодорхойлох

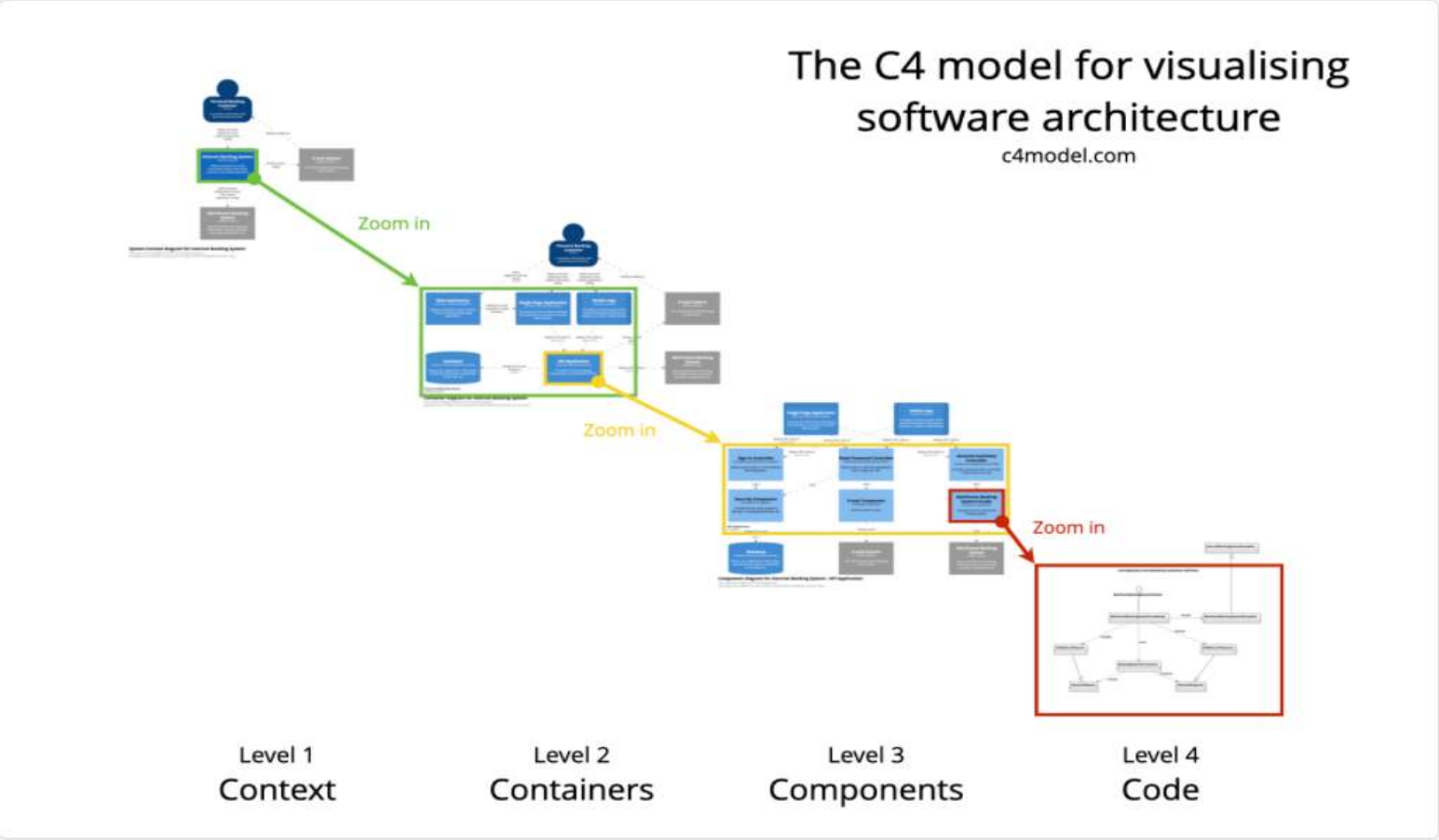
- Системийн functional шаардлагыг тодорхойлох
- Шаардлага= Систем юу хийхийг заадаг
- Энэ нь тухай системийг ашиглан хэрэглэгч юу хийлгэхийг хүсээд байгааг тодорхойлдог.
- Ихэвчлэн Систем аналистын мэргэжилтэнгүүд клиенттэй холбогдон тодруулдаг.
- Ихэвчлэн багийн болон албаны уулзалтуудаар системийн юу, юу хийх боломжтой мөн ямар шинж чанаруудтай байхыг тодорхойлдог.
- Системийн Non functional шаардлагыг тодорхойлох
- Тухайн системийн техникийн үзүүлэлт, хүчин чадал зэргийг тодорхойлно.
- Жнь: Системд хэдэн хэрэглэгчдийн тоо, ачаалал даах чадвар, гүйцэтгэлийн хурд гэх мэтчилэн
- Энэ ажлыг системийн архитектурч өөрөө хийнэ.

Компонентуудыг тодорхойлох

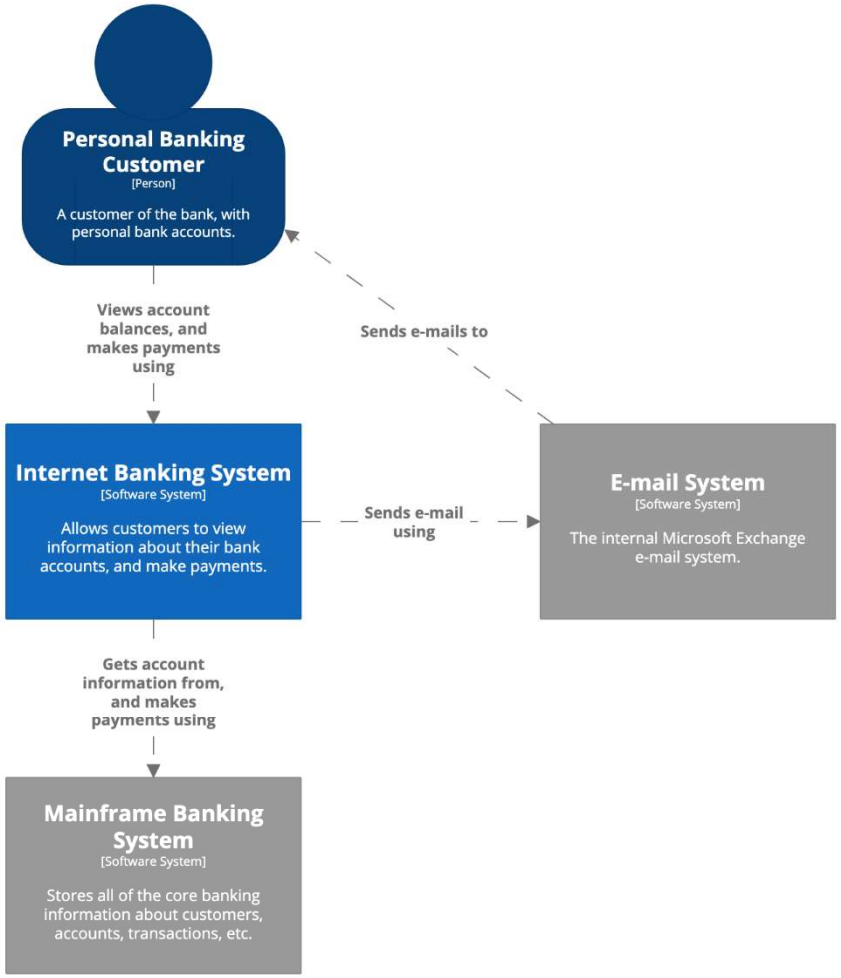
- Системүүдийн гүйцэтгэх даалгавруудыг тодорхойлж гаргана.
- Энэ хэсэг нь 2 үүрэгтэй. Үүнд:
  - 1.а Системийн функциональ үйл ажиллагаануудыг ойлгох
  - 1.б Клиент болон систем аналистуудтай харилцаж ойлгомжгүй зүйлийг тодруулах
  - 2 Техникийн бус шийдэл гаргах
  - Тухай системийг ямар архитектур дээр, ямар технологи, ямар өгөгдлийн сан дээр хийх эсэхийг тодруулах биш зөвхөн юу юу хийх бүрэлдэхүүн хэсгүүдтэй байхыг тодорхойлно.

THE C4 MODEL FOR VISUALISING SOFTWARE ARCHITECTURE

- https://c4model.com/ - Agile manifesto-https://agilemanifesto.org/
- Context, Containers, Components and Code-C4 загвар нь контекст, контейнер, компонент, програм хангамжийн код гэсэн архитектурын шаталсан багцаас бүрдэнэ.

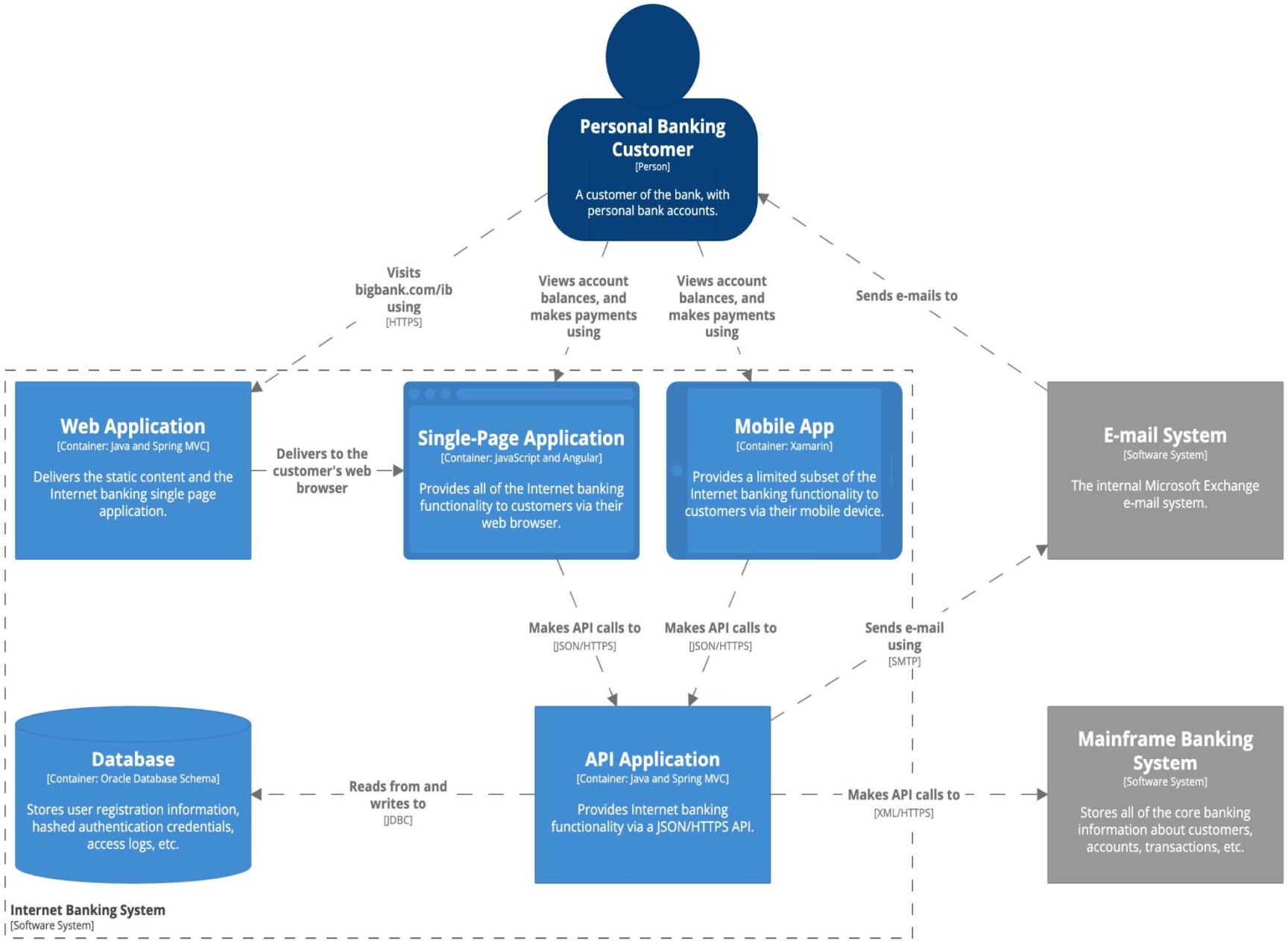


Level 1: System Context diagram



System Context diagram for Internet Banking System

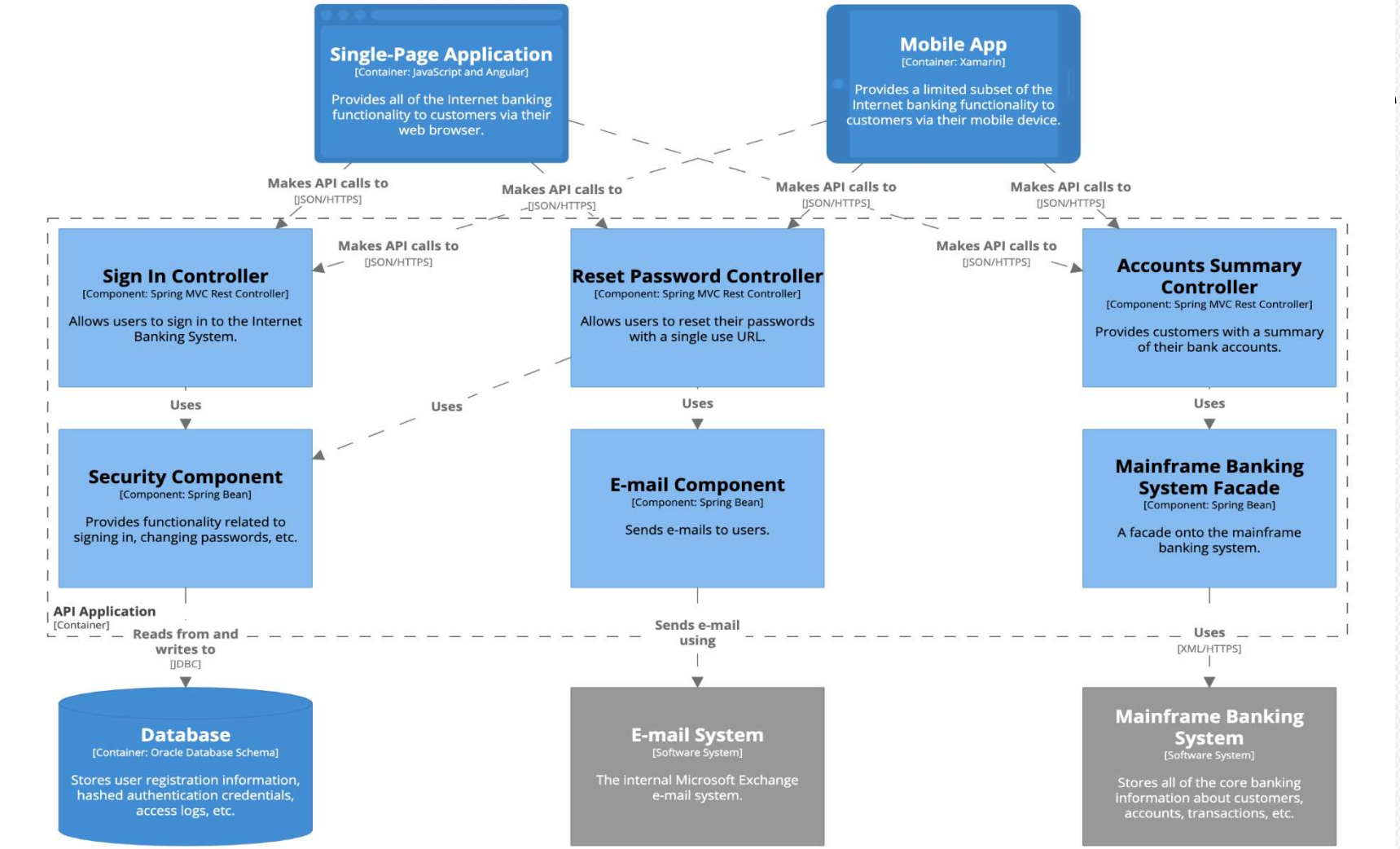
The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)



Container diagram for Internet Banking System

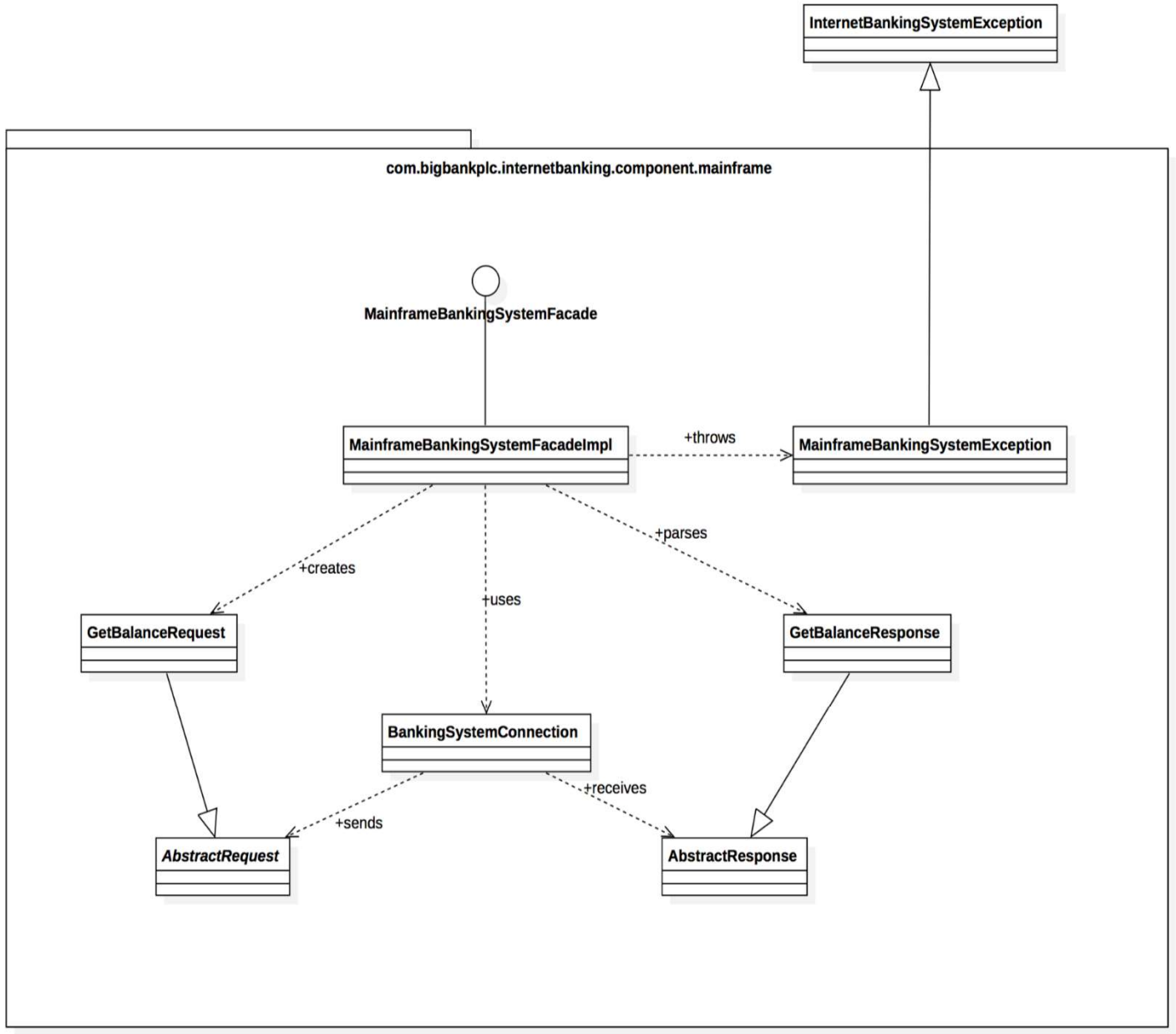
The container diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

LEVEL 3: COMPONENT DIAGRAM



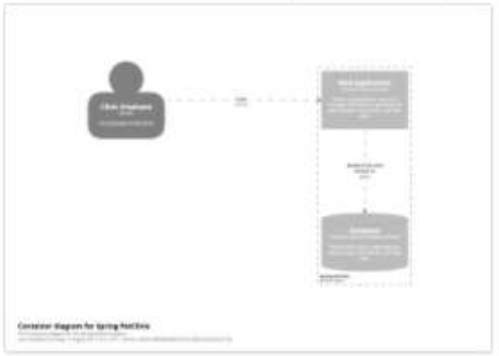
Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

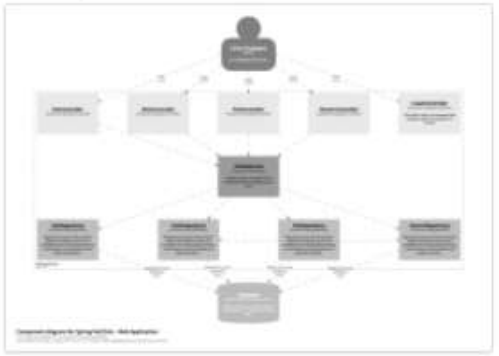

InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+receives
+sends
AbstractRequest
AbstractResponse

C4 AND UML HTTPS://C4MODEL.COM/#SYSTEMCONTEXTDIAGRAM

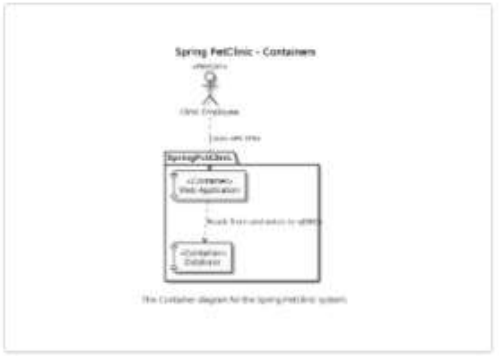
System Context diagram

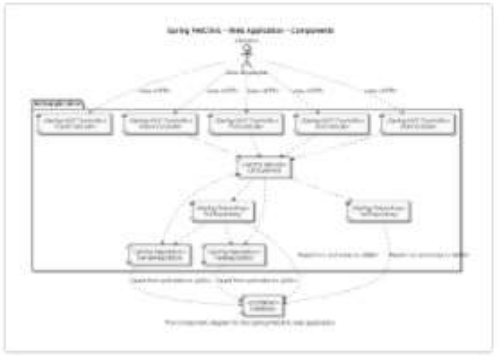


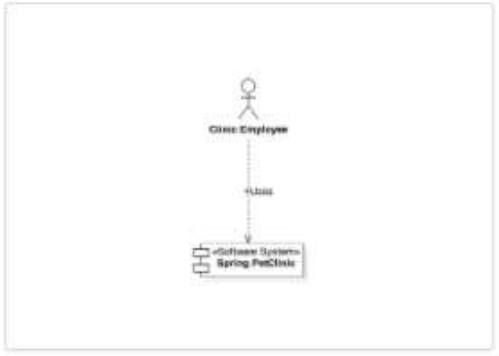
Container diagram

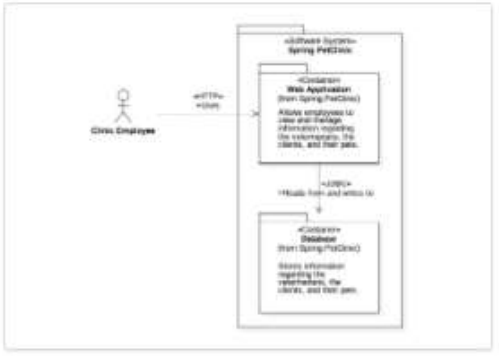


Component diagram









MODELING TOOL

Modelling tools (recommended - why?)

Structurizr

Structurizr is specifically designed to support the C4 model, and allows you to create diagrams as code (Java, .NET, TypeScript, PHP, Python, Go), diagrams as text (text-based DSL, YAML), or by using a web-based UI.

Ciaran Treanor has built a Structurizr DSL syntax highlighting extension for Visual Studio Code, while the open source Structurizr CLI can also output diagrams in PlantUML, Mermaid, and WebSequenceDiagrams formats.

Text-based | GUI-based | Cloud | On-premises | $ Freemium

Archi

Archi provides a way for you to create C4 model diagrams with ArchiMate. See C4 Model, Architecture Viewpoint and Archi 4.7 for more details..

GUI-based | On-premises | $ Free

Sparx Enterprise Architect

LieberLieber Software has built an extension for the C4 model, based upon the MDG Technology built into Sparx Enterprise Architect.

GUI-based | On-premises | $ Paid

MooD

MooD has support for the C4 model via a set of blueprints.

GUI-based | Cloud | On-premises | $ Paid

Diagramming tools

PlantUML

There are a number of extensions for PlantUML to asisst in the creation of C4 model diagrams:

C4-PlantUML by Ricardo Niepel
C4-PlantumlSkin by Savvas Kleanthous
c4builder by Victor Lupu
plantuml-libs by Thibault Morin

Text-based | Cloud | On-premises | $ Free

diagrams.net

diagrams.net includes support for the C4 model, and there are also a couple of plugins that allow you to create diagrams using pre-built shapes:

c4-draw.io by Chris Kaminski
c4-draw.io by Tobias Hochgürtel

GUI-based | Cloud | On-premises | $ Free

OmniGraffle

Dennis Laumen has created a C4 model stencil for OmniGraffle, that allows you to create diagrams using pre-built shapes.

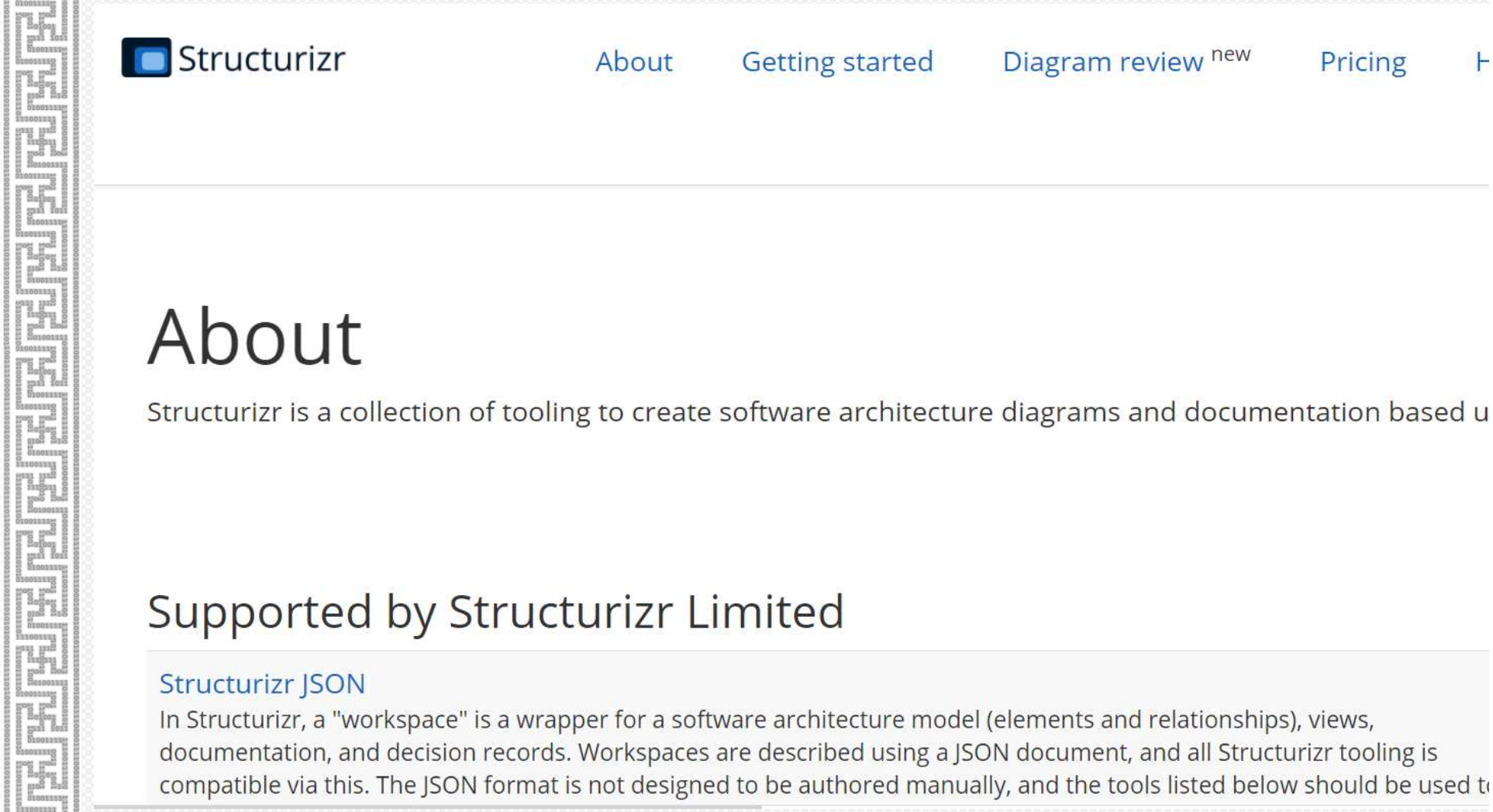
GUI-based | On-premises | $ Paid

Microsoft Visio

"pihalve" has created a C4 model template for Microsoft Visio, that allows you to create diagrams using pre-built shapes.

GUI-based | Cloud | On-premises | $ Paid

HTTPS://STRUCTURIZR.COM/



HTTPS://PLANTUML.COM/

PlantUML in a nutshell

PlantUML is a component that allows to quickly write :

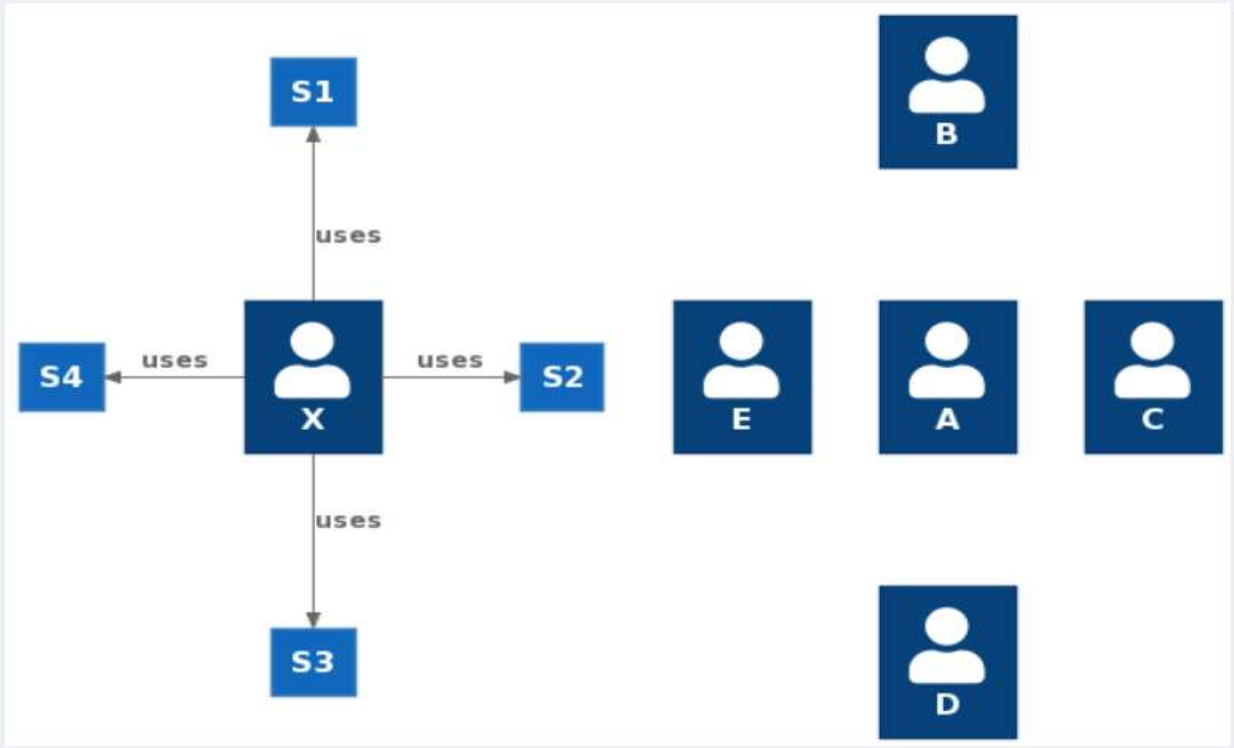
- Sequence diagram
- Usecase diagram
- Class diagram
- Activity diagram (here is the legacy syntax)
- Component diagram
- State diagram
- Object diagram
- Deployment diagram BETA
- Timing diagram BETA

The following non-UML diagrams are also supported:

- Wireframe graphical interface
- Archimate diagram

PLANT UML ДЭЭР БИЧСЭН C4 МОДЕЛ HTTPS://GITHUB.COM/PLANTUML-STDLIB/C4-PLANTUML

- @startuml
- !include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4
- HIDE_STEREOTYPE()
- Person(a, "A")
- Person(b, "B")
- Person(c, "C")
- Person(d, "D")
- Person(e, "E")
- Lay_U(a, b)
- Lay_R(a, c)
- Lay_D(a, d)
- Lay_L(a, e)
- Person(x, "X")
- System(s1, "S1")



HTTPS://MERMAID-JS.GITHUB.IO/MERMAID/#/

mermaid

Search Docs...

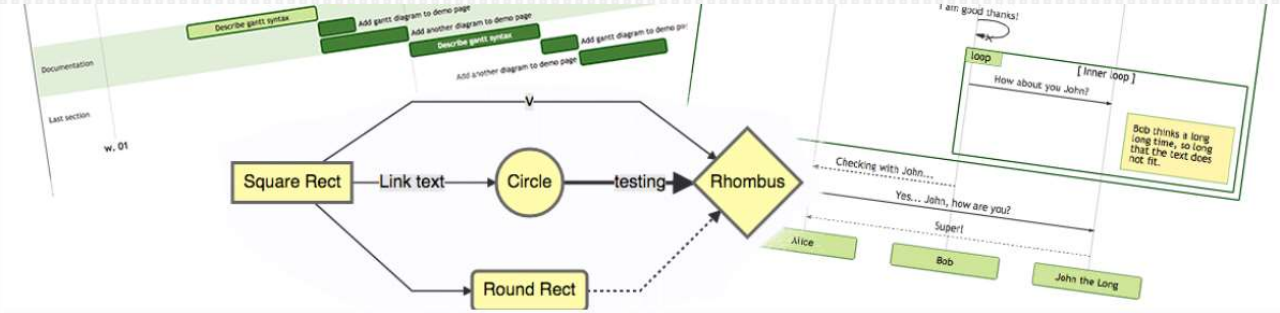
OVERVIEW

1. Overview
2. Use-Cases and Integrations
3. FAQ

GETTING STARTED

Tutorials

1. Getting started - easier
2. API-Usage
3. Configurations
4. Directives



🏆 Mermaid was nominated and won the JS Open Source Awards (2019) in the category "The most exciting use of technology"!!!

Thanks to all involved, people committing pull requests, people answering questions and special thanks to Tyler Long who is helping me maintain the project 🙏

About

Анхаарал хандуулсанд баярлалаа.