fwdays

Course

Software Architecture Design in practice

Oleksandr Savchenko

Oleksandr Savchenko

Course Structure

fwdays

Module 1 - Software Architecture Fundamentals

- Definition of software architecture
- Types of architectures
- Architectural schools
- Architecture development process (from ASRs to ADRs)
- Initial Information analysis

Module 2 - Architecturally Significant Requirements

- Overview of ASRs
- Analysis of functional requirements and identification of functional components
- Constraints and Concerns
- Identification and analysis of risks and assumptions
- Quality Attributes fundamentals
- Quality Attributes catalogues
- Overview of Main Quality Attributes
- Flow for identification and analysis

Module 3 - Architectural Governance

- Architectural committee
- Architectural schools, frameworks and methods
- Architectural viewpoints and Tools
- Architectural Decision Records (ADRs)
- Architectural principles

Module 4 - Architectural Design

- Levels of Architectural Decisions
- Main design concepts to use and their catalogues
- General architectural decisions
- Front-End, Back-End architectural decisions
- Infrastructure design
- Data, Integration, AI architecture
- Platform specific design decisions

Module 5 - Architecture Implementation

- How Architect should work with delivery teams
- Sprint 0 activities
- Setup Development, Testing and DevOps strategies
- Experiments (PoC/Prototypes) driven delivery
- Release preparation

Module 6 - Architecture for Brownfield

- The process of creating architecture for Brownfield
- Architecture evaluation methods
- Creation of checklists for Brownfield assessment
- Observability-driven approach (Metrics and Architectural Fitness Functions)

Summary of the course and learning materials

Today's Agenda

- [18:30] Theoretical section
- Q&A and ⏸️Break
- [19:30] Practical exercise
- [20:30] Homework overview

fwdays

Module 4

Architectural Design

Course: Software Architecture Design in practice

by Oleksandr Savchenko

Oleksandr Savchenko

Agenda

fwdays

Part 1:

1. Levels of Architectural Decisions
2. Main design concepts to use and their catalogues
3. General architectural decisions
4. Back-End, Front-End architectural decisions

Part 2:

5. Infrastructure design
6. Component integrations
7. Data Management
8. AI architecture
9. Platform specific design decisions

Software Architecture Development process

fwdays

2 Builds **Design Team** and agreed **roadmap**

3
- Creates questions / points for clarification
- Chooses correct elicitation methods (workshop, brainstorming sessions, send questions via email)
- Collects as much as possible information

Analyses

1

Analysis of Initial information

Design Team

1. Stakeholders identification
2. Business domain analysis
3. Main objectives
4. Type of product / components (new or existing)
5. Clarify past, current, future states
6. Request and analyse useful artefacts

Stakeholders

C level: CEO, CTO, COO

Product Team

Development Team

...

4 Analyses info from different Stakeholders and sources

5 Identifies, analyses and prioritises architecturally significant requirements

ASRs

6 Agreeds Architectural Governance: design methods, Architecture toolset, design scope, Architectural Principles

We are here...

7 Creates Design

ADRs

Architectural Views

8 Creates Engineering strategies / guides

Development, Testing, DevOps, Data, ...

9 Creates Estimation, Roadmap, Team Composition

10 Contributes to Delivery & Product strategies

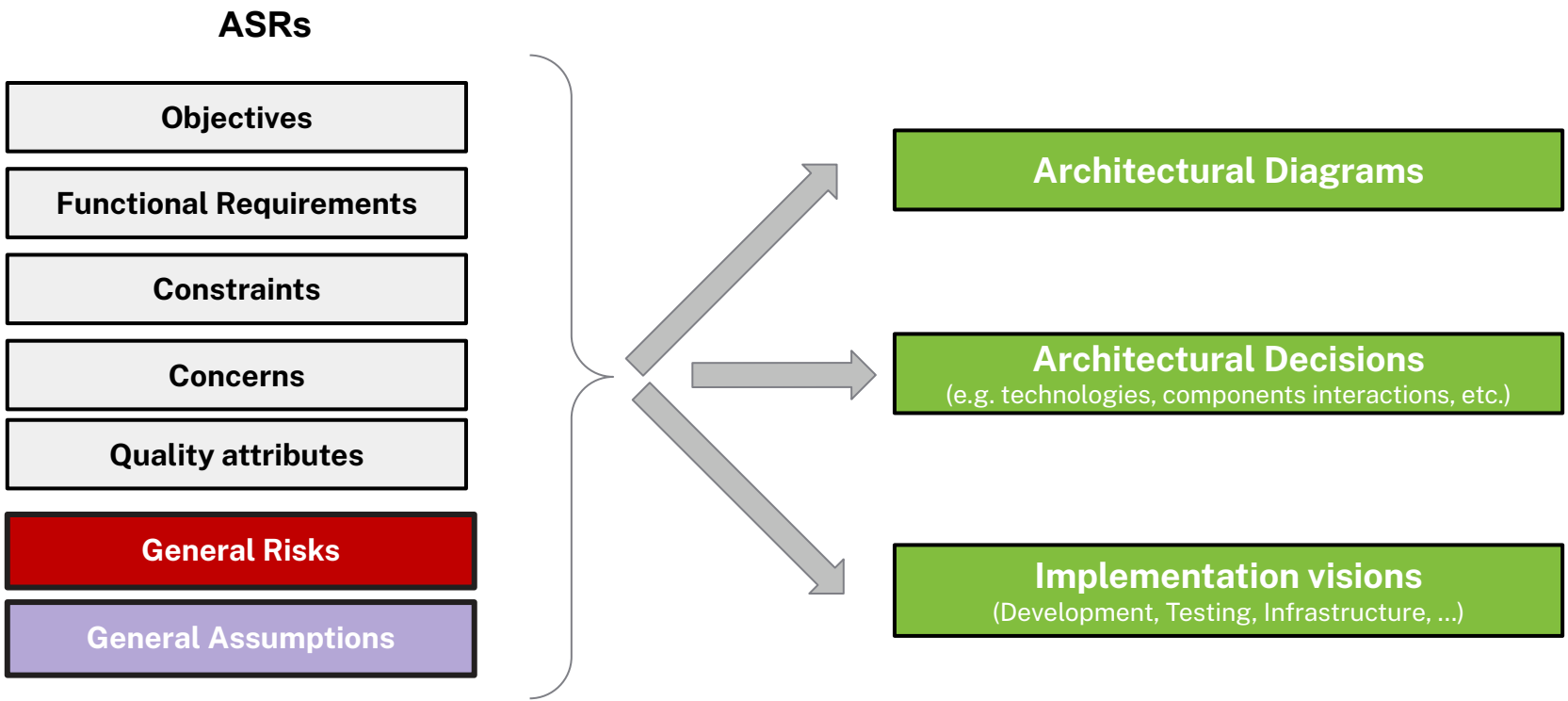
fwdays

Levels of Architectural Decisions

Oleksandr
Savchenko

From ASRs to Design Decisions

fwdays



Main goal of the Design is to cover ASRs

fwdays

Use cases

ID	Overview
UC-01	A search feature allows Customers to search books by entering the query in the search bar.
UC-02	The app should allow Customers to create accounts and login using credentials like email and password or through social networks integration.
UC-03	The system sends a confirmation email when a new customer account is created.
UC-04	The Customer can review items in the cart, change their number, or remove them before checkout.

Constraints

ID	Overview
CRN-01	Deployment and computing on the Azure.
CRN-02	Use current used technologies for BE (.NET) and FE (React).
CRN-03	Release 1 after 9 months.
CRN-04	Budget for the discovery and project delivery should not more than forecasted and approved budget for next year.

Concerns

ID	Overview
CON-01	Current team does not have experience with modern technologies (docker, K8s).
CON-02	Cost for infrastructure should be forecasted and optimised.
CON-03	Should be integration with some 3rd party solutions that have no final API.
CON-04	New product strategy that should be defined (new product metrics).
CON-05	Internal team is focus on current platform right now and potentially will support and extend this new platform in the future.
CON-06	Potential highload on release date because marketing company.

Quality Attributes

ID	Name and Overview	QAS ID	QA Scenario
QA-100	**Interoperability** - enable seamless integration with existing audit tools, third-party systems, and external data sources to enhance interoperability.	QAS-101	QAS-41: User completes registration and system component for integration with Mailchimp API sends a request to the Mailchimp API to send email to the User (Customer). The request to the Mailchimp should be completed within 2 seconds on average. The integration should handle intermittent network failures gracefully, attempting retries up to 3 times before reporting a failure to the user.
QA-200	**Security** - Implement robust security measures, including encryption, access controls, and secure data transmission, to safeguard sensitive audit information.	QAS-201	Customer personal information after registration should be stored in secure encrypted storage based
		QAS-202	Unauthorized user sends a series of invalid login credentials to gain access to a secured resource via authentication subsystem. The system detects the invalid login attempts, after 5 not correct attempts blocks further access from the suspicious source for 3 minutes and triggering an alert to the system administrator in real-time.

Architectural Decision Records (ADRs)

Category	ADR ID	Title	ASRs to cover
Platform specific	ADR-401	Use event-driven design and messaging system to send emails to Customers via async way	UC-05, CON-03, CON-06, QAS-101
...			

What should You know to create decisions?

fwdays

1. List of ADRs

What are decisions should be created to cover ASRs based on priority?

Category	ADRs
General architectural styles & technologies	• ADR-101: N-tier architecture • ADR-102: Architectural style for BE services • ADR-103: BE tech stack • ADR-104: Architectural style for FE web-application • ADR-105: FE tech stack • ADR-106: Cross FE-BE communication approach • ADR-107: Event-driven design
Infrastructure	• ADR-201: Computing Deployment Model • ADR-202: Cloud Provider • ADR-203: Deployment model for BE services • ADR-204: Computing approach for BE services • ADR-205: Storing and rendering FE web-app • ADR-206: Infrastructure as Code approach • ADR-207: Scaling general approach • ADR-208: Monitoring approach • ADR-209: Logging approach
Data Management & Integrations	• ADR-301: API Gateway implementation • ADR-302: Main RDBMS • ADR-303: Main NoSQL storage • ADR-304: Cache for applications • ADR-305: Messaging system • ADR-306: Integrations with Payment Gateway
Platform related	• ADR-401: Backoffice approach • ADR-402: Identity and access management system • ADR-403: SEO approach for public web site • ADR-404: Internationalisation

2. For each ADR - understand Context and choose alternatives

What options do You have to create (or choose) correct decision?

ADR-302: Main DB for Books
options: MongoDB, PostgreSQL, ElasticSearch, …

ADR-511: Performance testing main toolset
options: JMeter, Gatling, Locust, k6, …

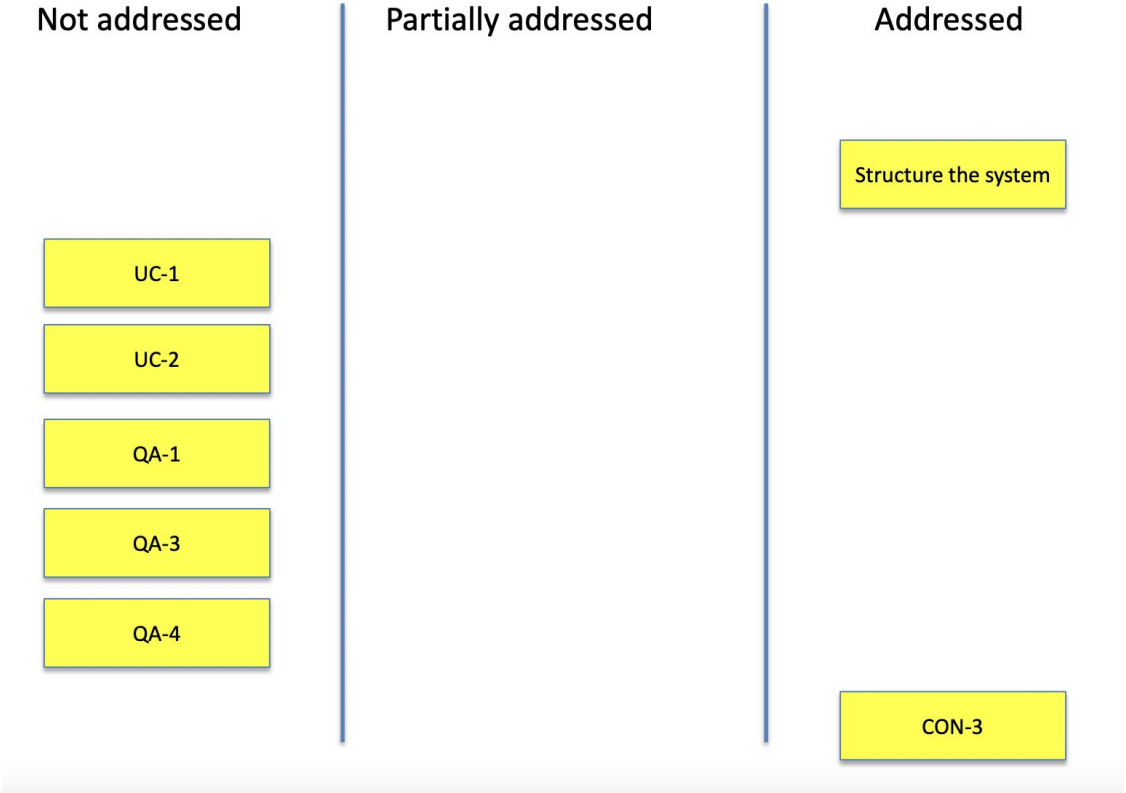
Exemplary list of ADRs with links to ASRs

fwdays

Category	ADRs	ASRs links
General architectural styles & practices	• ADR-101: N-tier architecture • ADR-102: Architectural style for BE services • ADR-103: BE tech stack • ADR-104: Architectural style for FE web-application • ADR-105: FE tech stack • ADR-106: Cross FE-BE communication approach • ADR-107: Event-driven design	• CRN-10, UC-@all • CRN-12, CON-12, RSK-12 • … • … • … • … • …
Infrastructure	• ADR-201: Computing Deployment Model • ADR-202: Cloud Provider • ADR-203: Deployment model for BE services • ADR-204: Computing approach for BE services • ADR-205: Storing and rendering FE web-app • ADR-206: Infrastructure as Code approach • ADR-207: Scaling general approach • ADR-208: Monitoring approach • ADR-209: Logging approach	• CRN-01 • CRN-02, CON-03 • … • … • … • … • … • … • …
Data Management & Integrations	• ADR-301: API Gateway implementation • ADR-302: Main RDBMS • ADR-303: Main NoSQL storage • ADR-304: Cache for applications • ADR-305: Messaging system • ADR-306: Integrations with Payment Gateway	• … • … • … • … • … • …
Platform related	• ADR-401: Backoffice approach • ADR-402: Identity and access management system • ADR-403: SEO approach for public web site • ADR-404: Internationalisation	• … • … • … • …

And You can use ASRs as scope to cover

You can organise ASR covering as tasks (working items) for covering by ADRs in Kanban way.



You can add ADRs links to ASRs

fwdays

ID	Name	Architectural Tactics	Architectural Decisions
QA-01	Interoperability	• Following API Guidelines and best practices • SDLC Guidelines	ADR-001 - N-tier architecture ADR-002 - Arch style for BE (service-based, SOA, etc.) ADR-003 - BE tech stack ADR-006 - Cross FE-BE communication approach ADR-007 - .NET service template
QA-02	Data consistency	• Data decoupling first • Measure and manage Data converge time	ADR-201 - Strangler fig facade for integration ADR-203 - Client's data file storage ADR-204 - Client's data DB ADR-205 - Client and Auditor portals DB
QA-03	Customization	• Managing dependencies • Control system state • Manage deployments	ADR-107 - Monitoring and Logging general approach (incl. tools) ADR-111 - Reference CI/CD process ADR-112 - Branching strategy
QA-04	Usability	• Separate user interface • Support use initiatives (e.g. cancel, undo, etc.) • Support system initiatives (maintain user model and system model)	ADR-004 - Arch style for FE webapp ADR-005 - FE tech stack
QA-05	Collaboration	• Use web-based collaborative platform. • Use real-time system notifications about some events/insides. • Implement security controls, encryption, and access controls to protect sensitive information and ensure compliance with data privacy regulations during collaboration.	ADR-203 - Client's data file storage ADR-204 - Client's data DB ADR-205 - Client and Auditor portals DB
QA-06	Security	• Data encryption at rest and in transit. • Security component between internal and external components. • Security component to control end-user region.	ADR-109 - Secrets management ADR-303 - Admin functionality in Auditor portal ADR-304 - Security auth approach (Azure AD B2C)
QA-07	Reliability	• Cache lifetime management. • Read-Write replicas. • Increase computational efficiency & Reduce computational overhead • Introduce concurrency (load balancing).	ADR-107 - Monitoring and Logging general approach (incl. tools) ADR-108 - Availability general approach
QA-08	Scalability	• Implementation of Dockerisation and orchestration • Scalability Strategy implementation: X-Y-Z, up and down strategies	ADR-106 - Scaling general approach

Then move to Architecture Decision Record (Example)

fwdays

ADR Group: Architectural styles & main technologies

Related ADR: N-tier architecture

ASRs to cover: CRN-12, CRN-15, CON-11, ASM-5

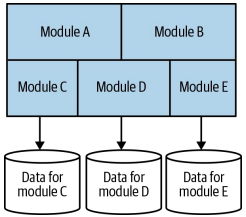
Context and Problem Statement: Architectural style for BE services lies in striking the right balance between modularity, scalability, maintainability, and complexity, as each style comes with its trade-offs and considerations that can significantly impact the overall success and efficiency of the product.

Considered Options:

Option 1 - Modular Monolith

Main attributes:

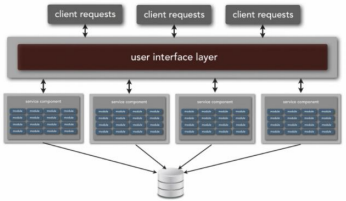
- Single codebase - all functionality are in one application
- Modules encapsulate business logic based on domain
- Tightly Coupled Components
- Single tech stack and Single Build and Deployment
- Scalability via scaling of full app



Option 2 - Service Based Architecture

Main attributes:

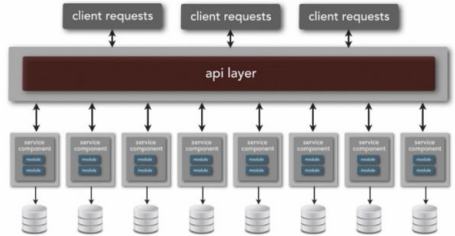
- Quick start because can be Macro-services
- Separate deploy is not required
- DB per service are not required (multiple services can connect to 1 DB)
- Diverse Technology Stacks
- Easy evolutionary approach



Option 3 - Microservices

Main attributes:

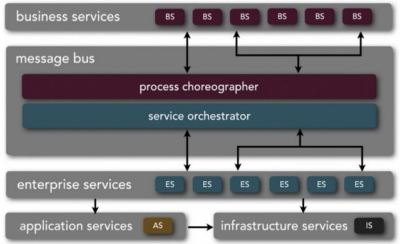
- Minimum functionality in service
- Services are distributed
- DB per service
- Separate deploy and easy scalability
- API Layer
- Diverse Technology Stacks
- Loose Coupling - minimum dependencies between services



Option 4 - Service Oriented Architecture

Main attributes:

- Can be macro and micro services
- Business Process Composition
- Specific types of services
- Diverse Technology Stacks
- Message bus for process choreography and service orchestration



Then move to Architecture Decision Record (Example)

fwdays

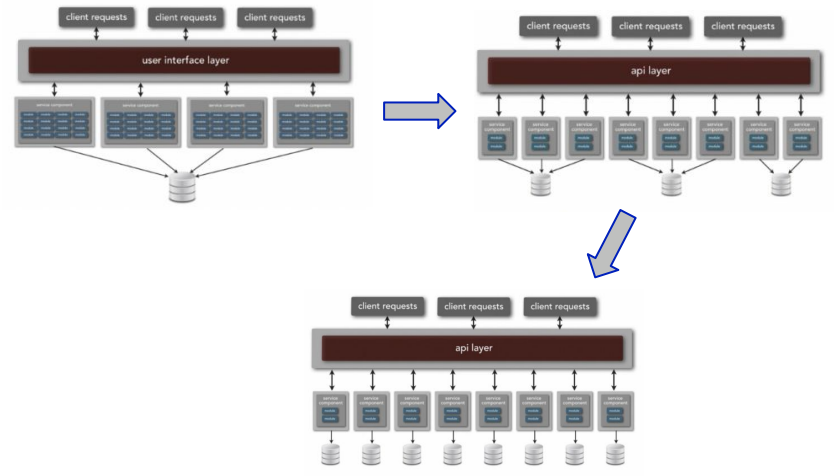
Considered Options - Trade off analysis:

	Modular Monolith	Microservices	Service Based	Service Oriented
agility	**	*****	****	*
configurability	*	**	***	**
cost	*****	*	****	*
deployability	**	*****	****	*
fault-tolerance	*	*****	****	***
integration	**	***	**	*****
performance	***	**	***	**
scalability	*	*****	***	***
testability	**	*****	****	*

Decision:

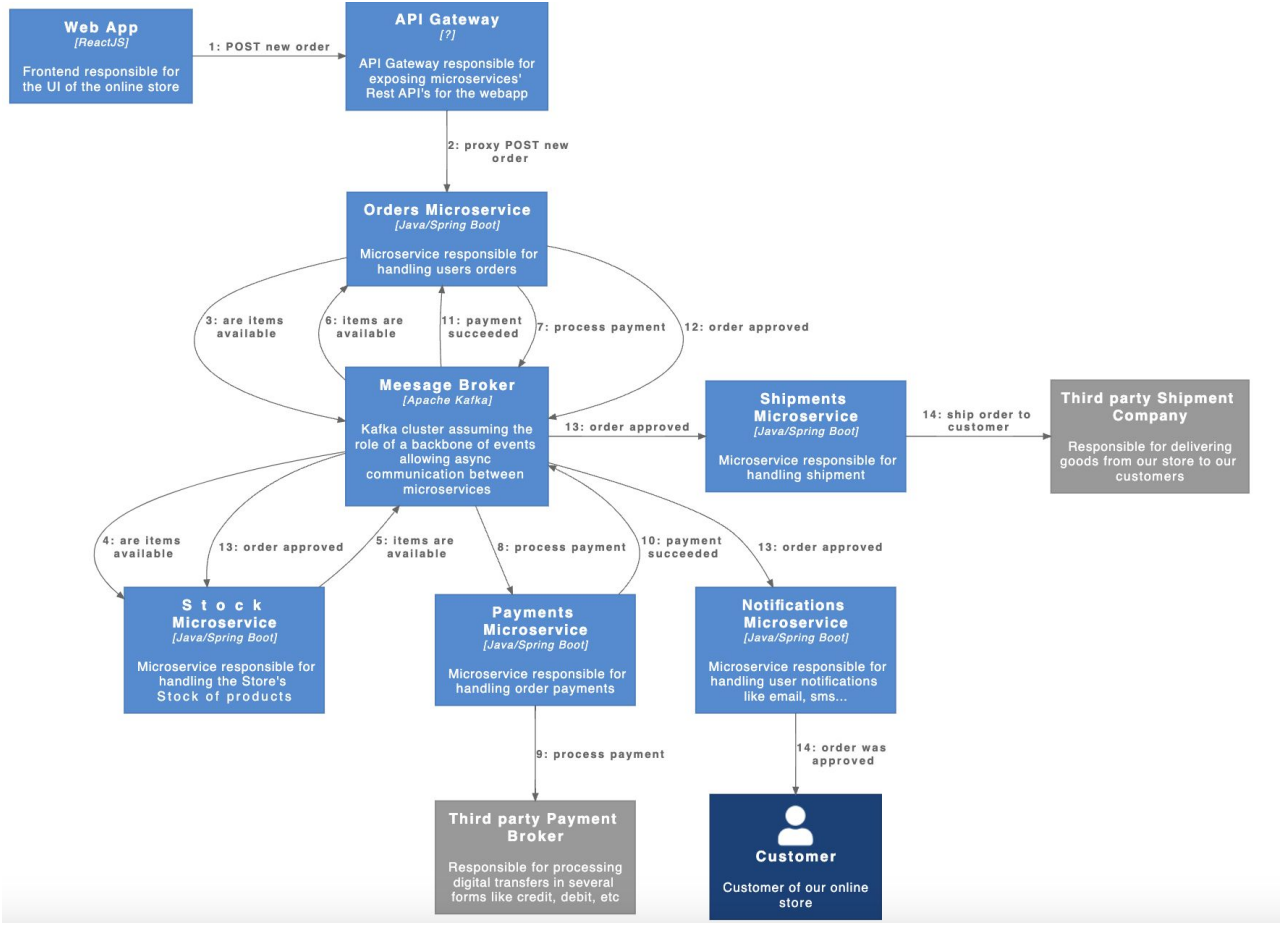
Use **Option 2 “Service Based Architecture” (SBA)** as main architectural style for BE services.

This architectural style will provide speed of development (time-to-market) and possibility to use evolutionary approach and migration to Microservices or SOA if needed.



In parallel we need adjust/add details to Architectural Views

fwdays



Finally we need to have “Architectural Decisions Summary”

fwdays

Category	Summary	ADRs
Infrastructure	Use Public Cloud as computing model with AWS as main cloud provider and N-Tier Architecture decomposition, and instantiate an Infrastructure as Code methodology utilizing AWS CloudFormation.	ADR-101, ADR-201, ADR-202
Back-End components	BE services will be organised as microservices architectural (main tech stack for BE services is Java), deployed as “service instance per container” and computing as docker images (that stored on AWS ECR) on AWS EKS.	ADR-102, ADR-103, ADR-203, ADR-204, ADR-205
Front-End components	SPA architectural style will be used for FE web-app (main tech stack for FE is React). Web-app will be stored as static content on the AWS S3 storage and rendered by AWS S3.	ADR-104, ADR-105, ADR-206
Components communication	Platform will support cross FE-BE communication via API gateway and using HTTP/REST with Swagger API documentation. Back-End services will communication on synchronic way (direct calls via HTTP/REST) and async way using messaging system (AWS SQS).	ADR-105, ADR-305
Security and Auth	Auth and SSO will be implemented as dedicated component with using cloud service AWS Cognito.	ADR-401
Storages	Use PostgreSQL as a RDBMS, DynamoDB will be used as main NoSQL storage, use Redis for cache (Amazon ElastiCache for Redis).	ADR-301, ADR-302, ADR-303
Integrations	Platform components will be integrated with APIs of 3rd party systems (e.g. Email providers, Payment Gateways) via HTTPS/REST using official sdk/libraries.	ADR-304
Testing and Quality	Testing strategy will cover adoption of the Test pyramid to cover different levels of testing (e.g. Unit, Component, Integration, System, Performance) with defined testing metrics.	ADR-106, ADR-107
Performance, Scaling and Availability	Scaling approach based on X-Y-Z scaling cube principle via Cluster Autoscaler for AWS EKS. Availability approach based on AWS Business Continuity and Disaster Recovery best practices (inc. failover cluster, backups, quick recovery, etc.), and focus on pillars of the AWS Well Architected Framework.	ADR-205, ADR-207, ADR-208, ADR-209
Monitoring and Logging	Main tool for implementation Monitoring and Logging will be AWS CloudWatch.	ADR-210, ADR-211

fwdays

Main design concepts to use and their catalogues

Oleksandr Savchenko

Main Design Concepts to use

fwdays

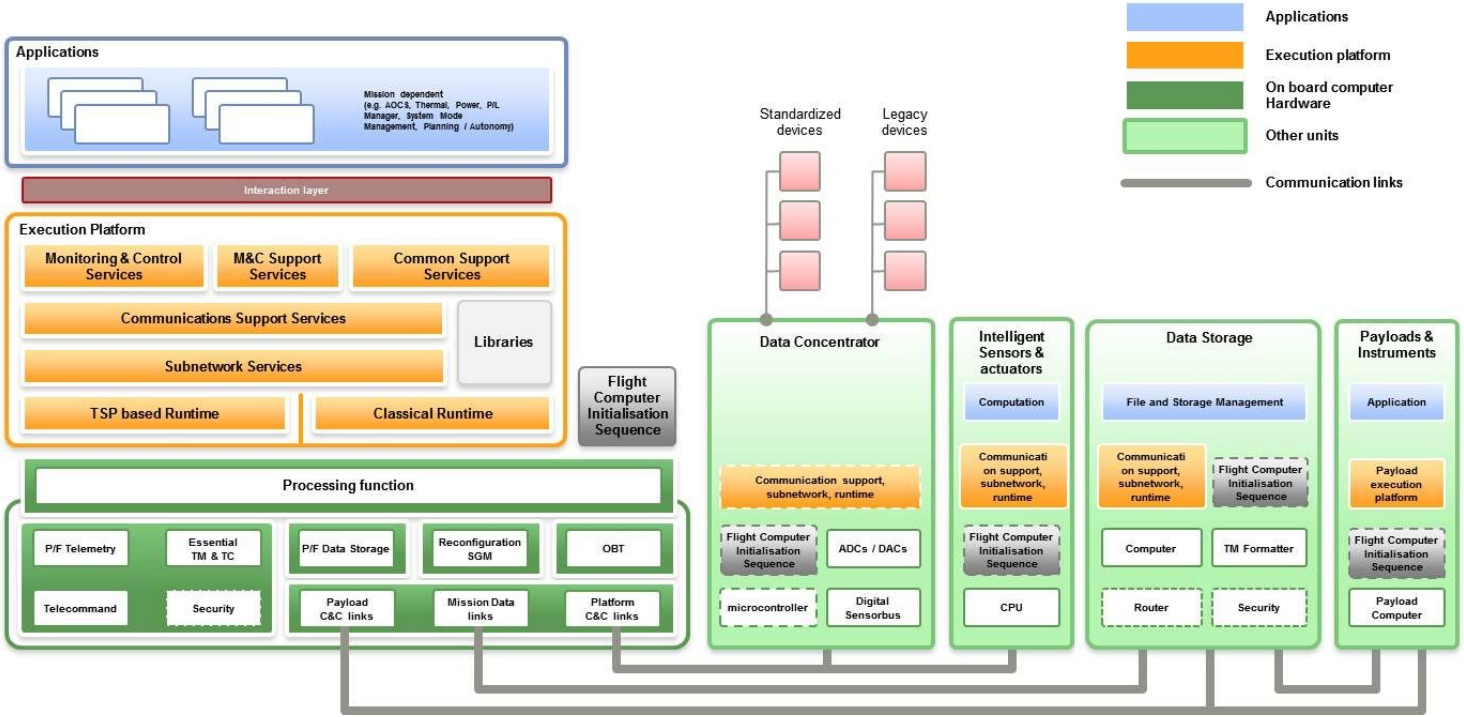
DESIGN CONCEPTS

- Reference Solution Architectures
- Architectural Styles and Patterns
- Deployment and Computing Patterns
- Reference Application Architectures
- Architectural Tactics
- Concrete Technologies
- Developed components (e.g. Frameworks, packages/modules)
- Standards (e.g. RFC, ISO)

Category	ADRs
General architectural styles & practices	• ADR-101: N-tier architecture • ADR-102: Architectural style for BE services • ADR-103: BE tech stack • ADR-104: Architectural style for FE web-application • ADR-105: FE tech stack • ADR-106: Cross FE-BE communication approach • ADR-107: Event-driven design
Infrastructure	• ADR-201: Computing Deployment Model • ADR-202: Cloud Provider • ADR-203: Deployment model for BE services • ADR-204: Computing approach for BE services • ADR-205: Storing and rendering FE web-app • ADR-206: Infrastructure as Code approach • ADR-207: Scaling general approach • ADR-208: Monitoring approach • ADR-209: Logging approach
Data Management & Integrations	• ADR-301: API Gateway implementation • ADR-302: Main RDBMS • ADR-303: Main NoSQL storage • ADR-304: Cache for applications • ADR-305: Messaging system • ADR-306: Integrations with Payment Gateway
Platform related	• ADR-401: Backoffice approach • ADR-402: Identity and access management system • ADR-403: SEO approach for public web site • ADR-404: Internationalisation

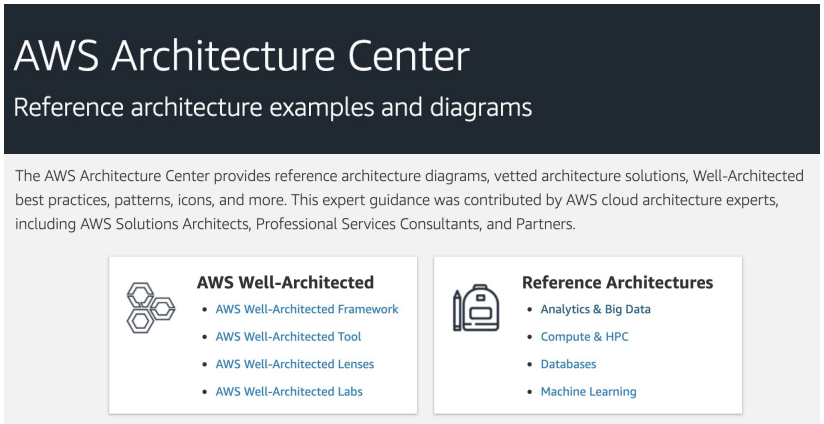
1. Reference Solution Architectures

Reference architectures are blueprints that provide an overall logical structure for particular types of applications.

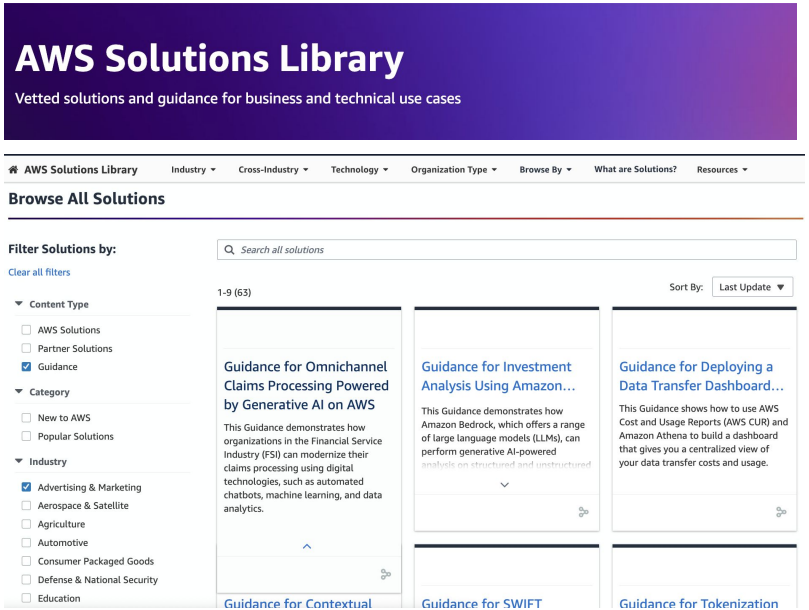


SAVOIR (Space Avionics Open Interface Architecture)

1. Reference Solution Architectures: AWS



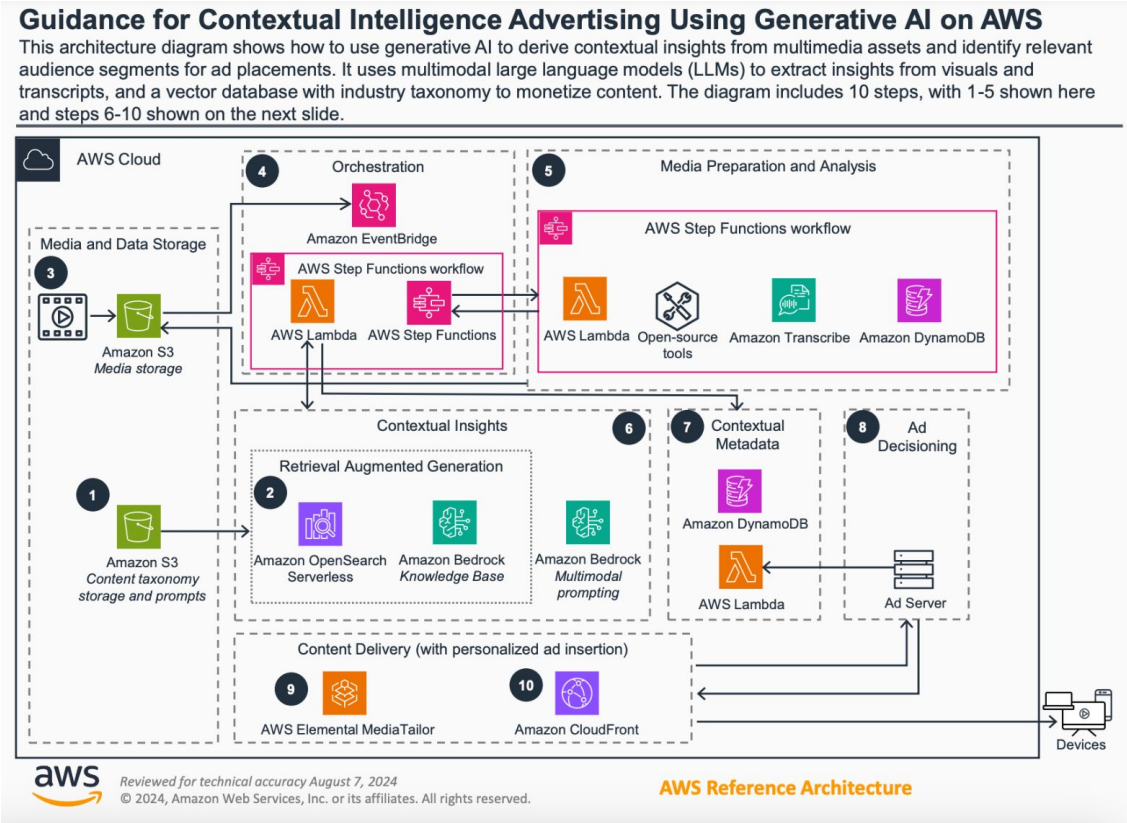
https://aws.amazon.com/architecture/



https://aws.amazon.com/solutions

1. Reference Solution Architectures: AWS

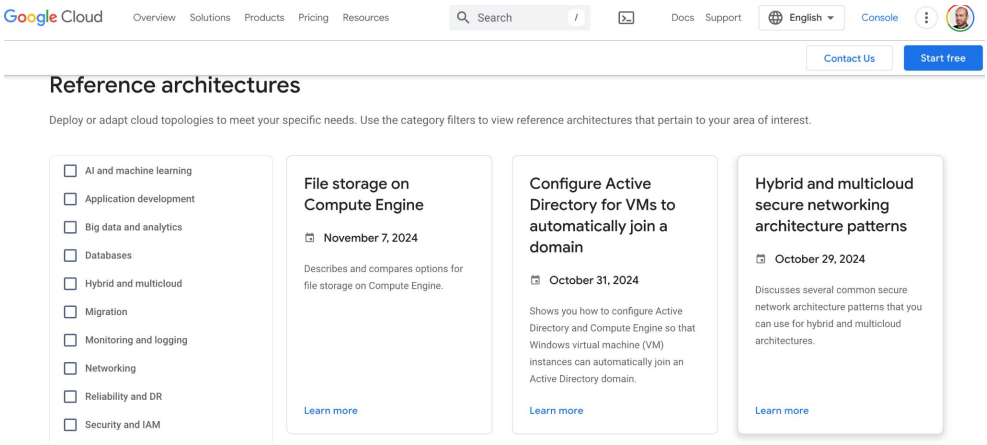
fwdays



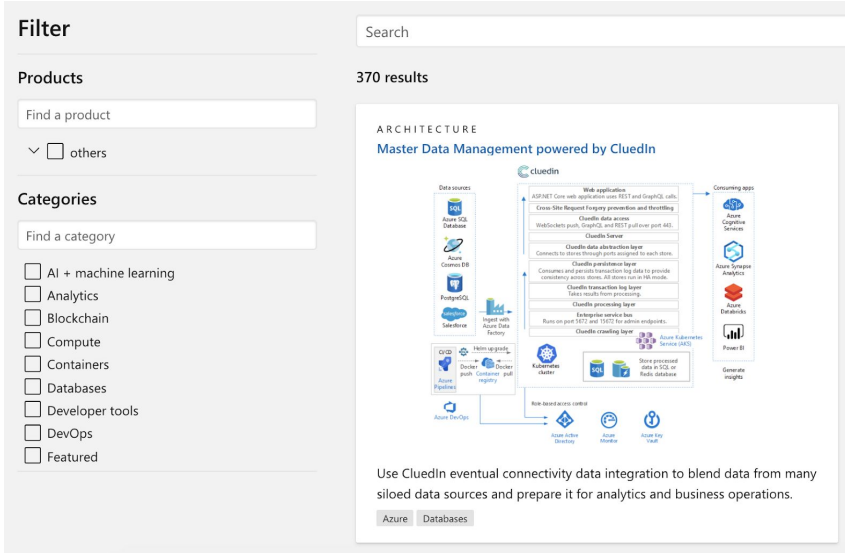
1. Upload the latest version of the IAB Content Taxonomy to **Amazon Simple Storage Service (Amazon S3)**.
2. Create a knowledge base in **Amazon Bedrock**, a fully managed service that offers a choice of high-performing foundation models (FMs). Select **Amazon OpenSearch Serverless** as the vector database and **Amazon Titan Text Embeddings v2 model** as the embeddings model.
3. Upload media content to an **Amazon S3** bucket.
4. **Amazon EventBridge** receives object creation notifications from **Amazon S3**. This triggers an orchestration workflow that executes **AWS Step Functions** for media preparation and analysis, as well as **AWS Lambda** functions to invoke **Amazon Bedrock** for contextual insights extraction.
5. The **Step Functions** workflow executes media preparation and analysis tasks, invoking **Lambda** functions that use open-source tools like ffmpeg and perceptual hashing, generating transcriptions using **Amazon Transcribe**. The workflow metadata is stored in **Amazon DynamoDB**, and the processed media files are persisted in **Amazon S3**.

https://aws.amazon.com/solutions/guidance/contextual-intelligence-advertising-using-generative-ai-on-aws/

1. Reference Solution Architectures: GCP and Azure

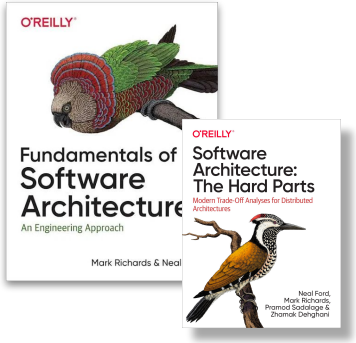


https://cloud.google.com/architecture/all-reference-architectures

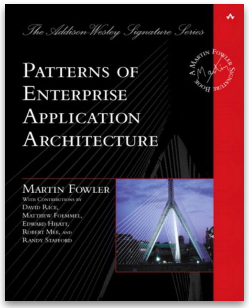


https://docs.microsoft.com/en-us/azure/architecture/browse/

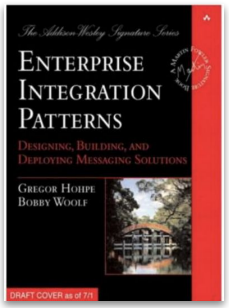
2. Architectural Styles and Patterns



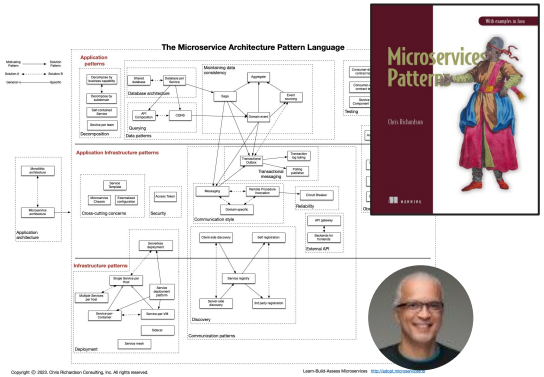
https://docs.microsoft.com/en-us/azure/architecture/patterns/



https://martinfowler.com/eaaCatalog/



https://www.enterpriseintegrationpatterns.com/patterns/messaging/



https://microservices.io/

https://patterns.arcitura.com/

Hint: Architectural Styles and Patterns Azure Architecture Center

fwdays

Design and Implementation

Pattern	Summary
Ambassador	Create helper services that send network requests on behalf of a consumer service or application.
Anti-Corruption Layer	Implement a façade or adapter layer between a modern application and a legacy system.
Backends for Frontends	Create separate backend services to be consumed by specific frontend applications or interfaces.
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.
Compute Resource Consolidation	Consolidate multiple tasks or operations into a single computational unit
Edge Workload Configuration	The great variety of systems and devices on the shop floor can make workload configuration a difficult problem.
External Configuration Store	Move configuration information out of the application deployment package to a centralized location.
Gateway Aggregation	Use a gateway to aggregate multiple individual requests into a single request.
Gateway Offloading	Offload shared or specialized service functionality to a gateway proxy.
Gateway Routing	Route requests to multiple services using a single endpoint.
Leader Election	Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the other instances.
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.
Sidecar	Deploy components of an application into a separate process or container to provide isolation and encapsulation.
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.
Strangler Fig	Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

Messaging

Pattern	Summary
Asynchronous Request-Reply	Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.
Claim Check	Split a large message into a claim check and a payload to avoid overwhelming a message bus.
Choreography	Have each component of the system participate in the decision-making process about the workflow of a business transaction, instead of relying on a central point of control.
Competing Consumers	Enable multiple concurrent consumers to process messages received on the same messaging channel.
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.
Priority Queue	Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.
Publisher-Subscriber	Enable an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers.
Queue-Based Load Leveling	Use a queue that acts as a buffer between a task and a service that it invokes in order to smooth intermittent heavy loads.
Saga	Manage data consistency across microservices in distributed transaction scenarios. A saga is a sequence of transactions that updates each service and publishes a message or event to trigger the next transaction step.
Scheduler Agent Supervisor	Coordinate a set of actions across a distributed set of services and other remote resources.
Sequential Convoy	Process a set of related messages in a defined order, without blocking processing of other groups of messages.

Data Management

Pattern	Summary
Cache-Aside	Load data on demand into a cache from a data store
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.
Event Sourcing	Use an append-only store to record the full series of events that describe actions taken on data in a domain.
Index Table	Create indexes over the fields in data stores that are frequently referenced by queries.
Materialized View	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for required query operations.
Sharding	Divide a data store into a set of horizontal partitions or shards.
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.
Valet Key	Use a token or key that provides clients with restricted direct access to a specific resource or service.

Azure Architecture Center

Guidance for architecting solutions on Azure using established patterns and practices

https://docs.microsoft.com/en-us/azure/architecture/patterns/

Hint: Architectural Styles and Patterns
Microservices.io by Chris Richardson

Chris Richardson

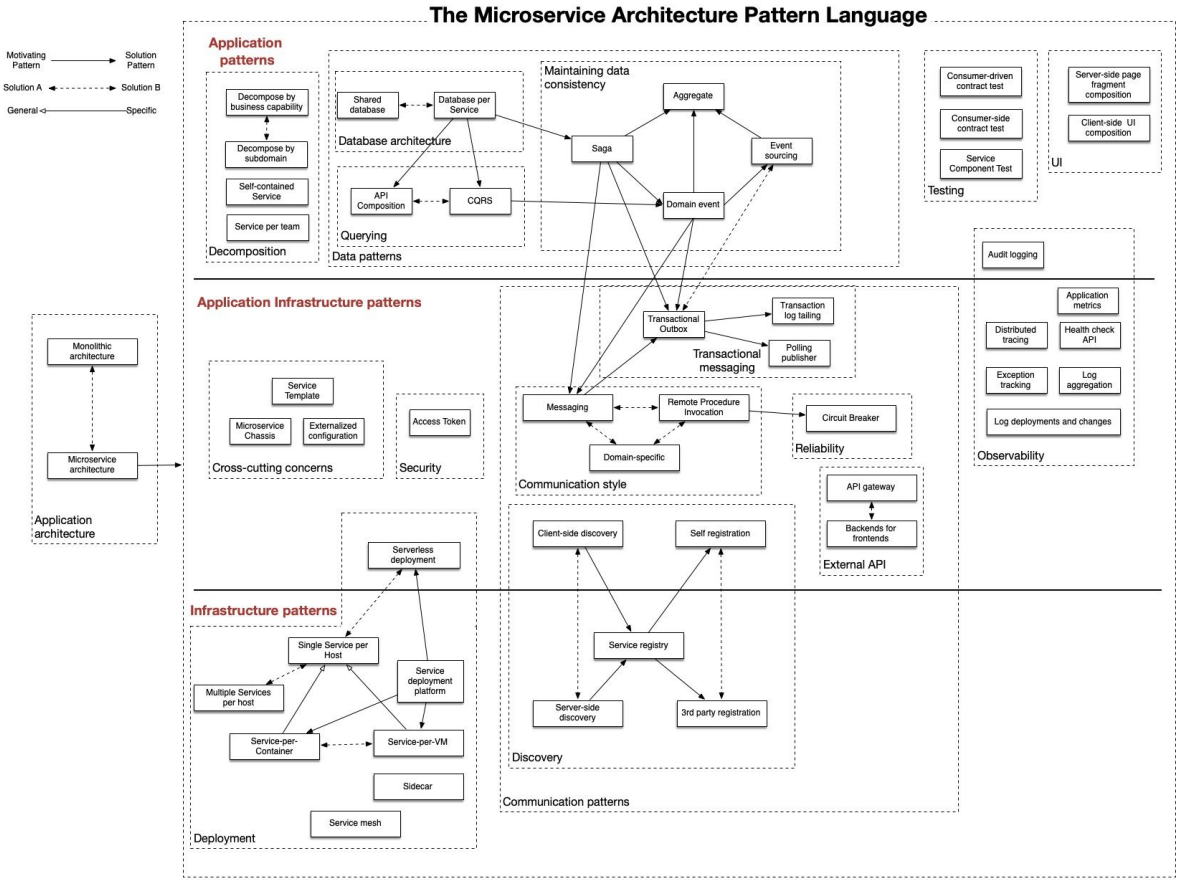
is an experienced software architect and entrepreneur

https://www.chrisrichardson.net/

https://www.linkedin.com/in/pojos/

https://microservices.io

- **Application patterns**
  - Decomposition
  - Data patterns
    - DB architecture
    - Querying
    - Maintaining data consistency
  - Testing
  - UI
- **Application Infrastructure patterns**
  - Cross-Cutting
  - Security
  - Communication style
  - External API
  - Transactional messaging
  - Observability
- **Infrastructure patterns**
  - Deployment
  - Discovery



Hint: Architectural Styles and Patterns Patterns by Arcitura

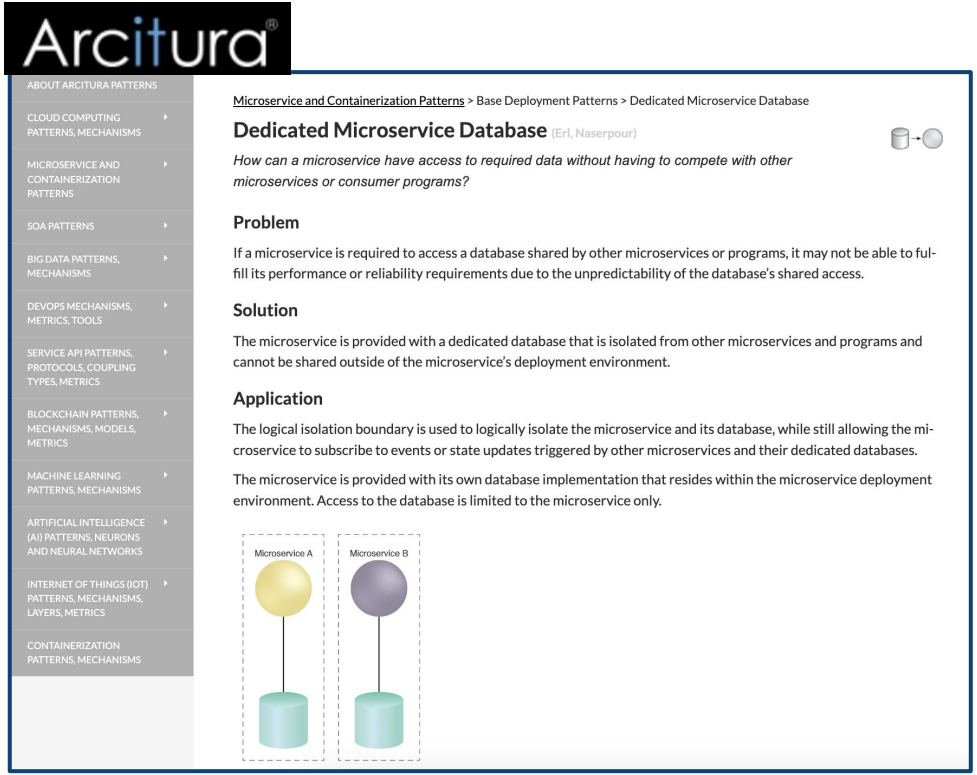
fwdays

Patterns categories:

1. Cloud Computing patterns, mechanisms
2. Microservices and containerization patterns
3. SOA patterns
4. Big Data patterns, mechanisms
5. DevOps mechanisms, metrics, tools
6. Service API patterns, protocols, coupling types, metrics
7. Blockchain patterns, mechanisms, models, metrics
8. Machine learning patterns, mechanisms
9. Artificial Intelligence (AI) patterns, Neurons and Neural Networks
10. Internet Of Things (IOT) patterns, mechanisms, layers, metrics
11. Containerization patterns, mechanisms

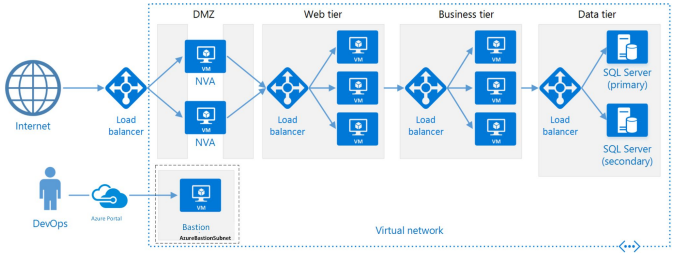
Pattern Profiles:

- Requirement
- Problem
- Solution
- Application
- Mechanisms

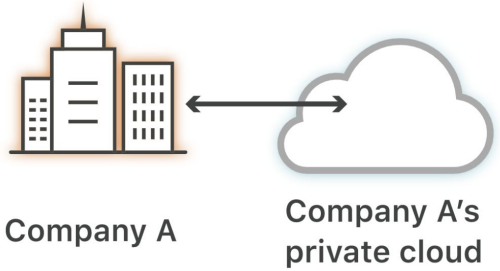


https://patterns.arcitura.com/

3. Deployment and Computing Patterns

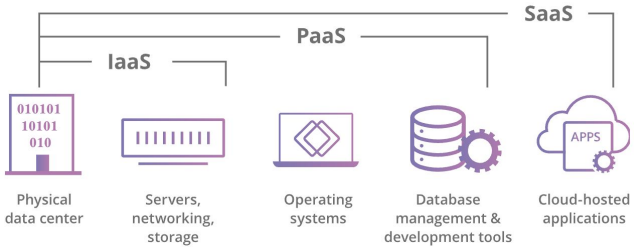


N-Tier deployment

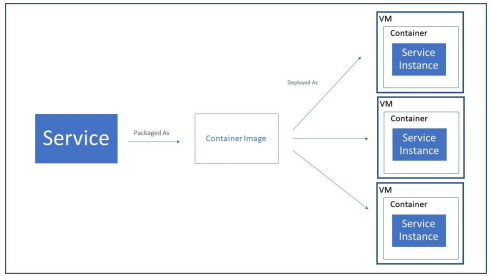


On-premise
Private Cloud
Public Cloud
Hybrid Cloud
Multi-cloud
Community Cloud

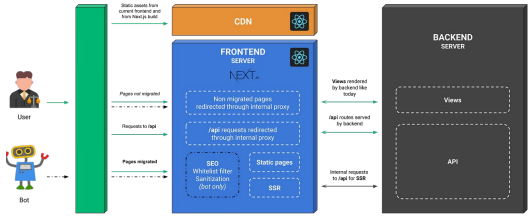
Computing Deployment Models



Cloud computing service models

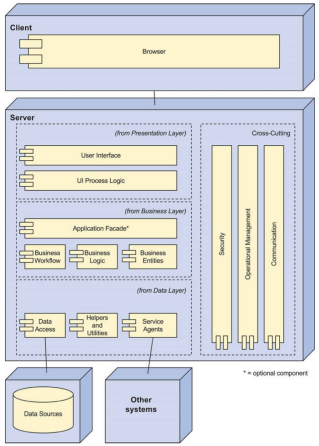


Deployment patterns for BE services

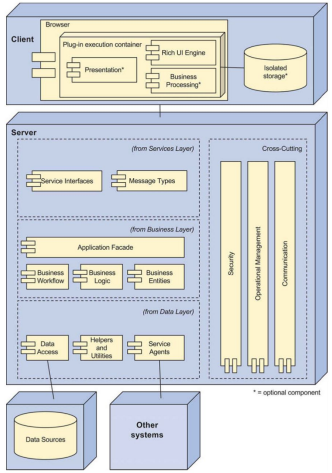


Deployment patterns for FE web-application

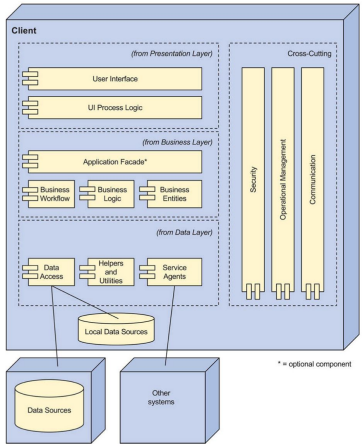
4. Reference Application Architectures



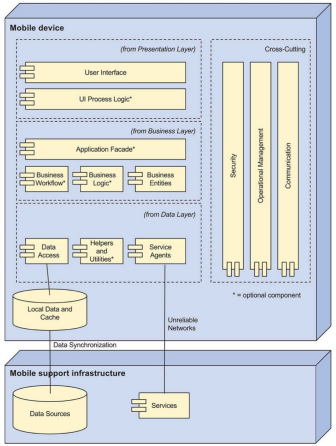
1. Web Application



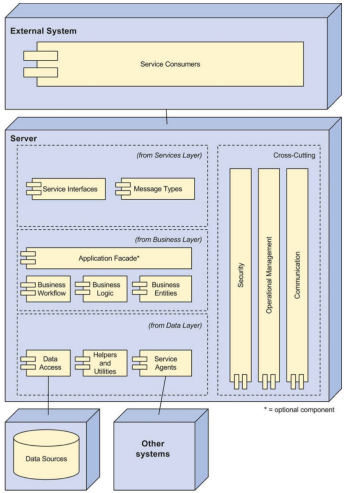
2. Rich-Internet Application



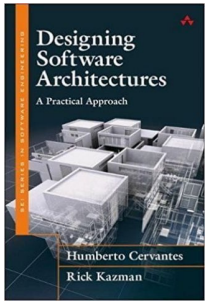
3. Rich Client Application



4.Mobile Client Application



5. Services

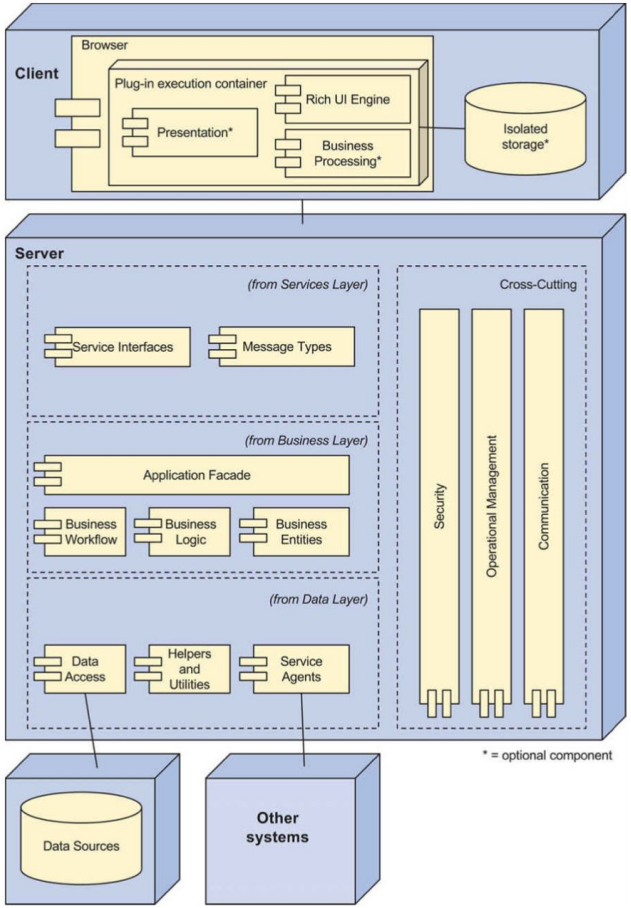


https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658107(v=pandp.10)

4. Reference Application Architectures

fwdays

Rich Internet Application

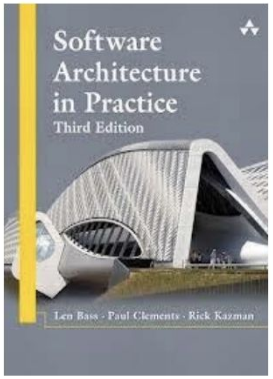


Component Name	Responsibility
Presentation	Responsible for managing user interaction (represents both UI components and UI process logic components).
Rich UI engine	Responsible for rendering user interface elements inside the plug-in execution container.
Business processing	Responsible for managing business logic on the client side.
Service interfaces	Responsible for exposing services that are consumed by the components that run on the browser.
Message types	Responsible for managing the types of messages that are exchanged between the client part and the server part of the application.

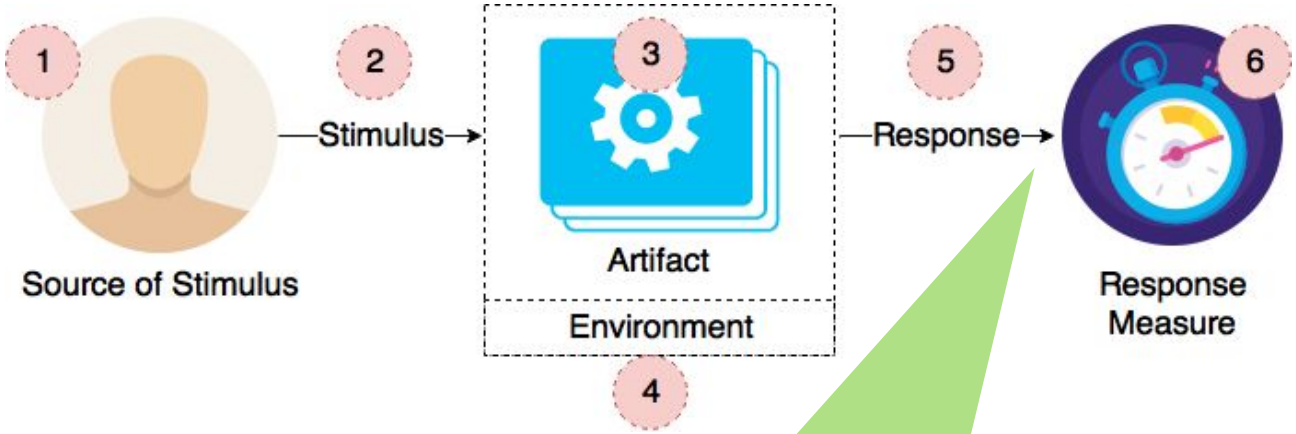
https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658107(v=pandp.10)

5. Architectural Tactics

fwdays



1. Availability
2. Interoperability
3. Modifiability
4. Performance
5. Security
6. Testability
7. Usability

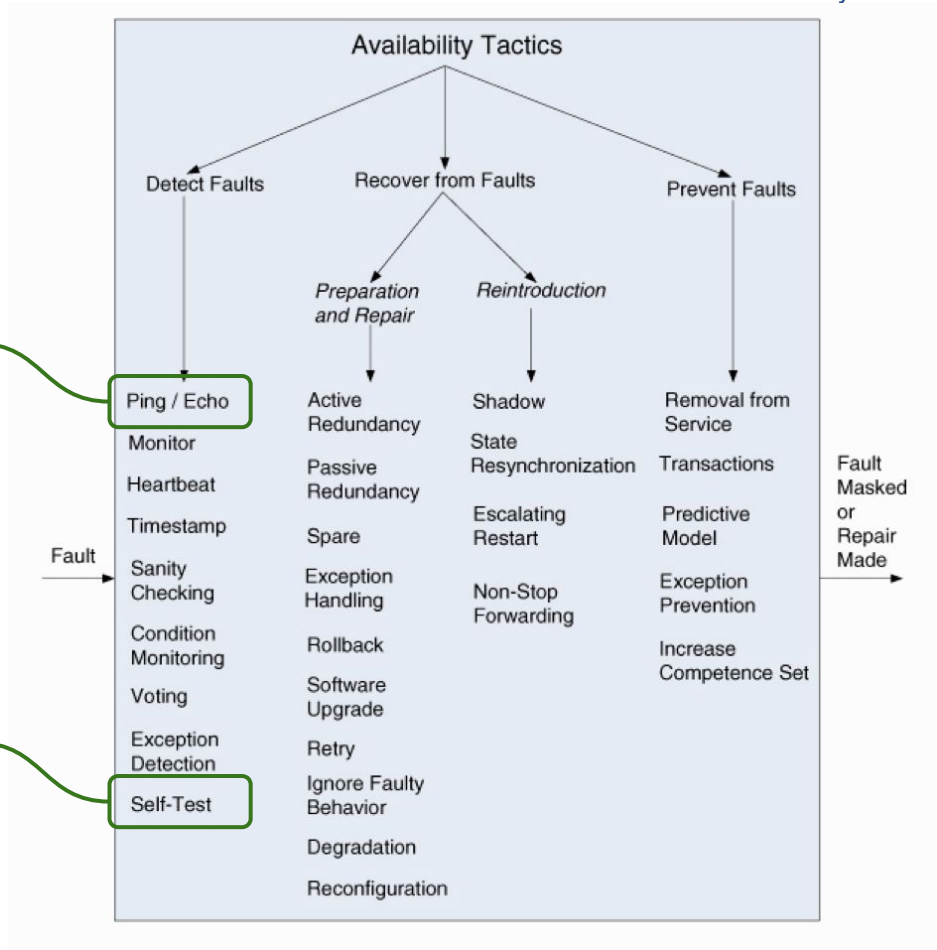


Architectural Tactics - formal design decisions that influence the control of a Quality Attribute Scenario Response.

5. Architectural Tactics

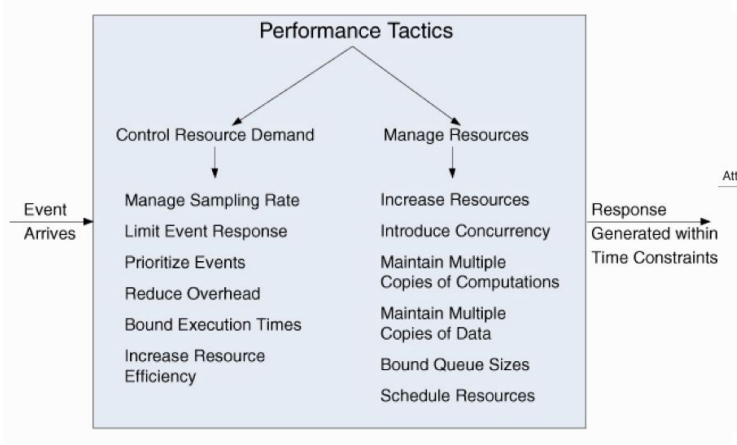
Ping/echo refers to an asynchronous request/response message pair exchanged between nodes, used to determine reachability and the round-trip delay through the associated network path. But the **echo** also determines that the pinged component is alive and responding correctly. The **ping** is often sent by a system monitor. **Ping/echo** requires a time threshold to be set; this threshold tells the pinging component how long to wait for the echo before considering the pinged component to have failed ("timed out"). Standard implementations of ping/echo are available for nodes interconnected via IP.

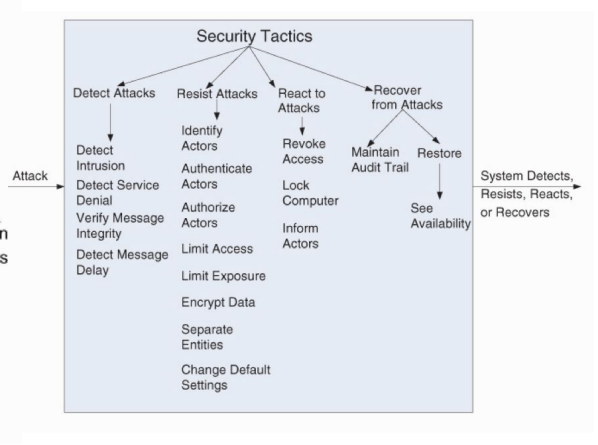
Self-test. Components (or, more likely, whole subsystems) can run procedures to test themselves for correct operation. **Self-test** procedures can be initiated by the component itself, or invoked from time to time by a system monitor. These may involve employing some of the techniques found in condition monitoring, such as checksums.

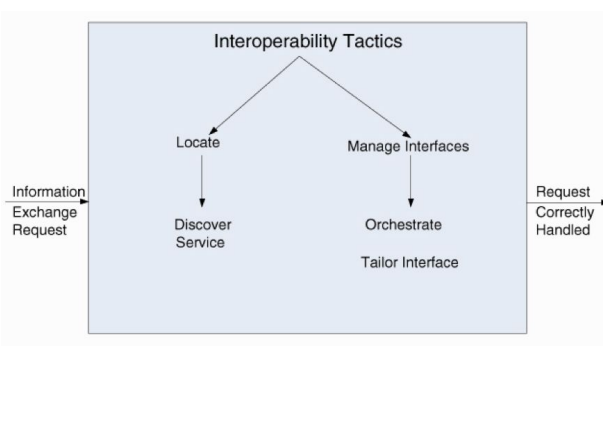


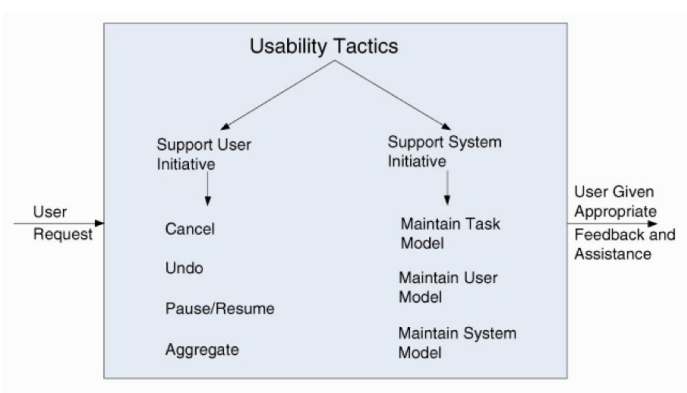
5. Architectural Tactics

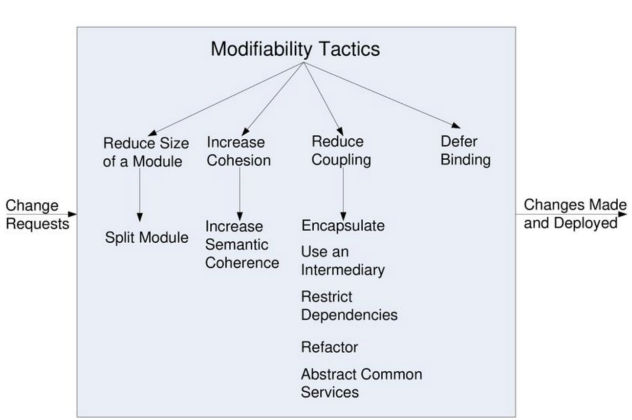
fwdays

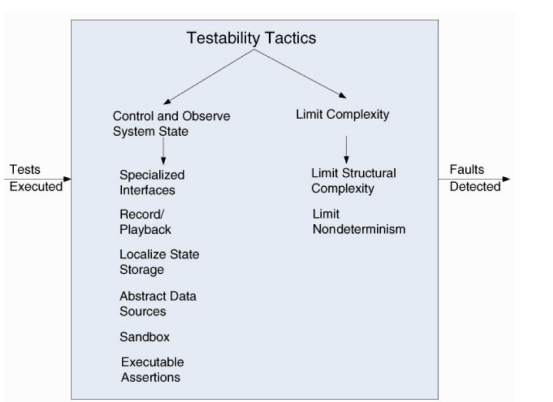












fwdays

General Architectural Decisions

Oleksandr
Savchenko

Exemplary list of decisions

fwdays

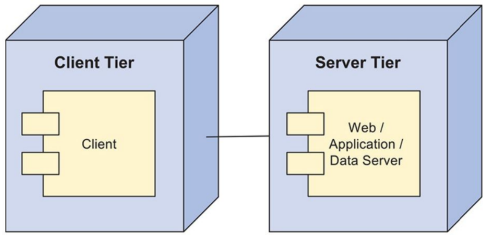
Category	ADRs
General architectural styles & practices	• **N-tier architecture** • **Computing Deployment Model** • **Cloud Computing service model** • **Architectural style for BE services** • **Architectural style for FE web-application** • **…**
Infrastructure & DevOps	• Deployment model and computing approach for BE and FE • Infrastructure as Code approach • Scaling approach • Availability approach • CI/CD approach • Observability (Monitoring, Logging, Tracing) approach • …
Components interactions and applications	• Tech stack for BE, FE, … • Cross FE-BE communication approach • API Gateway implementation • Cross services communication approach • Application structures • …
Data Management, AI and Integrations	• Databases (RDBMS, NoSQL) • Cache for applications • Data ingestion approaches • Integrations with external system • …
Platform related	• Backoffice approach • Identity and access management system • Internationalisation • SEO approach • …

N-Tier Architecture

[2-3-4] - Tier Deployment

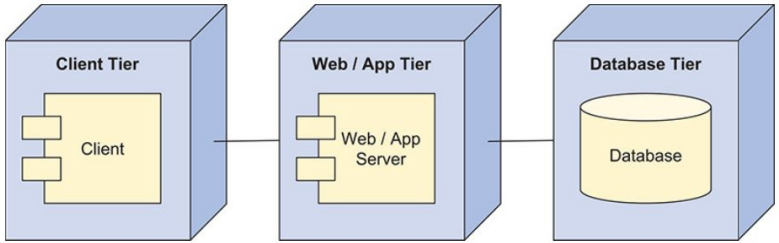
Important General decision to start with

fwdays

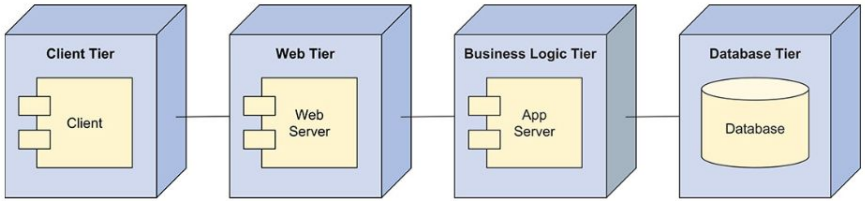


2-Tier Deployment

N-tier deployment - a pattern that provides a model for how to physically structure the system to deploy it.



3-Tier Deployment

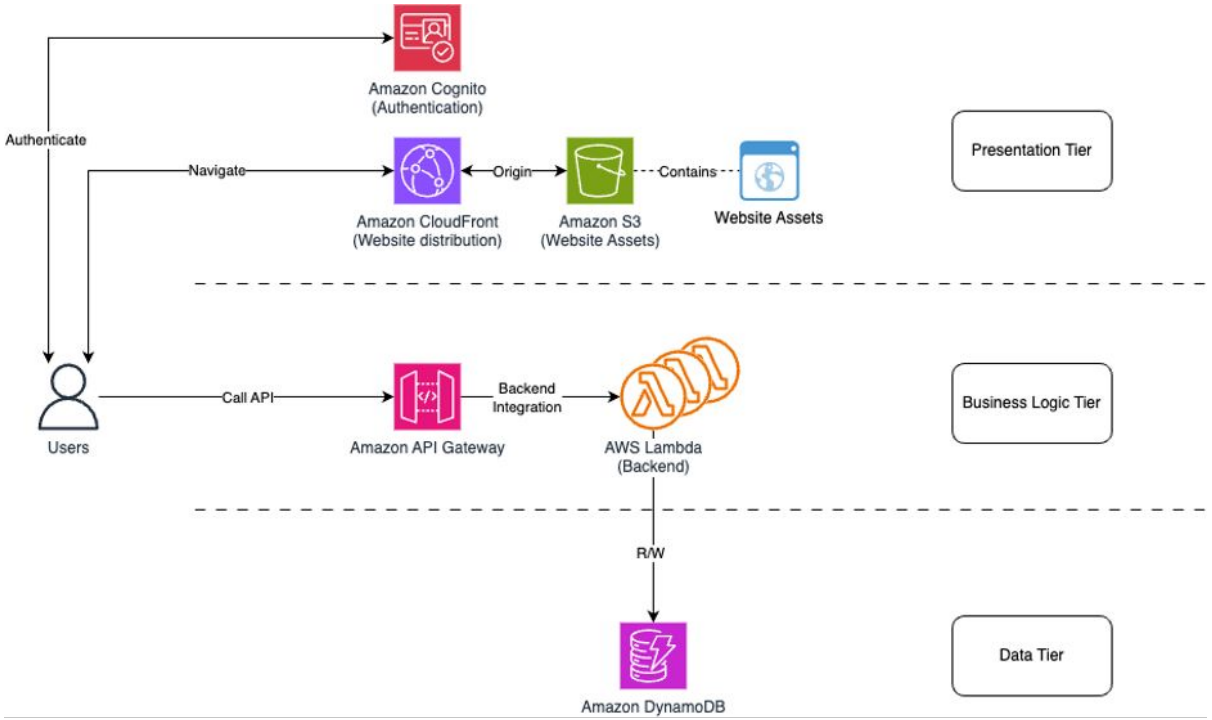


4-Tier Deployment

It can be used for identification of logical tiers too...

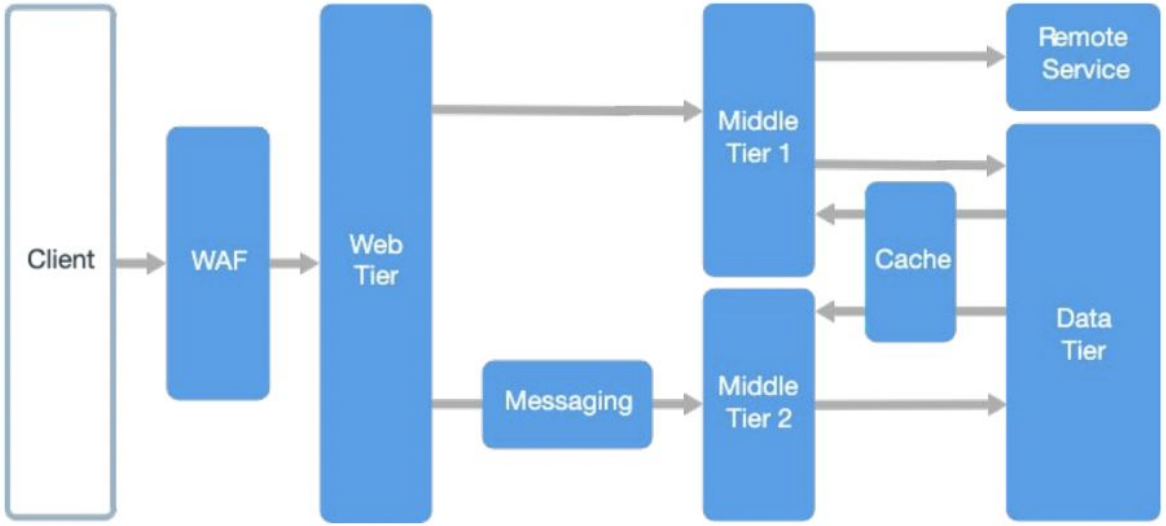
fwdays

Main principle is to understand how requests from the end-user will be handled and response will be formed, as result to divide the system into logical layers and physical tiers.



https://aws.amazon.com/blogs/architecture/building-a-three-tier-architecture-on-a-budget/

N-Tier Architecture



https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

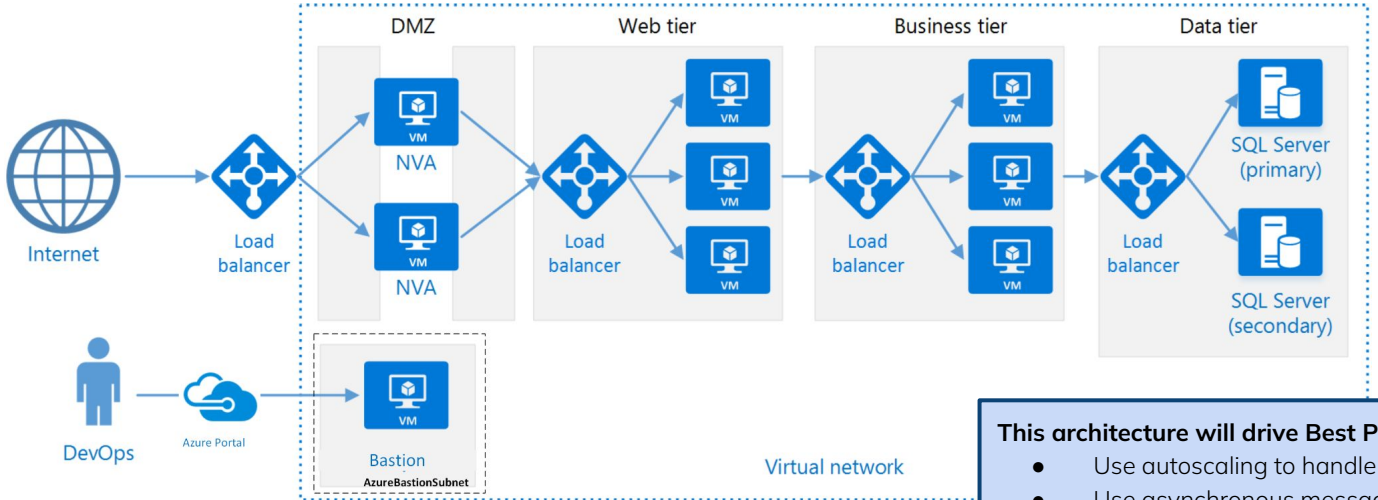
Main Benefits:

- **Each tier can run separately** on a separate operating system and server platform, and each tier runs on at least one dedicated hardware or virtual server, so the services of each tier can be customized and optimized without impact the other tiers.
- **Faster development** and less learning curve for most developers.
- **Natural evolution** from the traditional application model.
- Improved and managed **scalability, reliability and security !!!**

N-Tier Architecture supports in identification

fwdays

Example of N-tier architecture on VMs:



This architecture will drive Best Practices for implementation:

- Use autoscaling to handle changes in load.
- Use asynchronous messaging to decouple tiers.
- Cache semi-static data and configure the database tier for high availability, restrict access to the data tier, by allowing requests only from the middle tier(s).
- Place a web application firewall (WAF) between the front end and the Internet.
- Place each tier in its own subnet, and use subnets as a security boundary.

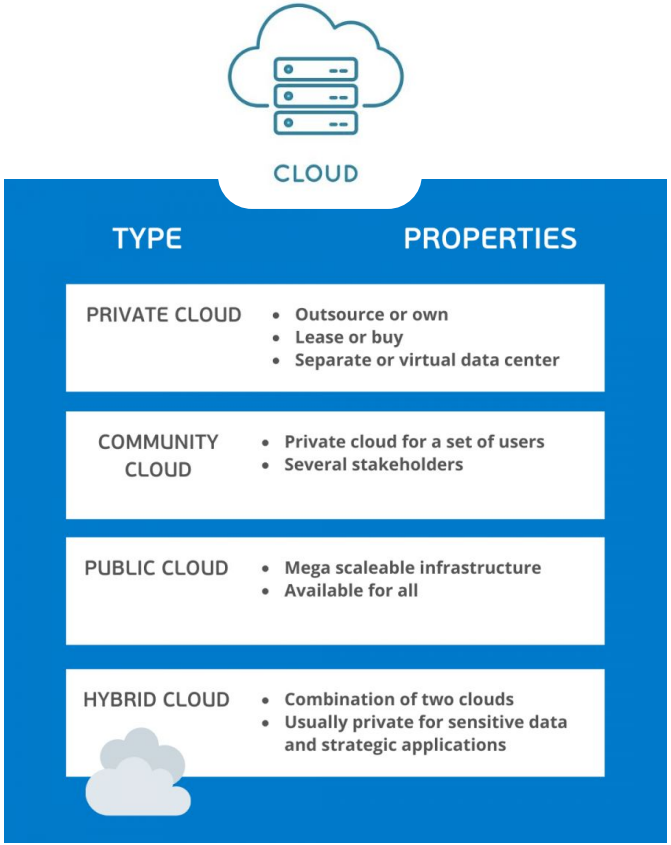
https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier#n-tier-architecture-on-virtual-machines

Computing Deployment and Service Models

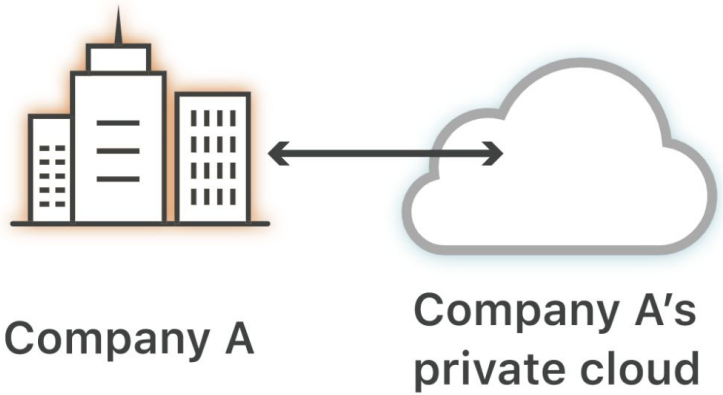
Computing Deployment Model

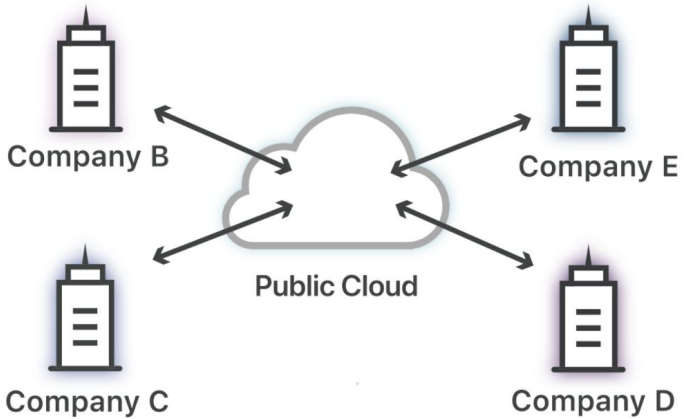
fwdays

On-Premise



Computing Deployment Model





Private cloud

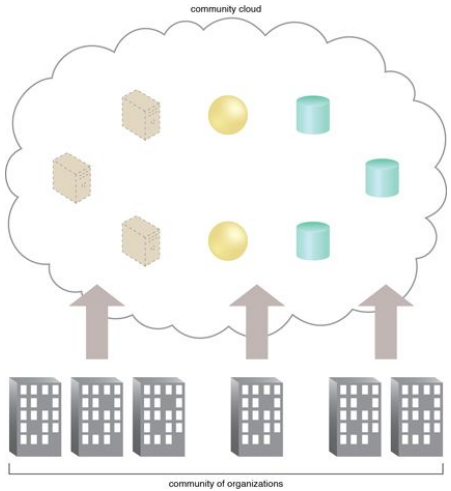
A private cloud is a server, data center, or distributed network wholly dedicated to one organization.

Public cloud

A public cloud is a service run by an external vendor that may include servers in one or multiple data centers. Unlike a private cloud, public clouds are shared by multiple organizations. Using virtual machines, individual servers may be shared by different companies, a situation that is called "multi tenancy" because multiple tenants are renting server space within the same server.

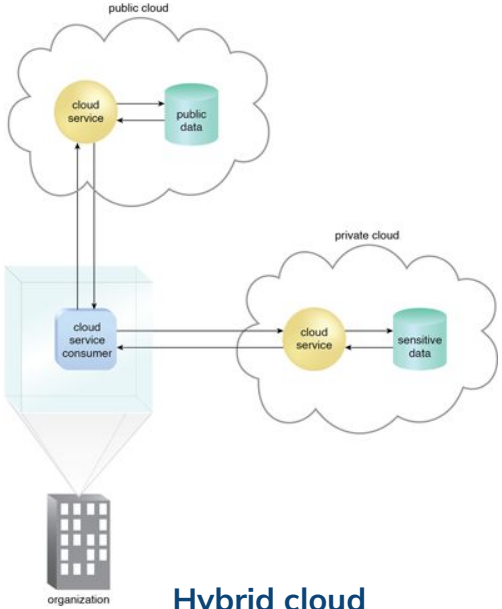
https://patterns.arcitura.com/cloud-computing-patterns/basics/cloud-deployment-models/overview

Computing Deployment Model



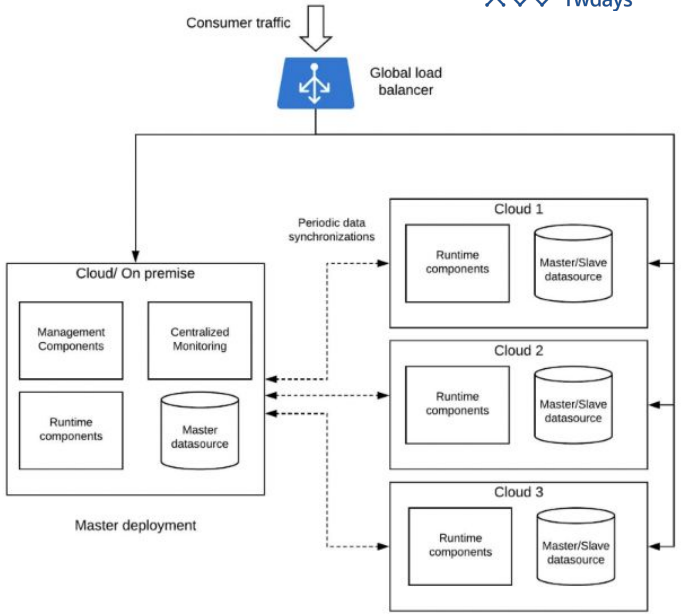
Community cloud

A public cloud is a service run by an external vendor that may include servers in one or multiple data centers. Unlike a private cloud, public clouds are shared by multiple organizations. Using virtual machines, individual servers may be shared by different companies, a situation that is called "multitenancy" because multiple tenants are renting server space within the same server.



Hybrid cloud

A hybrid cloud is a cloud environment comprised of two or more different cloud deployment models. For example, a cloud consumer may choose to deploy cloud services processing sensitive data to a private cloud and other, less sensitive cloud services to a public cloud.

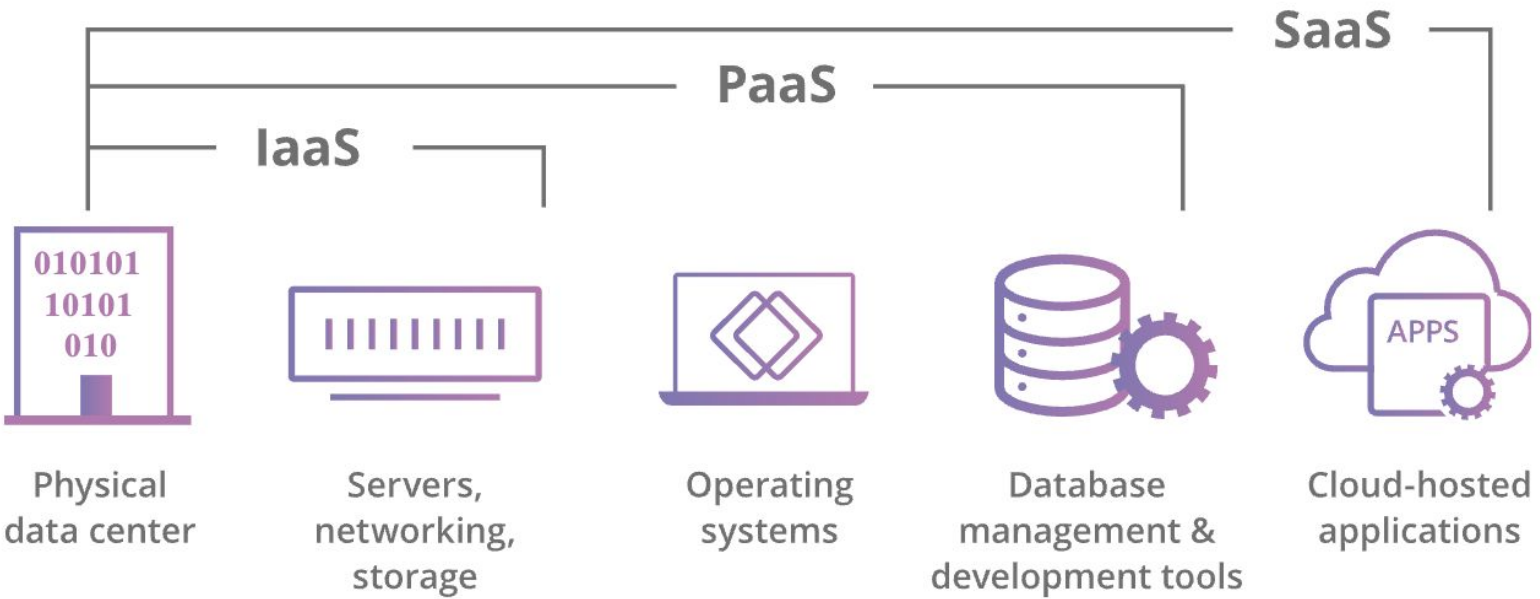


Multi cloud

An organization with a multi-cloud deployment rents virtual servers and services from several external vendors (different Public Cloud) — to continue the analogy used above, this is like leasing several adjacent plots of land from different landlords. Multi-cloud deployments can also be hybrid cloud, and vice versa.

https://patterns.arcitura.com/cloud-computing-patterns/basics/cloud-deployment-models/overview

Cloud Computing service models

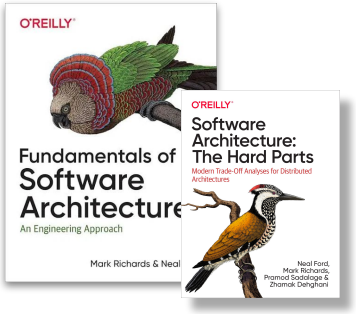


https://www.cloudflare.com/en-gb/learning/cloud/what-is-the-cloud/

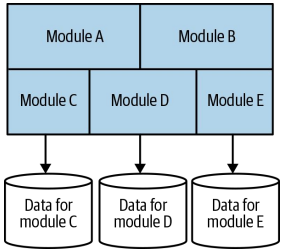
Architectural Styles for Back-End

Architectural Styles for Back-End

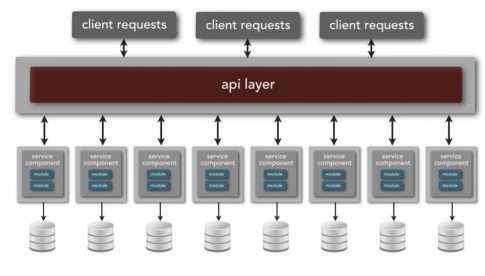
fwdays



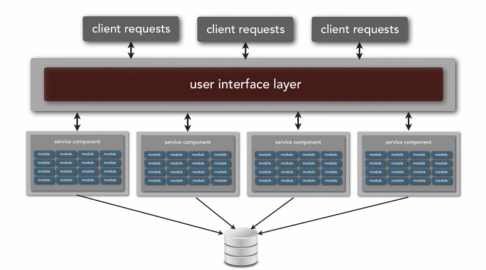
1. Modular Monolith



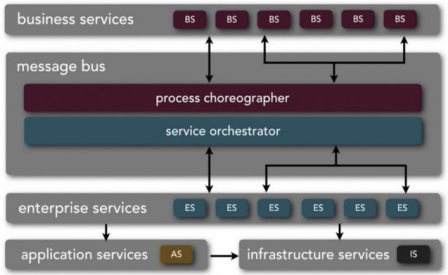
2. Microservices



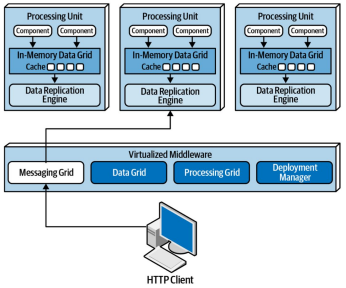
3. Service-Based architecture



4. Service-Oriented architecture



5. Space-Based architecture

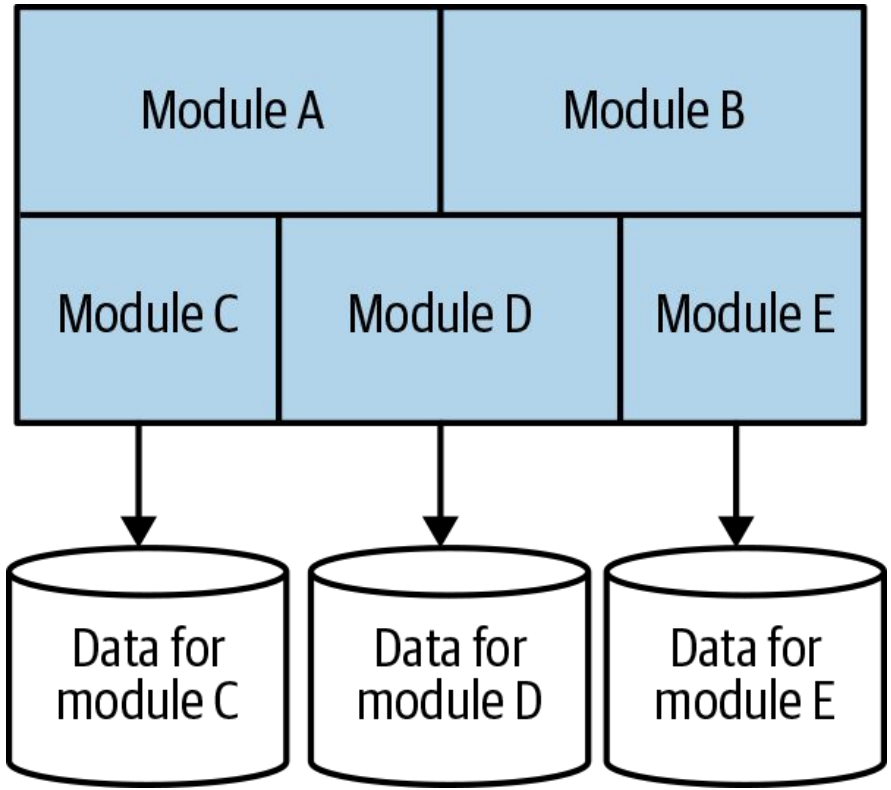


Modular Monolith

fwdays

Main attributes:

- Single codebase - all functionality are in one application
- Modules encapsulate business logic based on domain
- Tightly Coupled Components
- Single tech stack and Single Build and Deployment
- Scalability via scaling of full app

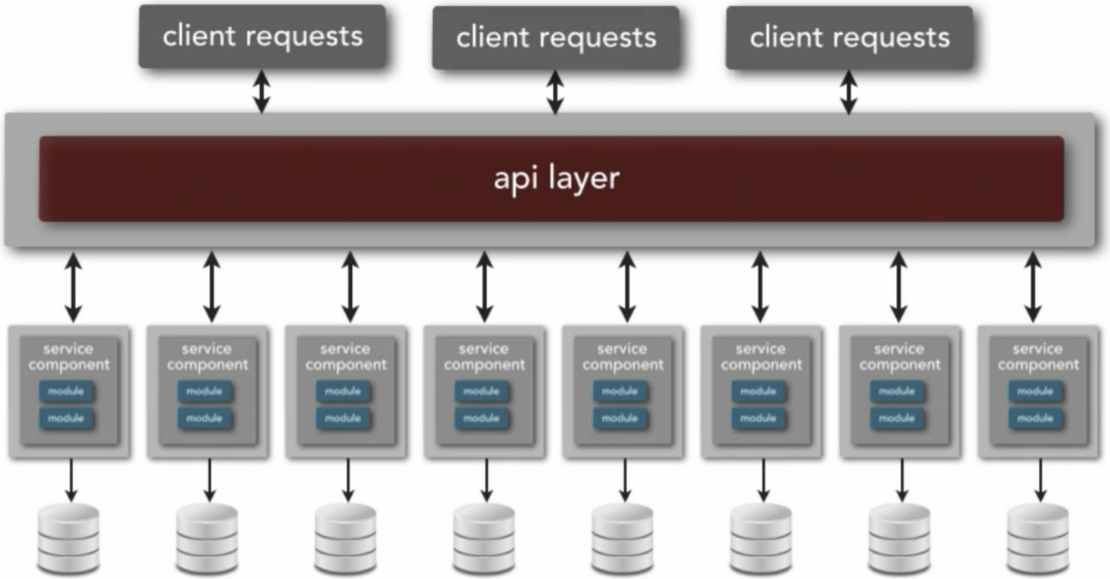


Microservices

fwdays

Main attributes:

- Minimum functionality in service
- Services are distributed
- DB per service
- Separate deploy and easy scalability
- API Layer
- Diverse Technology Stacks
- Loose Coupling - minimum dependencies between services

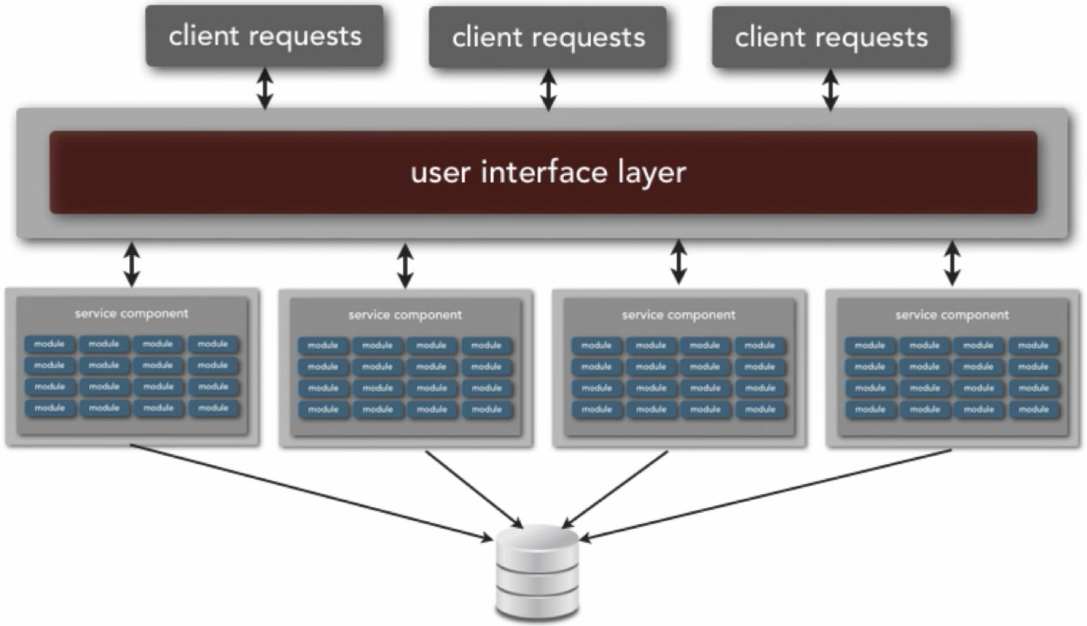


Service-Based architecture

fwdays

Main attributes:

- Macro-services is possible
- Separate deploy is not required
- DB per service is not required (multiple services can connect to 1 DB)
- Diverse Technology Stacks
- Potential Quick start because can be Macro-services
- Easy evolutionary approach

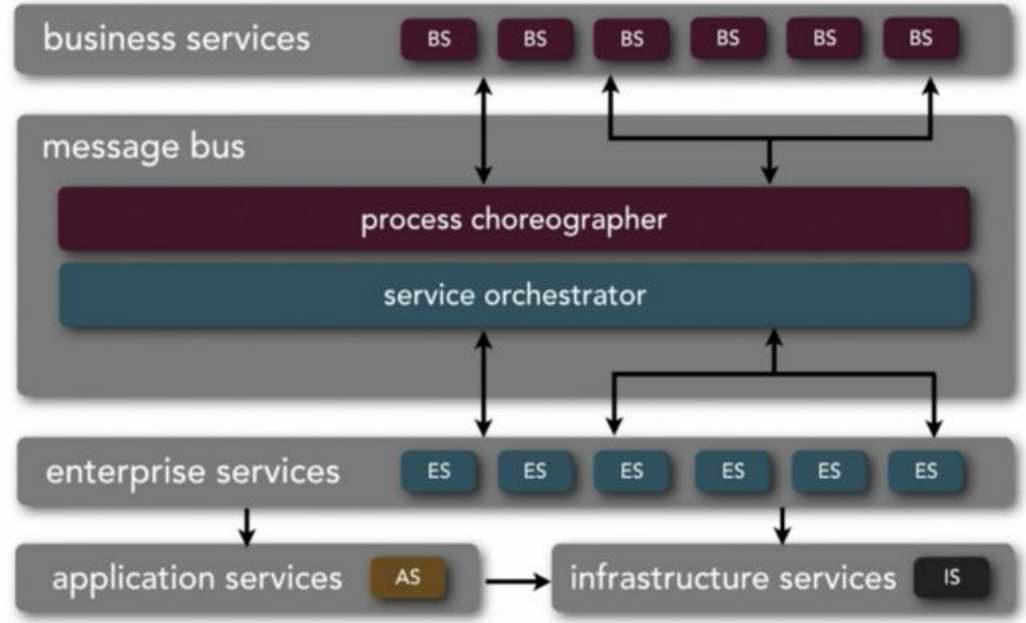


Service-Oriented architecture

fwdays

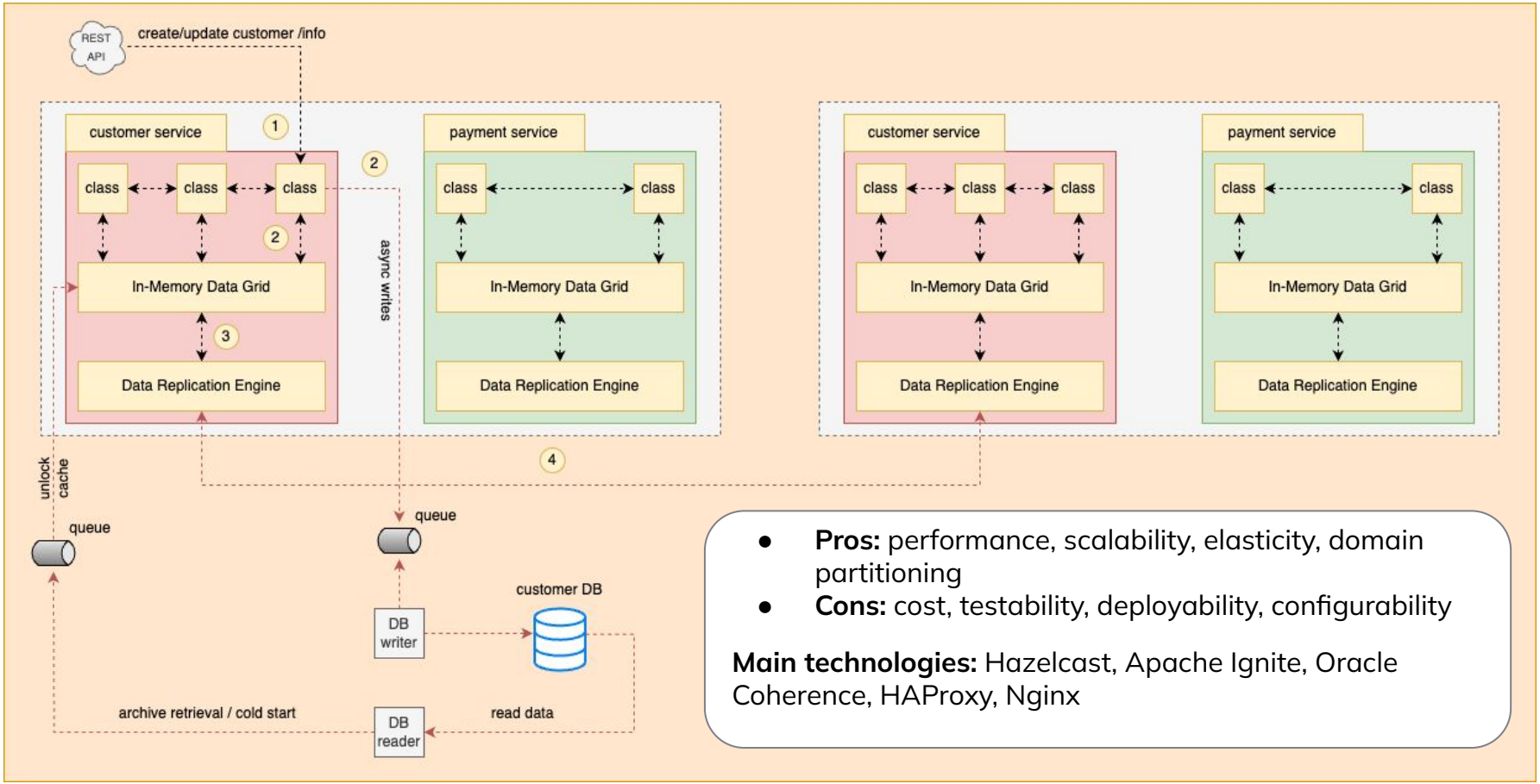
Main attributes:

- Can be macro and micro services
- Business Process Composition
- Strict specific types of services
- Diverse Technology Stacks
- Message bus for process choreography and service orchestration



Space-Based architecture

fwdays



- **Pros:** performance, scalability, elasticity, domain partitioning
- **Cons:** cost, testability, deployability, configurability

Main technologies: Hazelcast, Apache Ignite, Oracle Coherence, HAProxy, Nginx

Hint: Architectural styles worksheet

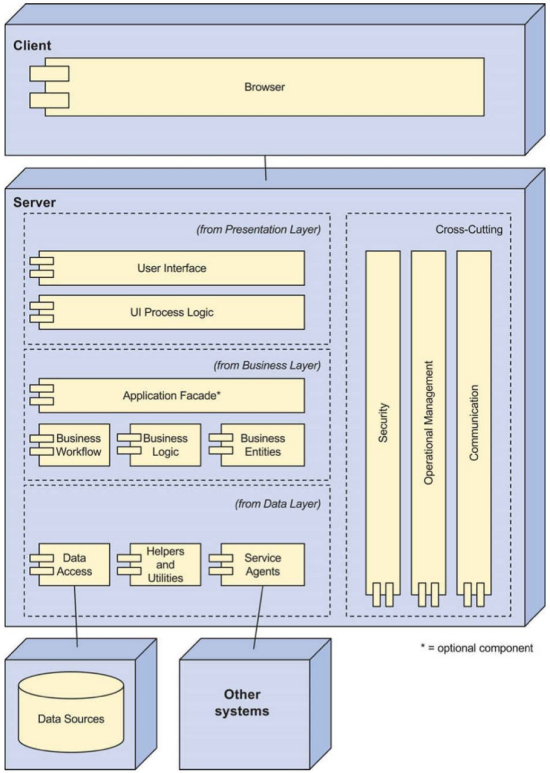
fwdays

	layered	modular monolith	microkernel	microservices	service-based	service-oriented	event-driven	space-based
agility	★	★★	★★★	★★★★★	★★★★	★	★★★	★★
abstraction	★	★	★★★	★	★	★★★★★	★★★★	★
configurability	★	★	★★★★	★★★	★★	★	★★	★★
cost	★★★★★	★★★★★	★★★★★	★	★★★★	★	★★★	★★
deployability	★	★★	★★★	★★★★★	★★★★	★	★★★	★★★
domain part.	★	★★★★★	★★★★★	★★★★★	★★★★★	★	★	★★★★★
elasticity	★	★	★	★★★★★	★★	★★★	★★★★	★★★★★
evolvability	★	★	★★★	★★★★★	★★★	★	★★★★★	★★★
fault-tolerance	★	★	★	★★★★★	★★★★	★★★	★★★★★	★★★
integration	★	★	★★★	★★★	★★	★★★★★	★★★	★★
interoperability	★	★	★★★	★★★	★★	★★★★★	★★★	★★
performance	★★★	★★★	★★★	★★	★★★	★★	★★★★★	★★★★★
scalability	★	★	★	★★★★★	★★★	★★★	★★★★★	★★★★★
simplicity	★★★★★	★★★★★	★★★★	★	★★★	★	★	★
testability	★★	★★	★★★	★★★★★	★★★★	★	★★	★
workflow	★	★	★★	★	★	★★★★★	★★★★★	★

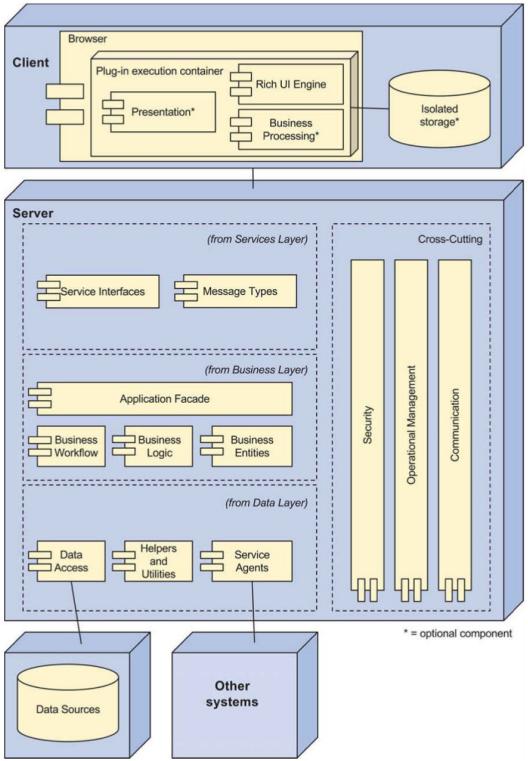
https://www.developertoarchitect.com/downloads/architecture-styles-worksheet.pdf

Architectural Styles for Front-End

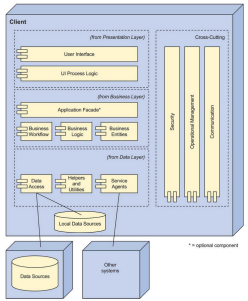
You can start with Reference Application Architectures



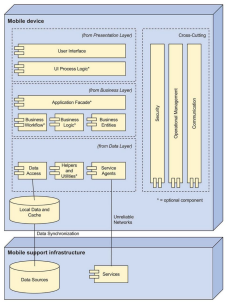
1. Web Application



2. Rich-Internet Application



3. Rich Client Application



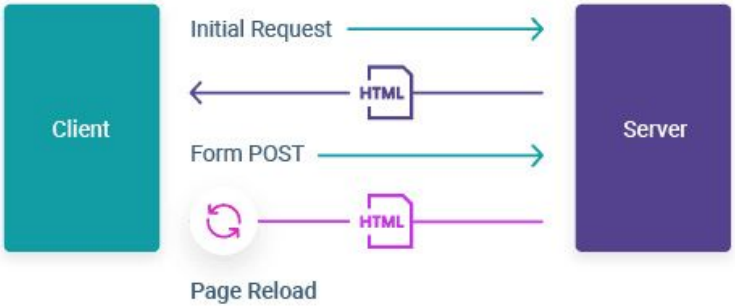
4.Mobile Client Application

https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658107(v=pandp.10)

Front-End architectural styles

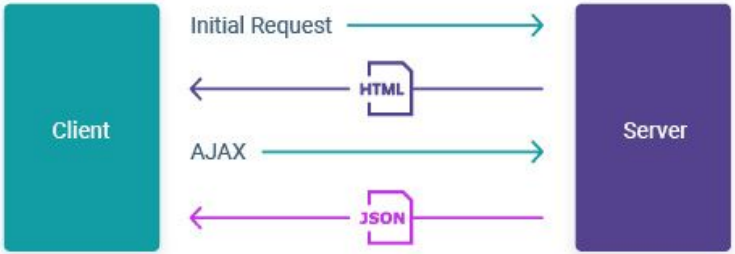
fwdays

Multi-Page Lifecycle



Multi-Page application is web applications that reload the entire page and display HTML from the server each time.

SPA Lifecycle



SPA is an style for web applications that aims to provide a seamless and more responsive user experience by loading all necessary resources (HTML, CSS, JS) on the initial page load and dynamically updating the content as the user interacts with the application, without requiring full-page reloads.

Front-End architectural styles tradeoff

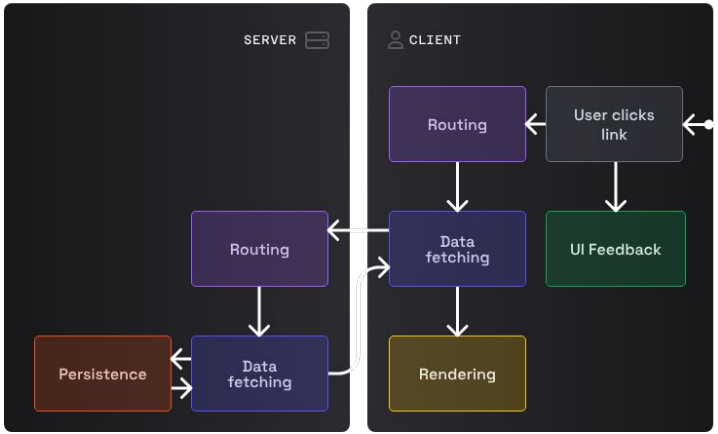
fwdays

	SPAs	MPAs
PERFORMANCE	Faster loading time	Slower loading time
DEBUGGING	More difficult	Well supported by debugging tools
DEVELOPMENT	Fast	Slower, more complex
MAINTENANCE	Fast & easy	Slower
SECURITY	simplified	More challenging
SEO	Limited	Easier & more effective
COST	More expensive	Less expensive
SCALABILITY	Not scalable	Scalable

Some “enhanced” alternatives

fwdays

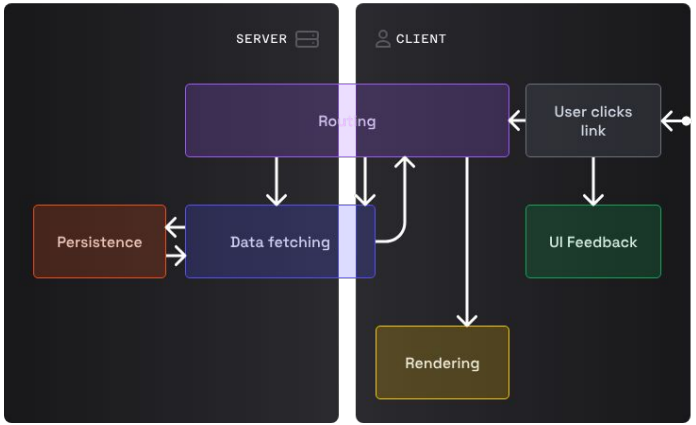
Progressively Enhanced Multi-Page App (PEMPA)



When the user requests the document for the first time, the same thing happens here as it does in the MPA example.

Client-side Navigation: When the user clicks an anchor element with an href that is within our app, our client-side data fetching code prevents the default full-page refresh behavior and uses JavaScript to update the URL.

Progressively Enhanced Single Page App (PESPA)



Document requests with PESPA is effectively identical to PEMPA. The initial HTML needed for the app is sent straight from the server and JavaScript is also loaded to enhance the experience for user interactions.

Client-side Navigation: When the user clicks a link, we’ll prevent the default behavior. Our router will determine the data and UI needed for the new route and trigger data fetching for whatever data the next route needs and render the UI that’s rendered for that route.

EPICWEB.DEV

https://www.epicweb.dev/the-webs-next-transition

fwdays

Thank You

Oleksandr Savchenko
Email: dev.olsav@gmail.com
LinkedIn: https://www.linkedin.com/in/o-savchenko/

Oleksandr
Savchenko