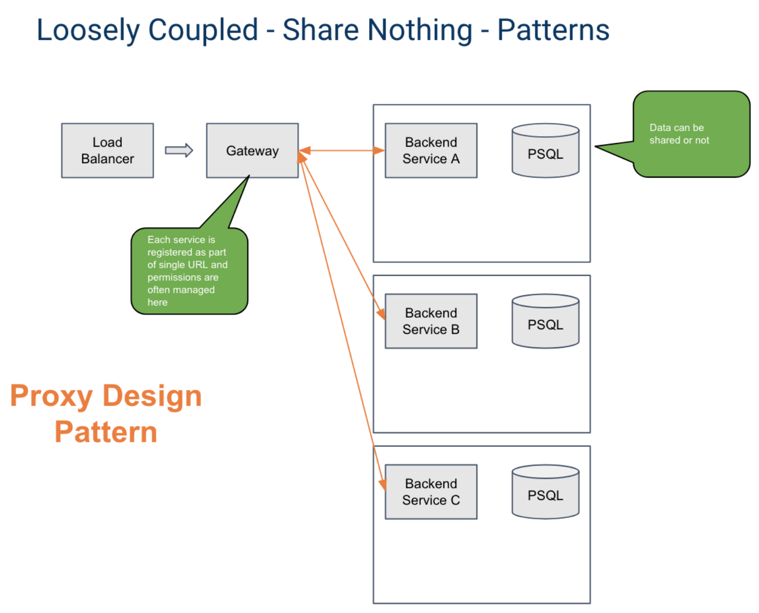
Microservice Flex Patterns

Greg Lind Follow
Apr 4 · 4 min read

Walhall is Humanitec's deployment and open source market place for rapid application building and a key component of our Logical Platform as a Service. But what really makes development rapid and flexible is much more than just the market place and deployment tool. The most important part is the BiFrost Core API that is deployed with each application.

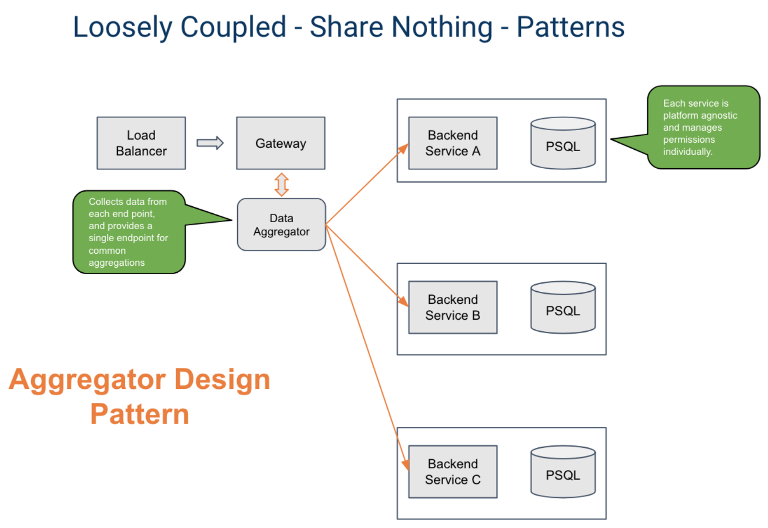
BiFrost acts as a bridge between the pre-configured and custom frontend clients (built with Midgard-Angular or Midgard-React) and the backend services deployed via Walhall. The BiFrost core api service provides multiple light weight services such as a configurable and flexible API gateway, data mesh aggregation service and an authentication and authorization layer. It's built around a decoupled Microservice architecture pattern but offers more flexibility then most, by giving the developer the option to pick and choose the parts of each Share Nothing, Proxy or Shared Data Microservice pattern they want to use with each independent service.

To get a better idea of how this works lets first review a few of the more popular MicroService patterns many developers use and how they can be implemented in Walhall.


Loosely Coupled - Share Nothing - Patterns
Load Balancer
Gateway
Backend Service A
PSQL
Data can be shared or not
Each service is registered as part of single URL and permissions are often managed here
Backend Service B
PSQL
Proxy Design Pattern
Backend Service C
PSQL

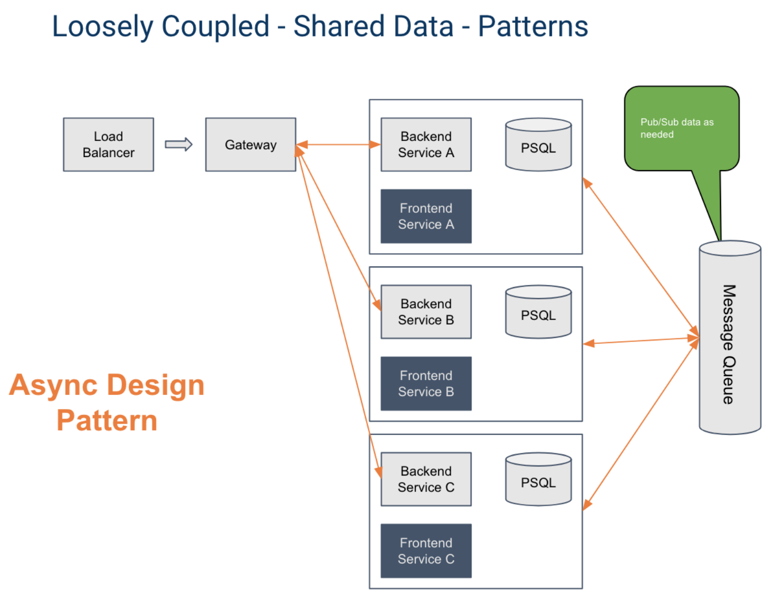
Proxy Design pattern as implemented in the Humanitec BiFrost core.

The proxy pattern provides a single url endpoint along with basic authentication and authorization across multiple end points. This pattern gives the Frontend or client developers a single URL and endpoint to manage for all data requests. It can also be extended to provide pass-through authentication via a token auth process like JWT that provides basic authentication to each Microservice.


Loosely Coupled - Share Nothing - Patterns
Load Balancer
Gateway
Backend Service A
PSQL
Each service is platform agnostic and manages permissions individually.
Collects data from each end point, and provides a single endpoint for common aggregations
Data Aggregator
Backend Service B
PSQL
Aggregator Design Pattern
Backend Service C
PSQL

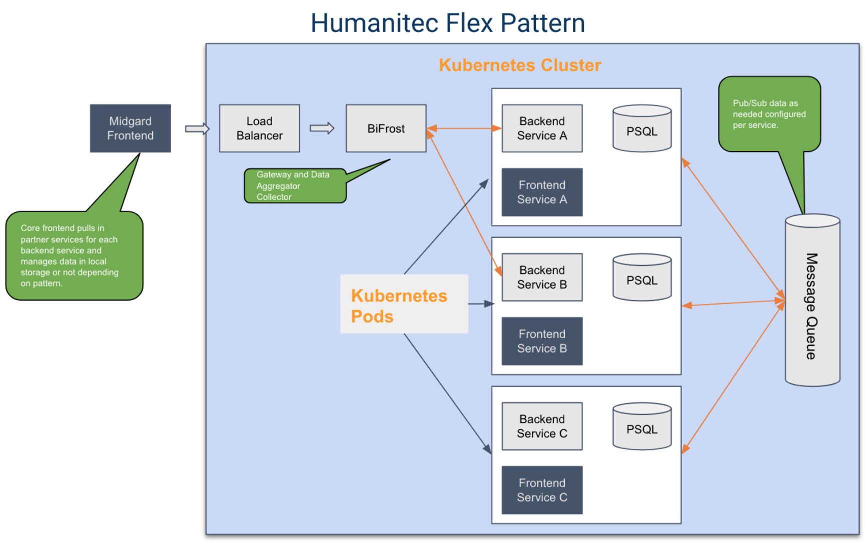
Aggregator Design pattern as implemented with the Humanitec BiFrost core

An extension to the proxy pattern is the aggregation design pattern. This common pattern usually uses a tool like GraphQL at it's center and allows you to aggregate all or parts of your end point data in one gateway service as a data query tool. This allows for decoupled services while still providing a single URL through a gateway for each service. It's also provides Frontend developers with one endpoint to query each of the Microservice endpoints as well as perform basic linking or aggregating of the data. In our case we provide a data grid tool that uses Swagger docs to create a simple REST API data mesh for all the related endpoints and allows the developers to pre-configure the desired join queries and cache levels.



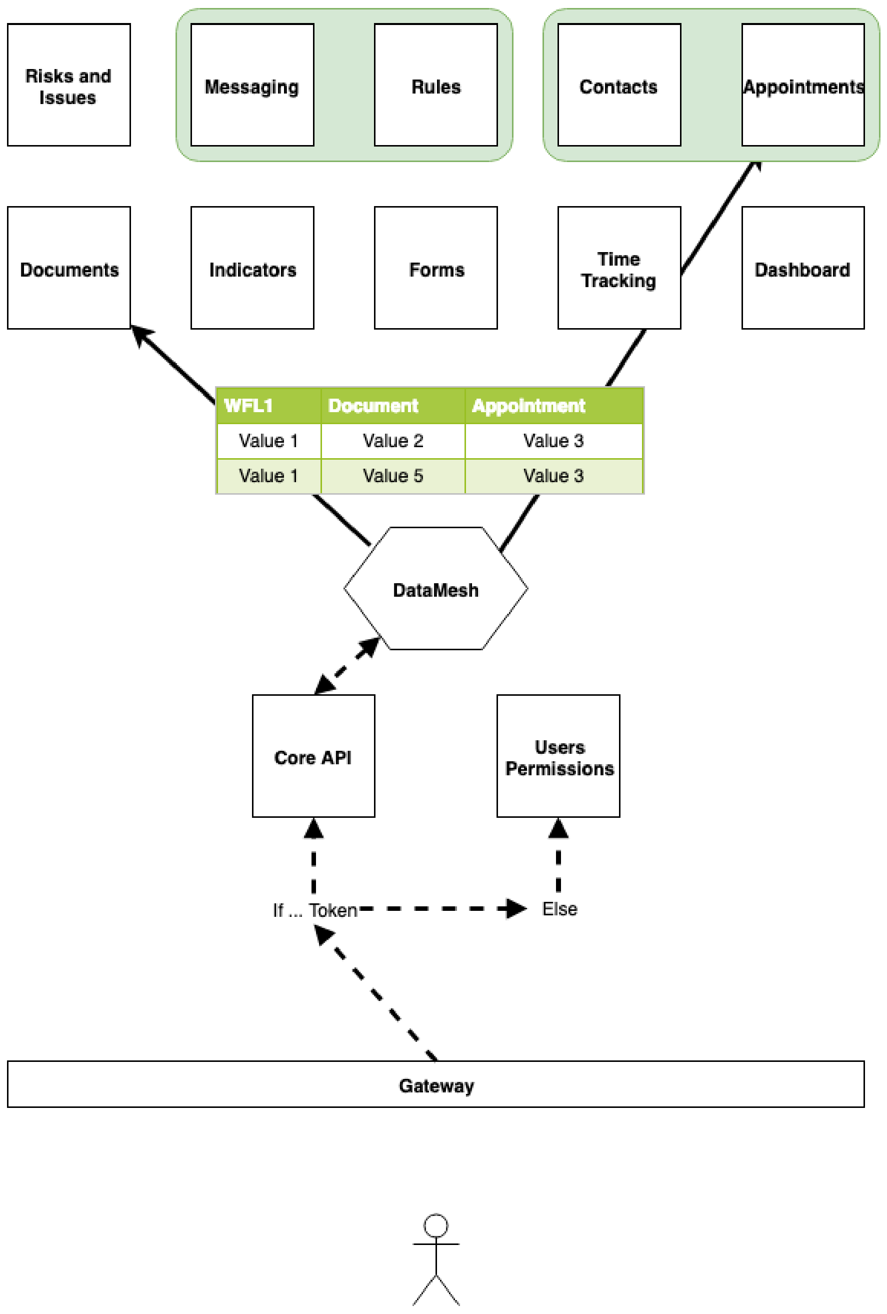
Asynchronous Design patter as implemented in Humanitec's BiFrost core.

The Asynchronous design pattern can be used with a gateway or not, but each Microservice can publish data to a messaging queue like RabbitMQ and subscribe to the data it needs from other services. This requires a bit more upfront configuration for each service but provides the needed data and only the needed data to each end point and can help to reduce query time compared to an aggregation service.



Humanitec BiFrost Flex Pattern

The Humanitec Flex Service built into BiFrost allows the developers of each LogicModule to choose how they want to share data, either via a messaging Queue or Gateway and API data mesh, or a fully decoupled backend for frontend pattern. The BiFrost service provides a simple way to use “workflows” as unique identifiers via UUID’s that are generated in BiFrost and that can be registered with the core service and each related LogicModule entry, then used as a lookup service between entries in each backend service endpoint.



BiFrost and the data mesh

The Flex pattern does not require the developer to subscribe to a particular pattern upfront because all the needed services are running in the BiFrost core. This way depending on the use case, type and shape of the data coming into a service developers can choose how to implement each Microservice in the optimal way for that service. The only requirement is that each service provides an OpenAPI swagger doc for each service. This helps ensure a standard style and practice for each API service, but also allows us to aggregate the service where needed into the gateway and data mesh.

These services also allow a developer to create lookup service as join tables for data across Microservices where needed and apply flexible centralized role based permissions for them as well. It gives you the developer a lot of flexibility to design an application the way you want, decouple and part out the work where you want and focus on the building of innovative and flexible applications.

. . .

Originally published at humanitec.com.