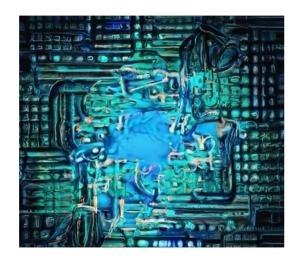# Decomposing Monoliths

CSSE6400

Richard Thomas

May 26, 2025

What are the benefits of a monolith architecture?

What are the benefits of a monolith architecture?

- Simple deployment

What are the benefits of a monolith architecture?

- Simple deployment
- Simple communication between modules

What are the benefits of a monolith architecture?

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

Why do monoliths have a bad name?

Why do monoliths have a bad name?

- Many legacy system nightmares were monoliths

Why do monoliths have a bad name?

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

Why do monoliths have a bad name?

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

Why do monoliths have a bad name?

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

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

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

- Greenfields replacement

Replacement

- Pro: Can choose any suitable architecture.

- Risk: Developing a new system and maintaining existing.

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

- 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.

How do I migrate a monolith to a new architecture?

How do I migrate a monolith to a new architecture?

Implies a service-based or micro-services architecture.

Decompose the monolith into services.

Strangler Fig Pattern

- Develop API for application's UI

May already have an API if the UI is a web or mobile app.

Strangler Fig Pattern

- Develop API for application's UI
- Proxy intercepts API calls
  - Proxy directs calls to application or new services

Initially deploy proxy and new interface into production, with only existing monolith. Test it works as expected.

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

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

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
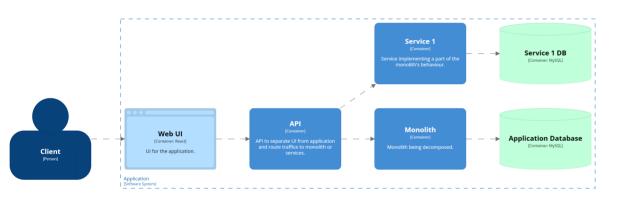- Shadow & Blue-Green Deployment

- Shadow deployment to test service with application.
- Blue-Green deployment to switch over to using service.

Monolith Deployment

Client
[Person]

Web UI
[Container: React]

UI for the application.

API
[Container]

API to separate UI from application and route traffic to monolith or services.

Monolith
[Container]

Monolith being decomposed.

Application Database
[Container: MySQL]

Application
[Software System]

Monolith Decompose: Step 1

Client
[Person]

Web UI
[Container: React]

UI for the application.

API
[Container]

API to separate UI from application and route traffice to monolith or services.

Service 1
[Container]

Service implementing a part of the monolith's behaviour.

Service 1 DB
[Container: MySQL]

Monolith
[Container]

Monolith being decomposed.

Application Database
[Container: MySQL]

Application
[Software System]

Monolith Decompose: Step 2

- Identify bounded-contexts

- Identify bounded-contexts
- Simple first service
  - e.g. Authentication

Use first service (or first few) to validate approach and deployment infrastructure.

Decomposition Process

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

- Monolith decomposition will continue.
  - Don't want to have to refactor service once functionality is removed from monolith.
  - Don't want services to end up depending on each other.

- Minimise changes required in monolith.
  - But expect a fair bit of work removing features from monolith.

- Reduce coupling between bounded-contexts
  - e.g. Customer account management
    - Profile, Wish List, Payment Preferences – separate services

Account management may be tightly coupled in monolith. Separate each aspect (context), one at a time.

- 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

- Do not focus only on UI or internal components.
  - Service needs to implement *all* parts of the business process.
    - Slice of cake analogy, you want a vertical slice containing all layers, not just the icing on top.

- Data management needs to be decentralised.

- Focus on pain points
  - Bottlenecks
  - Frequently changing behaviour

- 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?

- 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, ...

- Services deliver capabilities provided by monolith.
  - Usually 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.
    - May be used in both the monolith and service.

Stepwise Decomposition

Replace application functionality one service at a time.

> **Definition 0.** 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 0. 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.