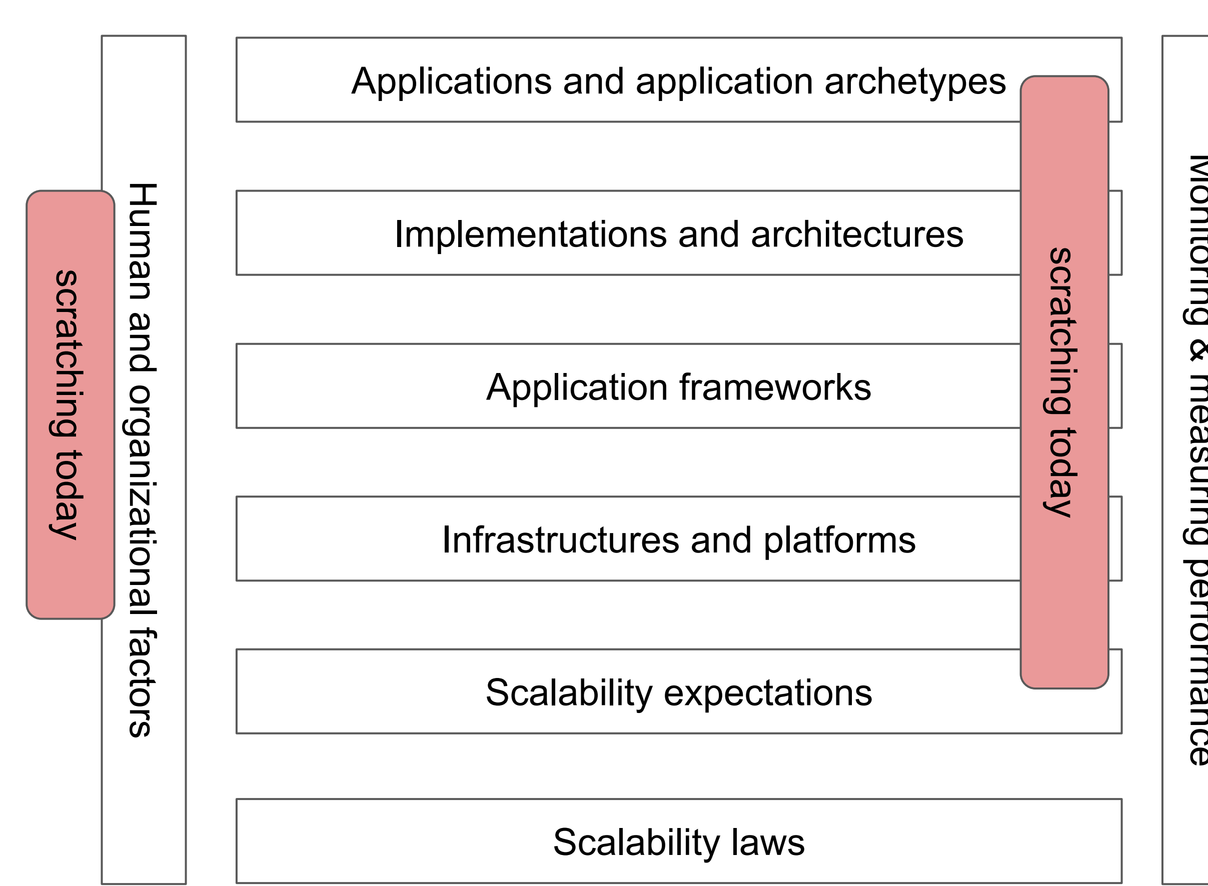
Designing and Building Scalable Web Applications

Lecture 4 / 14.11.2022

The Big Picture



Agenda

- Jamstack
- APIs
- API-first
- Server-side Architectural Patterns
- Scalability Dimensions

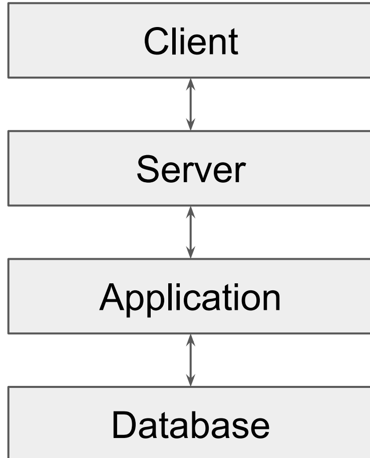
Jamstack and Classic web apps

Jamstack and Classic web apps

- Classic web apps
    - Client makes a request to a server
    - Server directs the request to an application
    - The application retrieves data related to the request from a database
    - The application creates a response and returns the response

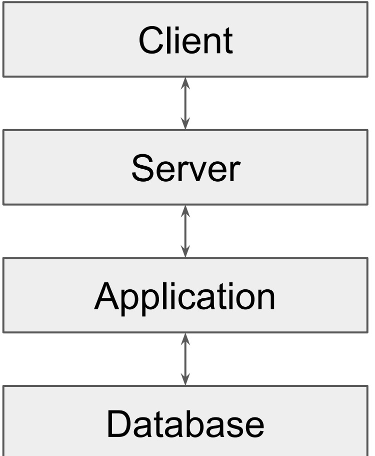
Jamstack and Classic web apps

- Classic web apps
  - Client makes a request to a server
  - Server directs the request to an application
  - The application retrieves data related to the request from a database
  - The application creates a response and returns the response



Jamstack and Classic web apps

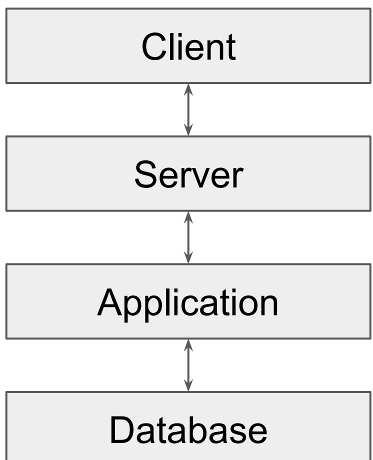
- Classic web apps
    - Client makes a request to a server
    - Server directs the request to an application
    - The application retrieves data related to the request from a database
    - The application creates a response and returns the response



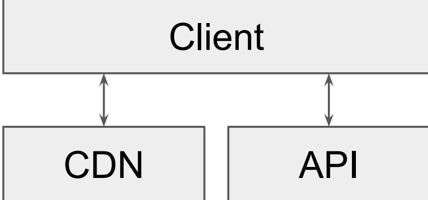
- Jamstack
    - Pre-rendered site deployed to a CDN
    - Client retrieves site from CDN
    - Site instructs client to use API(s)

Jamstack and Classic web apps

- Classic web apps
  - Client makes a request to a server
  - Server directs the request to an application
  - The application retrieves data related to the request from a database
  - The application creates a response and returns the response



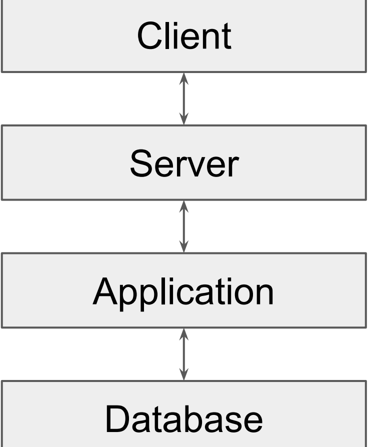
- Jamstack
  - Pre-rendered site deployed to a CDN
  - Client retrieves site from CDN
  - Site instructs client to use API(s)



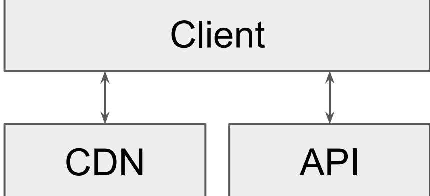
Jamstack and Classic web apps

https://jamstack.org/ – "Jamstack is an architectural approach that decouples the web experience layer from data and business logic, improving flexibility, scalability, performance, and maintainability."

- Classic web apps
  - Client makes a request to a server
  - Server directs the request to an application
  - The application retrieves data related to the request from a database
  - The application creates a response and returns the response



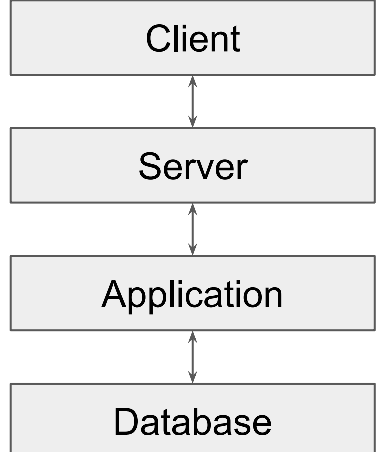
- Jamstack
  - Pre-rendered site deployed to a CDN
  - Client retrieves site from CDN
  - Site instructs client to use API(s)



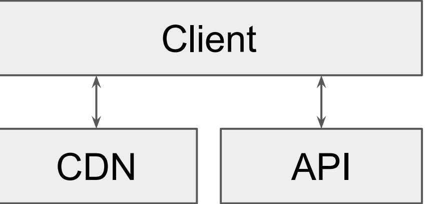
Jamstack and Classic web apps

https://jamstack.org/ – "Jamstack is an architectural approach that decouples the web experience layer from data and business logic, improving flexibility, scalability, performance, and maintainability."

- Classic web apps
  - Client makes a request to a server
  - Server directs the request to an application
  - The application retrieves data related to the request from a database
  - The application creates a response and returns the response



- Jamstack
  - Pre-rendered site deployed to a CDN
  - Client retrieves site from CDN
  - Site instructs client to use API(s)



Mathias Biilmann: The New Front-end Stack. Javascript, APIs and Markup – https://vimeo.com/163522126

Jamstack-..like?

Jamstack-..like?

- The term *Jamstack* has evolved over the years – three key features persist
  - Frontend and backend separated – frontend uses backend through an API
  - Frontend built and compiled into HTML, CSS, JavaScript
  - JavaScript included to sites on a need basis

https://jamstack.wtf

Jamstack-..like?

- The term *Jamstack* has evolved over the years – three key features persist
  - Frontend and backend separated – frontend uses backend through an API
  - Frontend built and compiled into HTML, CSS, JavaScript
  - JavaScript included to sites on a need basis

https://jamstack.wtf

Jamstack-..like?

- The term *Jamstack* has evolved over the years – three key features persist
  - Frontend and backend separated – frontend uses backend through an API
  - Frontend built and compiled into HTML, CSS, JavaScript
  - JavaScript included to sites on a need basis

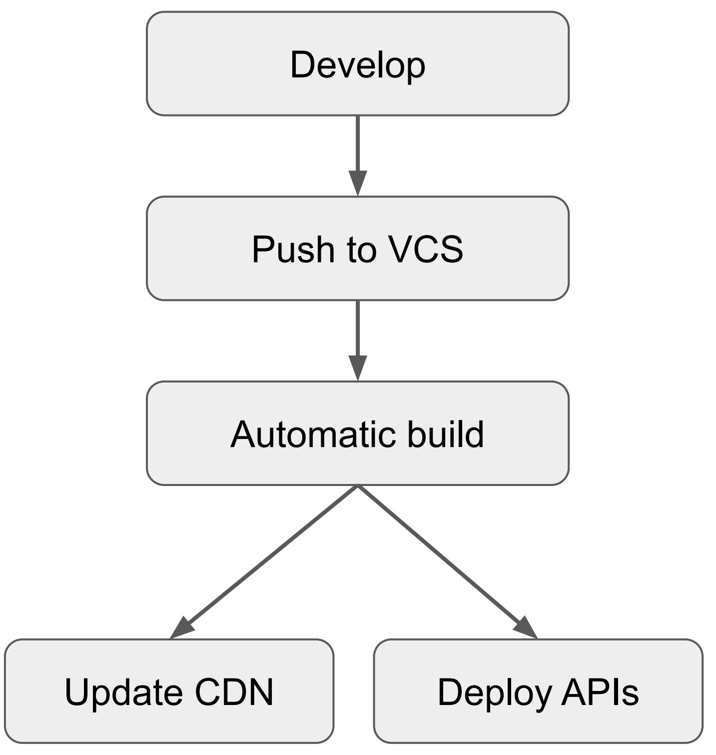
- The term also includes a workflow with meaningful tools and a CDN

https://jamstack.wtf

Jamstack-..like?

- The term *Jamstack* has evolved over the years – three key features persist
  - Frontend and backend separated – frontend uses backend through an API
  - Frontend built and compiled into HTML, CSS, JavaScript
  - JavaScript included to sites on a need basis

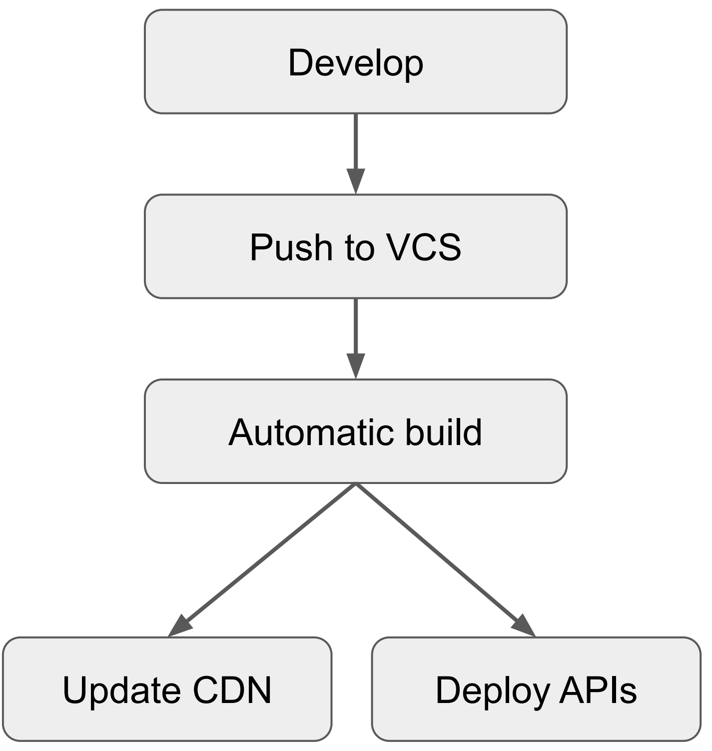
- The term also includes a workflow with meaningful tools and a CDN



https://jamstack.wtf

Jamstack-..like?

- The term *Jamstack* has evolved over the years – three key features persist
  - Frontend and backend separated – frontend uses backend through an API
  - Frontend built and compiled into HTML, CSS, JavaScript
  - JavaScript included to sites on a need basis
- The term also includes a workflow with meaningful tools and a CDN

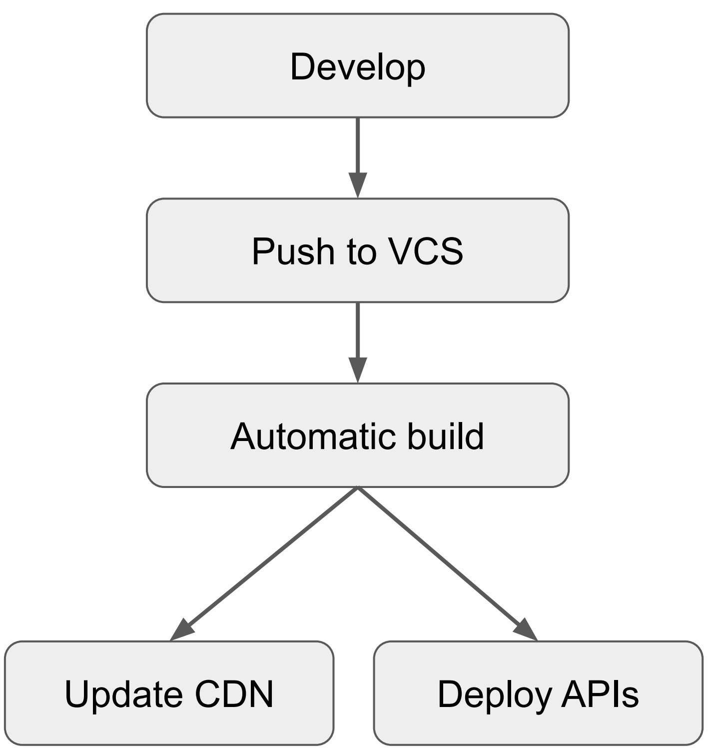


In the second course project, there is no need to use a VCS, to deploy the project, or to use a CDN → the project structure should allow this though.

https://jamstack.wtf

Jamstack-..like?

- The term *Jamstack* has evolved over the years – three key features persist
  - Frontend and backend separated – frontend uses backend through an API
  - Frontend built and compiled into HTML, CSS, JavaScript
  - JavaScript included to sites on a need basis
- The term also includes a workflow with meaningful tools and a CDN



In the second course project, there is no need to use a VCS, to deploy the project, or to use a CDN → the project structure should allow this though.

Thus, "Jamstack-like"

APIs

APIs

- Application Programming Interfaces (APIs) – allow access to resources and functionality

APIs

- Application Programming Interfaces (APIs) – allow access to resources and functionality

- Mainly two types of APIs
    - Synchronous APIs (REST, SOAP, RPC, …)
    - Asynchronous APIs (Event-driven APIs)

Synchronous APIs

Synchronous APIs

- Request-response paradigm: Client sends a request, the server returns a response

Synchronous APIs

GET /api/tasks/1 HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json
(headers)

{ “name”: “task name”, “completed”: false }

- Request-response paradigm: Client sends a request, the server returns a response

Synchronous APIs

- Request-response paradigm: Client sends a request, the server returns a response
- Representational State Transfer (REST) often used
  - Resources identified through URIs
  - Agreed upon representation formats (e.g. JSON)
  - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information

Synchronous APIs

- Request-response paradigm: Client sends a request, the server returns a response
- Representational State Transfer (REST) often used
  - Resources identified through URIs
  - Agreed upon representation formats (e.g. JSON)
  - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information

Synchronous APIs

HTTP/1.1 200 OK
Content-Type: application/json
(headers)

{ “name”: “task name”, “completed”: false }

- Request-response paradigm: Client sends a request, the server returns a response
- Representational State Transfer (REST) often used
  - Resources identified through URIs
  - Agreed upon representation formats (e.g. JSON)
  - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information

Synchronous APIs

HTTP/1.1 200 OK
Content-Type: application/json
(headers)

{ “name”: “task name”, “completed”: false }

- Request-response paradigm: Client sends a request, the server returns a response
- Representational State Transfer (REST) often used
    - Resources identified through URIs
    - Agreed upon representation formats (e.g. JSON)
    - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information

See also Roy Fielding's dissertation on the topic
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Example: REST Api

https://fitech101.aalto.fi/web-software-development/27-application-programming-interfaces/3-sample-task-api/

Synchronous APIs

HTTP/1.1 200 OK
Content-Type: application/json
(headers)

{ “name”: “task name”, “completed”: false }

- Request-response paradigm: Client sends a request, the server returns a response
- Representational State Transfer (REST) often used
  - Resources identified through URIs
  - Agreed upon representation formats (e.g. JSON)
  - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information

Synchronous APIs

- Request-response paradigm: Client sends a request, the server returns a response
- Representational State Transfer (REST) often used
  - Resources identified through URIs
  - Agreed upon representation formats (e.g. JSON)
  - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information
- Needs a polling mechanism to keep track of changes on server (e.g. long polling, short polling)

Synchronous APIs

GET /api/tasks/1 HTTP/1.1

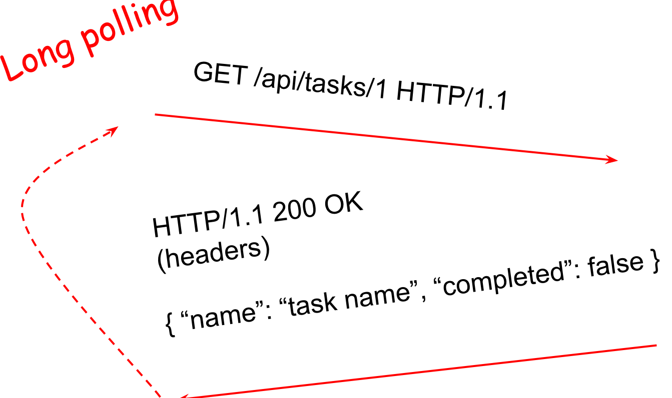
HTTP/1.1 200 OK
Content-Type: application/json
(headers)

{ "name": "task name", "completed": false }

- Request-response paradigm: Client sends a request, the server returns a response

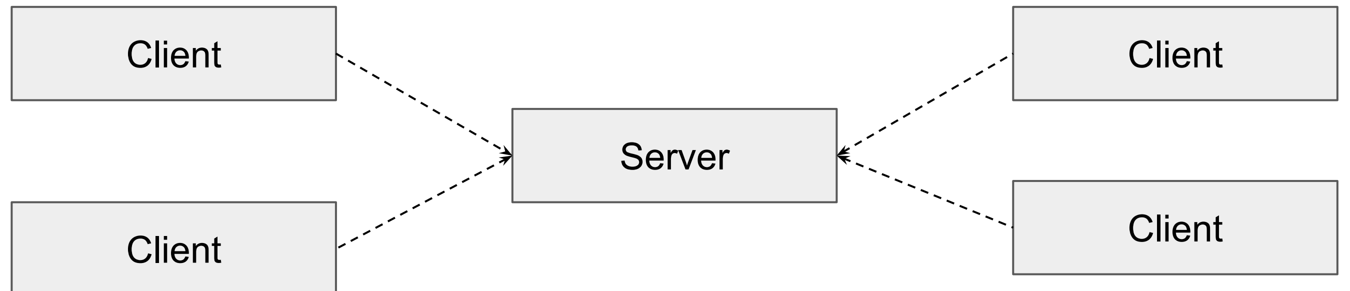
- Representational State Transfer (REST) often used
  - Resources identified through URIs
  - Agreed upon representation formats (e.g. JSON)
  - Standard methods (e.g. HTTP methods) used for semantics and for exchanging information

- Needs a polling mechanism to keep track of changes on server (e.g. long polling, short polling)



Server waits before sending a response, maintaining connection

Example: Long polling



Asynchronous APIs

Asynchronous APIs

- Client requests a bi-directional connection from the server, the server allows it
  - Traditionally WebSocket, soon (?) WebSocketStream
  - With HTTP/3, also WebTransport
  - (WebRTC for P2P, depending on use case)

Asynchronous APIs

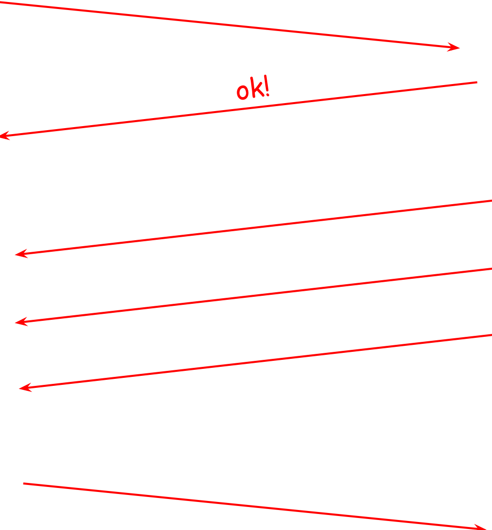
- Client requests a bi-directional connection from the server, the server allows it
    - Traditionally WebSocket, soon (?) WebSocketStream
    - With HTTP/3, also WebTransport
    - (WebRTC for P2P, depending on use case)

https://websockets.spec.whatwg.org/

https://developer.chrome.com/en/articles/websocketstream/

https://www.w3.org/TR/webtransport/

Asynchronous APIs



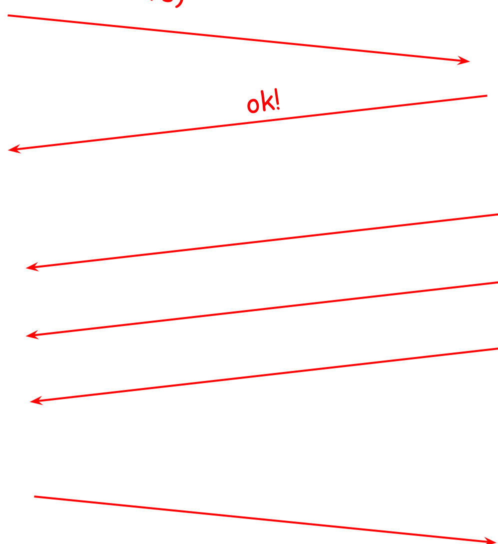
- Client requests a bi-directional connection from the server, the server allows it
  - Traditionally WebSocket, soon (?) WebSocketStream
  - With HTTP/3, also WebTransport
  - (WebRTC for P2P, depending on use case)

https://websockets.spec.whatwg.org/

https://developer.chrome.com/en/articles/websocketstream/

https://www.w3.org/TR/webtransport/

Asynchronous APIs



- Client requests a bi-directional connection from the server, the server allows it
  - Traditionally WebSocket, soon (?) WebSocketStream
  - With HTTP/3, also WebTransport
  - (WebRTC for P2P, depending on use case)

- No need for client-side polling of server, but, need to keep the connection and have the server up and running.

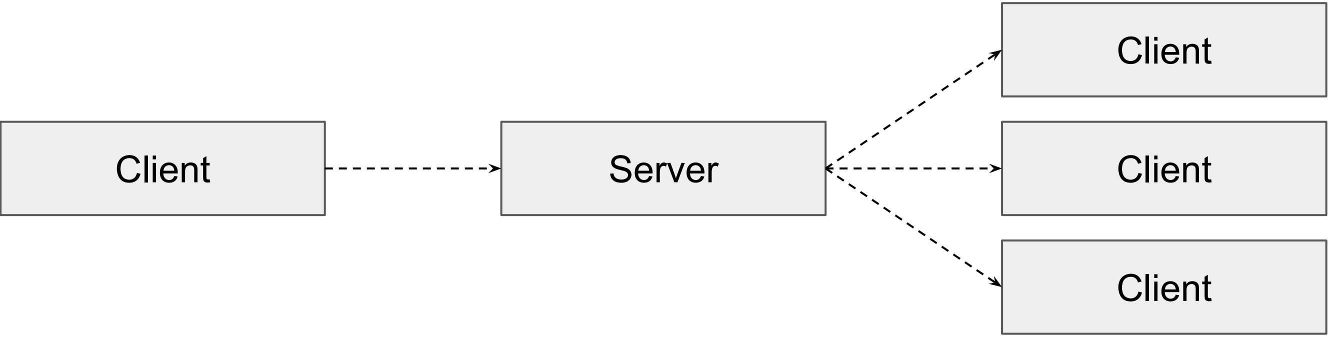
https://websockets.spec.whatwg.org/

https://developer.chrome.com/en/articles/websocketstream/

https://www.w3.org/TR/webtransport/

https://deno.land/api?s=Deno.upgradeWebSocket

Example: WebSockets



API-first

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- API-first approach sees APIs as products used by other systems (instead of them providing a set of endpoints supporting a single system).

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- API-first approach sees APIs as products used by other systems (instead of them providing a set of endpoints supporting a single system).
- Fosters thinking about how APIs are documented, how they are used, how they are built, how they are maintained, etc.

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- API-first approach sees APIs as products used by other systems (instead of them providing a set of endpoints supporting a single system).

- Fosters thinking about how APIs are documented, how they are used, how they are built, how they are maintained, etc.

- The idea has been around for a while – e.g. Bezos API Mandate at Amazon (2002).

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- API-first approach sees APIs as products used by other systems (instead of them providing a set of endpoints supporting a single system).
- Fosters thinking about how APIs are documented, how they are used, how they are built, how they are maintained, etc.
- The idea has been around for a while – e.g. Bezos API Mandate at Amazon (2002).

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols — doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- API-first approach sees APIs as products used by other systems (instead of them providing a set of endpoints supporting a single system).
- Fosters thinking about how APIs are documented, how they are used, how they are built, how they are maintained, etc.
- The idea has been around for a while – e.g. Bezos API Mandate at Amazon (2002).

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols — doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- APIs designed to be used as products → easier adoption in new applications and devices

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- APIs designed to be used as products → easier adoption in new applications and devices

- Designed collaboratively with API consumers → key functionality identified before coding

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- APIs designed to be used as products → easier adoption in new applications and devices

- Designed collaboratively with API consumers → key functionality identified before coding

- Adoption in (larger) companies; need for an API style guide including versioning strategies etc

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

API-first

- APIs designed to be used as products → easier adoption in new applications and devices

- Designed collaboratively with API consumers → key functionality identified before coding

- Adoption in (larger) companies; need for an API style guide including versioning strategies etc

Also a key part of MACH:

- **M**icroservices
- **A**PI-first
- **C**loud-native SaaS
- **H**eadless

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

https://machalliance.org/

API-first

- APIs designed to be used as products → easier adoption in new applications and devices
- Designed collaboratively with API consumers → key functionality identified before coding
- Adoption in (larger) companies; need for an API style guide including versioning strategies etc

Also a key part of MACH:

- **M**icroservices
- **A**PI-first
- **C**loud-native SaaS
- **H**eadless

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

https://machalliance.org/

API-first

- APIs designed to be used as products → easier adoption in new applications and devices
- Designed collaboratively with API consumers → key functionality identified before coding
- Adoption in (larger) companies; need for an API style guide including versioning strategies etc

Also a key part of MACH:

- **M**icroservices
- **A**PI-first
- **C**loud-native SaaS
- **H**eadless

"MACH technologies support a composable enterprise in which every component is pluggable, scalable, replaceable, and can be continuously improved through agile development to meet evolving business requirements."

Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

Server-side Architectural Patterns

Server-side Architectural Patterns – some concepts

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):
  - Layered architecture

How code is structured and how a request goes through the layers of an application.

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):
  - Layered architecture
  - Monolithic architecture

How code is structured and how a request goes through the layers of an application.

Application as a single coherent unit.

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):
    - Layered architecture
    - Monolithic architecture
    - Microservice architecture

How code is structured and how a request goes through the layers of an application.

Application as a single coherent unit.

Application as a collection of services.

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):
  - Layered architecture
  - Monolithic architecture
  - Microservice architecture
  - Serverless architecture

How code is structured and how a request goes through the layers of an application.

Application as a single coherent unit.

Application as a collection of services.

Application on a third party service, often launched up on demand.

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):
  - Layered architecture
  - Monolithic architecture
  - Microservice architecture
  - Serverless architecture

How code is structured and how a request goes through the layers of an application.

Application as a single coherent unit.

Application as a collection of services.

Application on a third party service, often launched up on demand.

- All of the above work for scalable applications (depending on archetype!)

Server-side Architectural Patterns – some concepts

- Web Software Development (CS-C3170):
  - Layered architecture
  - Monolithic architecture
  - Microservice architecture
  - Serverless architecture

How code is structured and how a request goes through the layers of an application.

Application as a single coherent unit.

Application as a collection of services.

Application on a third party service, often launched up on demand.

- All of the above work for scalable applications (depending on archetype!)

- Also, ..
  - Event-driven architecture
  - Microkernel / plugin architecture
  - Space-based architecture

Event-driven Architecture

Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published

Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published

- Typically realized using a publish / subscribe (Pub/Sub) pattern

Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published

- Typically realized using a publish / subscribe (Pub/Sub) pattern

    - Client (subscriber) subscribes to one or more topics of interest by making a request to the server

Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published

- Typically realized using a publish / subscribe (Pub/Sub) pattern

    - Client (subscriber) subscribes to one or more topics of interest by making a request to the server

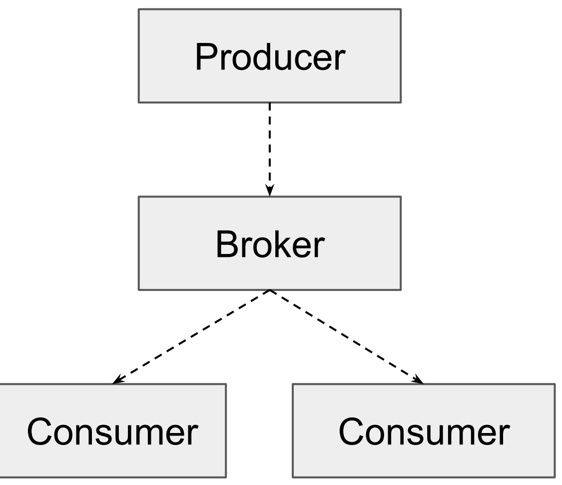
    - Server (publisher) defines available topics. When new data arrives (is *produced*) for a topic, data is sent to all clients subscribed to the topic

Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published
- Typically realized using a publish / subscribe (Pub/Sub) pattern
  - Client (subscriber) subscribes to one or more topics of interest by making a request to the server
  - Server (publisher) defines available topics. When new data arrives (is *produced*) for a topic, data is sent to all clients subscribed to the topic
- Needs a message broker (a service for passing and storing messages): Plenty of existing platforms including Kafka (which is in Week 4 readings).

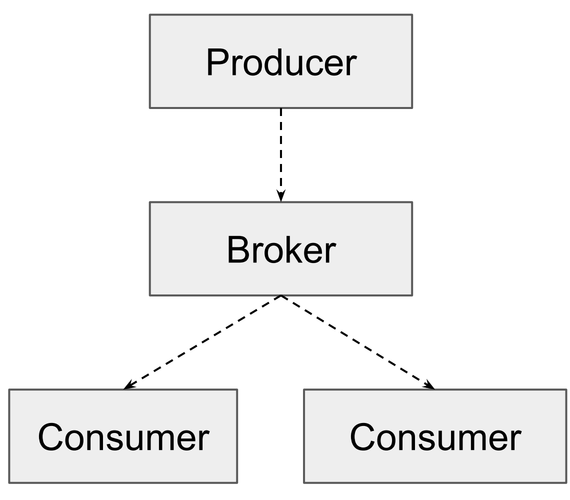
Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published
- Typically realized using a publish / subscribe (Pub/Sub) pattern
  - Client (subscriber) subscribes to one or more topics of interest by making a request to the server
  - Server (publisher) defines available topics. When new data arrives (is *produced*) for a topic, data is sent to all clients subscribed to the topic
- Needs a message broker (a service for passing and storing messages): Plenty of existing platforms including Kafka (which is in Week 4 readings).



Event-driven Architecture

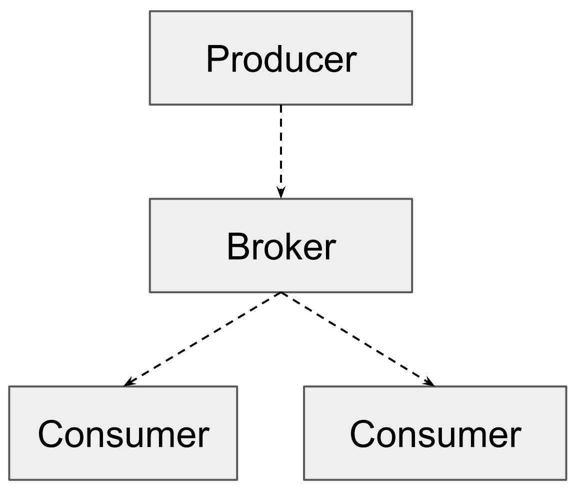
- When the state of an application changes (i.e. an *event* happens), information about the change is published
- Typically realized using a publish / subscribe (Pub/Sub) pattern
    - Client (subscriber) subscribes to one or more topics of interest by making a request to the server
    - Server (publisher) defines available topics. When new data arrives (is *produced*) for a topic, data is sent to all clients subscribed to the topic
- Needs a message broker (a service for passing and storing messages): Plenty of existing platforms including Kafka (which is in Week 4 readings).



Naive implementation possible with WebSockets and simple server

Event-driven Architecture

- When the state of an application changes (i.e. an *event* happens), information about the change is published
- Typically realized using a publish / subscribe (Pub/Sub) pattern
    - Client (subscriber) subscribes to one or more topics of interest by making a request to the server
    - Server (publisher) defines available topics. When new data arrives (is *produced*) for a topic, data is sent to all clients subscribed to the topic
- Needs a message broker (a service for passing and storing messages): Plenty of existing platforms including Kafka (which is in Week 4 readings).



Naive implementation possible with WebSockets and simple server

Good for figuring out the principles – an existing platform is better for actual use

https://deno.land/api?s=Deno.upgradeWebSocket

Example: (simple) Messaging with WebSockets

Event-driven Architecture

- Note! Event-driven architectures primarily for passing *messages* about events
- Passing large files (e.g. images) as messages *maybe* not a good idea
- Rather, store the data and pass a reference (or a link) to the data

https://dzone.com/articles/processing-large-messages-with-apache-kafka

https://www.kai-waehner.de/blog/2020/05/19/apache-kafka-event-streaming-pharmaceuticals-pharma-life-sciences-use-cases-architecture/

Event-driven Architecture

For the second course project, given the size of the programming exercises, passing full solutions as messages is ok.

- Note! Event-driven architectures primarily for passing *messages* about events

- Passing large files (e.g. images) as messages *maybe* not a good idea

- Rather, store the data and pass a reference (or a link) to the data

https://dzone.com/articles/processing-large-messages-with-apache-kafka

https://www.kai-waehner.de/blog/2020/05/19/apache-kafka-event-streaming-pharmaceuticals-pharma-life-sciences-use-cases-architecture/

We’ll discuss microkernel / plugin architecture and space-based architecture next week.

Scalability Dimensions

Scalability dimensions

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):
    - Numerical dimension – the number of users of the system and the number of objects and services

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):
  - Numerical dimension – the number of users of the system and the number of objects and services
  - Geographical dimension – the distance over which the system is scattered

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):
  - Numerical dimension – the number of users of the system and the number of objects and services
  - Geographical dimension – the distance over which the system is scattered
  - Administrative dimension – the number of organizations that control pieces of the system

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):
  - Numerical dimension – the number of users of the system and the number of objects and services
  - Geographical dimension – the distance over which the system is scattered
  - Administrative dimension – the number of organizations that control pieces of the system

With more services, the likelihood that something is down increases

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system

- Scale in distributed systems (1994):

  - Numerical dimension – the number of users of the system and the number of objects and services

  - Geographical dimension – the distance over which the system is scattered

  - Administrative dimension – the number of organizations that control pieces of the system

With more services, the likelihood that something is down increases

With larger distances come larger latencies

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):
    - Numerical dimension – the number of users of the system and the number of objects and services
    - Geographical dimension – the distance over which the system is scattered
    - Administrative dimension – the number of organizations that control pieces of the system

With more services, the likelihood that something is down increases

With larger distances come larger latencies

More governing bodies may lead to larger amounts of changes, conflicts, etc..

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Scalability dimensions

- Multiple dimensions in scaling a system
- Scale in distributed systems (1994):
  - Numerical dimension – the number of users of the system and the number of objects and services
  - Geographical dimension – the distance over which the system is scattered
  - Administrative dimension – the number of organizations that control pieces of the system

With more services, the likelihood that something is down increases

With larger distances come larger latencies

More governing bodies may lead to larger amounts of changes, conflicts, etc..

B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*. 1994.

Not fully applicable to web software...

Scalability dimensions – how to scale

Abbott, M.L. and Fisher, M.T., 2009. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

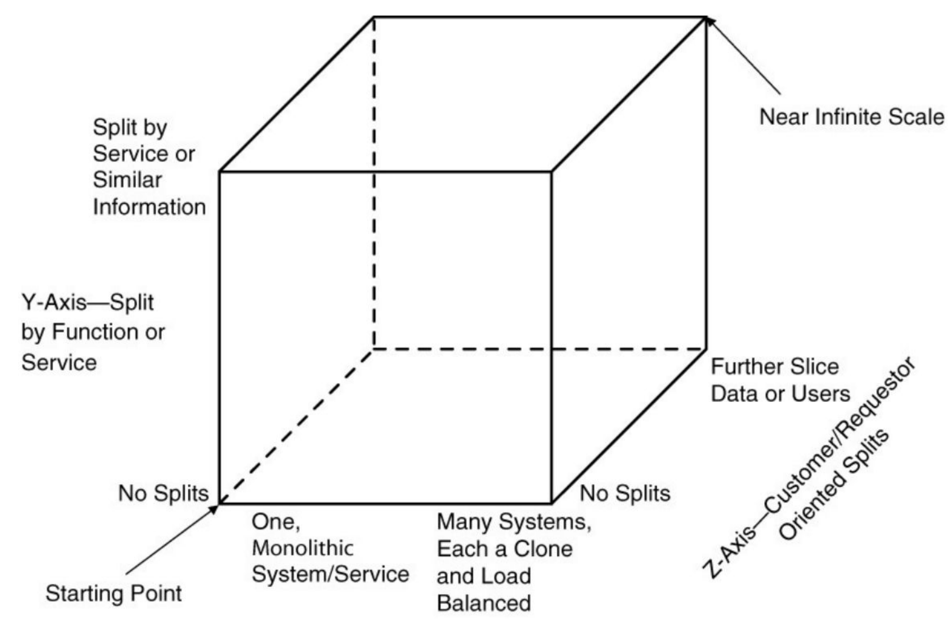
Scalability dimensions – how to scale

- AKF Scale Cube:
  - Cloning of systems or data
  - Separation of work by responsibility, action, or data
  - Separation of work by customer or requestor

Abbott, M.L. and Fisher, M.T., 2009. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

Scalability dimensions – how to scale

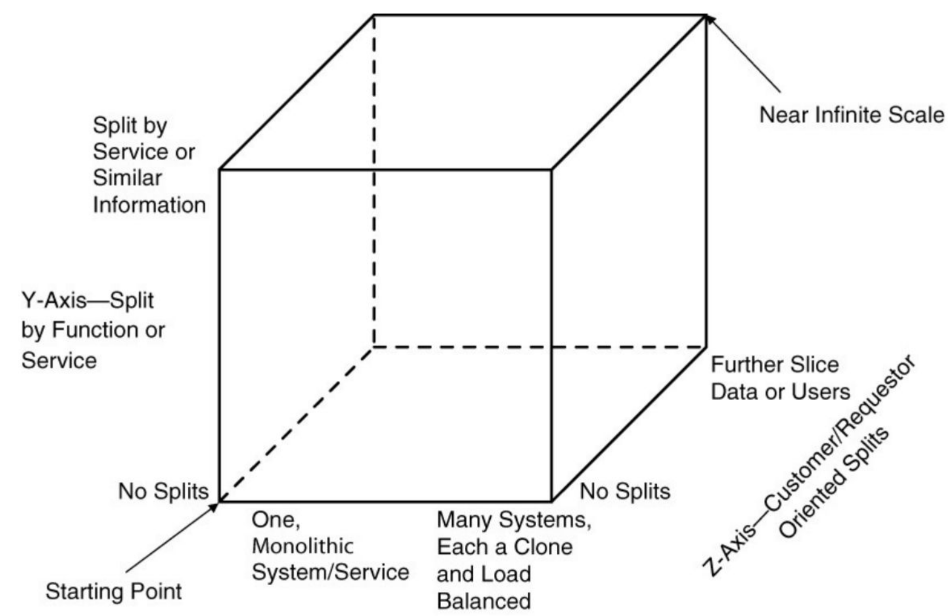
- AKF Scale Cube:
  - Cloning of systems or data
  - Separation of work by responsibility, action, or data
  - Separation of work by customer or requestor



Abbott, M.L. and Fisher, M.T., 2009. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

Scalability dimensions – how to scale

- AKF Scale Cube:
  - Cloning of systems or data
  - Separation of work by responsibility, action, or data
  - Separation of work by customer or requestor

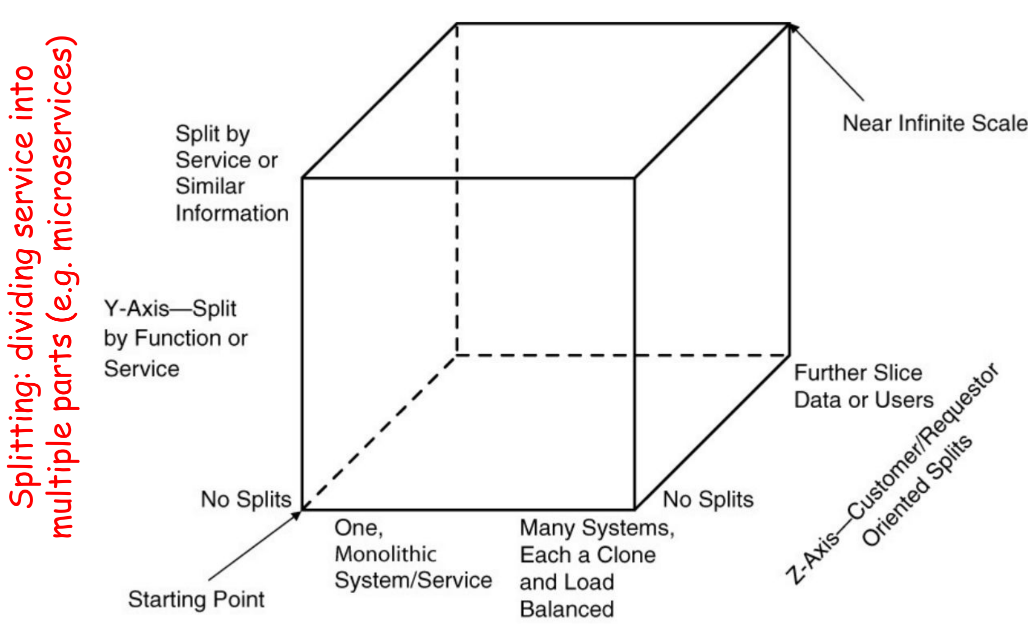


Duplication: creating multiple copies of a service

Abbott, M.L. and Fisher, M.T., 2009. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

Scalability dimensions – how to scale

- AKF Scale Cube:
  - Cloning of systems or data
  - Separation of work by responsibility, action, or data
  - Separation of work by customer or requestor

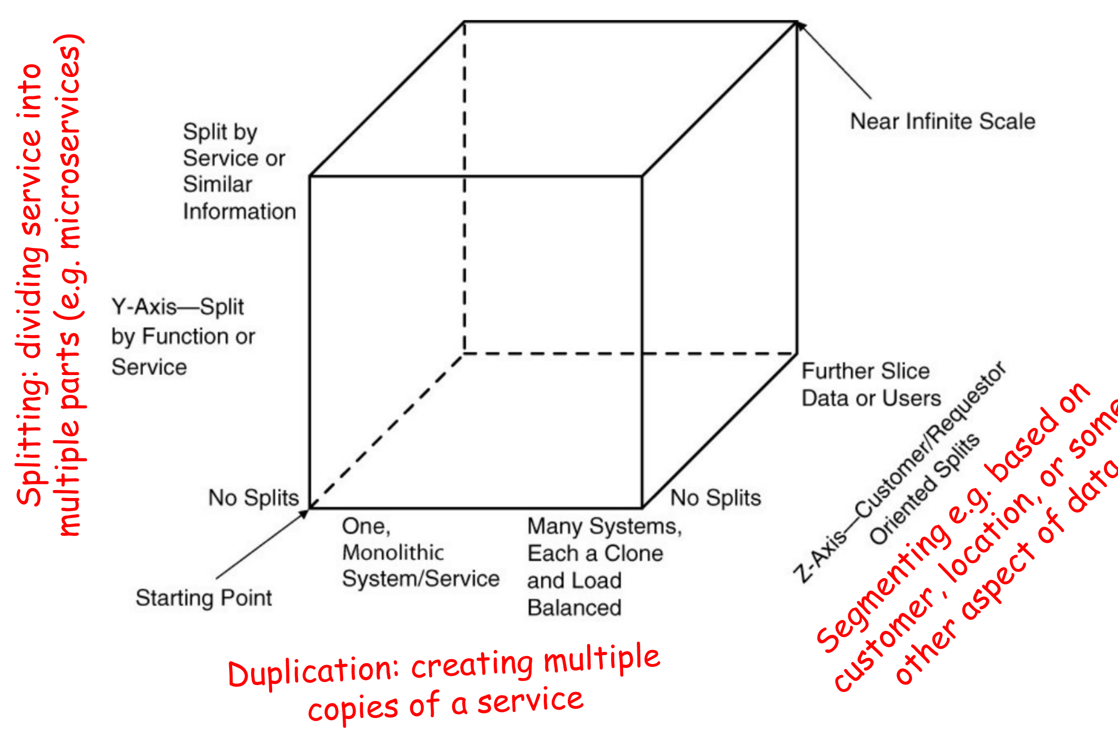


Duplication: creating multiple copies of a service

Abbott, M.L. and Fisher, M.T., 2009. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

Scalability dimensions – how to scale

- AKF Scale Cube:
  - Cloning of systems or data
  - Separation of work by responsibility, action, or data
  - Separation of work by customer or requestor



Abbott, M.L. and Fisher, M.T., 2009. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

Recap

- Jamstack
- APIs
- API-first
- Server-side Architectural Patterns
- Scalability Dimensions