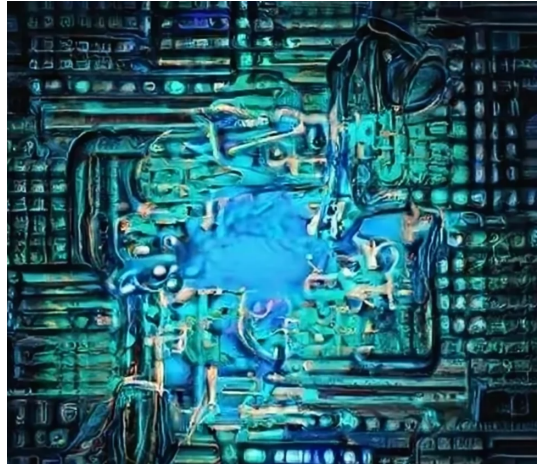
Decomposing Monoliths

CSSE6400



Richard Thomas

May 20, 2024

Question

What are the benefits of a monolith architecture?

Question

What are the benefits of a monolith architecture?

Answer

- Simple deployment

Question

What are the benefits of a monolith architecture?

Answer

- Simple deployment
- Simple communication between modules

Question

What are the benefits of a monolith architecture?

Answer

- Simple deployment
- Simple communication between modules
- Simple system testing & debugging

Question

Why do monoliths have a bad name?

Question

Why do monoliths have a bad name?

Answer

- Many legacy system nightmares were monoliths

Question

Why do monoliths have a bad name?

Answer

- Many legacy system nightmares were monoliths
- Easy to defeat modularity

Question

Why do monoliths have a bad name?

Answer

- Many legacy system nightmares were monoliths
- Easy to defeat modularity
- Cannot scale components of system

Question

Why do monoliths have a bad name?

Answer

- Many legacy system nightmares were monoliths
- Easy to defeat modularity
- Cannot scale components of system
- Monolith databases scale poorly

Question

What can be done if a monolith architecture is no longer suitable?

Question

What can be done if a monolith architecture is no longer suitable?

Answer

- Greenfields replacement

Replacement

- Pro: Can choose any suitable architecture.
- Risk: Developing a new system and maintaining existing.

Question

What can be done if a monolith architecture is no longer suitable?

Answer

- Greenfields replacement
- Migrate to another architecture

Migration

- Adaptive maintenance, changing architecture slowly.
- Some limitation on choice of architecture, but most sophisticated architectures can be used.

Question

How do I migrate a monolith to a new architecture?

Question

How do I migrate a monolith to a new architecture?

Answer

Decompose the monolith into services.

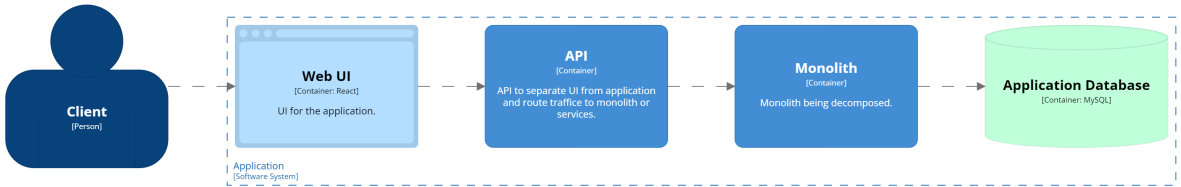
Implies a service-based or microservices architecture.

Strangler Fig Pattern

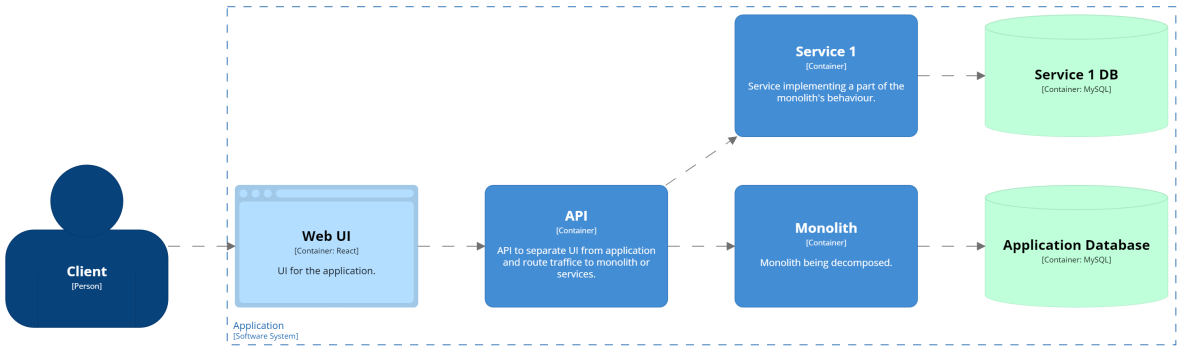
- Develop API for application's UI
- Proxy intercepts API calls
  - Proxy directs calls to application or new services
- Implement a service
  - Redirect calls to service
- Progressively replace monolith
- Shadow & Blue-Green Deployment

- May already have an API if the UI is a web or mobile app.
- Initially deploy proxy and new interface into production, with only existing monolith. Test it works as expected.
- Shadow deployment to test service with application.
- Blue-Green deployment to switch over to using service.

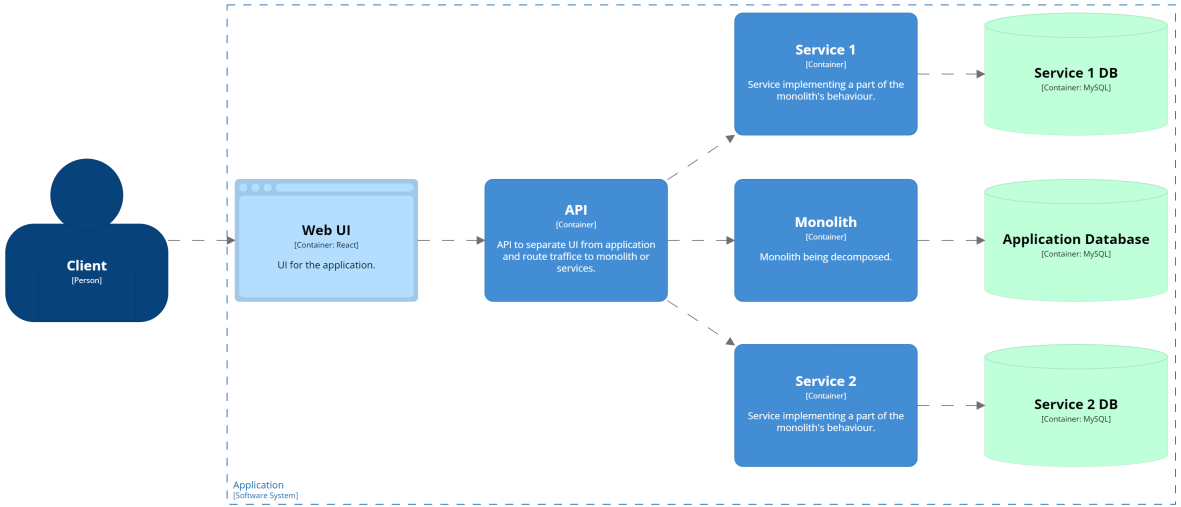
Monolith Deployment



Monolith Decompose: Step 1



Monolith Decompose: Step 2



Decomposition Process

- Identify bounded-contexts
- Simple first service
  - e.g. Authentication
- Minimise dependency from services to monolith
  - Monolith may use services

- Use first service (or first few) to validate approach and deployment infrastructure.
- Minimise changes required in monolith.

- Reduce coupling between bounded-contexts
  - e.g. Customer account management
    - Profile, Wish List, Payment Preferences – separate services
- Decouple vertically
  - Service delivers entire bounded-context
    - Data is decoupled from monolith

- Account management may be tightly coupled in monolith. Separate each aspect (context), one at a time.
- Do not focus only on UI or internal components, service needs to implement all parts of the business process.
- Data management needs to be decentralised.

- Focus on pain points
  - Bottlenecks
  - Frequently changing behaviour
- Rewrite, don't reuse
  - Redesign for new infrastructure
  - Reuse complex logic
    - e.g. Discounts based on customer loyalty and behaviour, bundle offers, ...

- Extract services that deliver highest value.
- What contexts may need to scale more than others?
- What contexts change more frequently and benefit from separate deployment?
- Services deliver capabilities provided by monolith.
- Most often it is better to rewrite the capability to take advantage of new infrastructure.
- Only reuse code that has complex logic that will be difficult to duplicate and test fully.

- Refactor monolith
  - Use service to deliver application functionality
    - Monolith may need to invoke service
  - Remove service logic from monolith

- Atomic replacement of monolith behaviour by service's behaviour.
- Don't deploy production code with service behaviour left in monolith. Leads to a maintenance nightmare determining where behaviour is used, or it may be used in both the monolith and service.

Stepwise Decomposition

Replace application functionality one service at a time.

Definition 1. Macroservice

Separate service, but may span more than one domain or share a database with the monolith or other services.

- Similar scalability and deployment issues to a monolith, but grouped by clusters of macroservices if they share a database.
- Interim step to build microservices.

Definition 2. Nanoservice

Service that depends on other services and cannot be deployed independently – its context is too small.

- Anti-pattern where services are too fine grained and need to be coupled to deliver business processes.
- Some use the term "nanoservice" to refer to independently deployable functions, similar to serverless architecture.