FHIR® Architecture Decisions

Selecting a FHIR server and choosing between facade, hybrid, and FHIR-native approaches

v1.0

Darren Devitt

FHIR® Architecture Decisions

Selecting a FHIR server and choosing between facade, hybrid, and FHIR-native approaches

by Darren Devitt

Version 1.0

darrendevitt.com

Contents

Introduction

On Microsoft Azure, AWS or Google Cloud, you can spin up a new FHIR server in about ten minutes.

On AWS, you can select a checkbox to pre-populate the server with rich synthetic patient data – Observations, Encounters, Procedures, MedicationRequests and Claims. This takes another ten minutes.

On Google Cloud, there is a pre-selected option to sync your FHIR data with a BigQuery data warehouse. Configuring this adds an extra five minutes.

Once your server is up and running, you can create a simple C# or Java project, add the appropriate SDK and write a few lines of code to send or receive data from your new FHIR server.

From start to finish: one hour. This is how easy it is to start a FHIR project in 2026.

But it's deceptive.

Production reality

Getting started may take one hour. Releasing an application to production and seeing real patient and clinical data flowing into and out of your FHIR server may take one to three years.

Perception and reality are at odds when it comes to building solutions that use FHIR. That first day, when your junior developer shows you patient data flowing into your FHIR server, is only the start of the years of development work that follows.

Work that is often filled with bad decisions and costly mistakes. Mistakes that lead to code rewrites, architecture reversals, cost overruns, missed deadlines and poor solutions released to production.

The missing guidance

There is no single "best" architectural model when it comes to implementing FHIR. Technical and solution architects have to work out for themselves what the different models are and which one best meets their requirements.

The HL7 organization defines the resources and the API. They manage future versions and common Implementation Guides, but they have nothing to say about how your enterprise application should be built.

Most advice comes from vendors or from articles and presentations that document single implementations.

There is no "*FHIR Architecture Patterns*" book you can buy on Amazon. There is no impartial guide on "*How to Choose a FHIR Server.*" Every organization is left to make these key decisions on their own.

I've seen FHIR facades where the performance is so poor they are unusable. Yet they meet the regulations. I've seen hybrid approaches where the data in the FHIR server is regularly out of sync with the data in the "source of truth" database. Yet they're released anyway. And I've seen FHIR-native systems with no data governance at all. Systems that rely on FHIR profile validation in place of proper data governance and master data management.

In all of these cases, architectural choices were made when the technical team had the least understanding of FHIR. Incorrect assumptions led to weak decisions, and in some cases, wrong decisions. The resources that might have helped were biased, insufficient or non-existent.

The result: poor inputs led to poor outputs.

This guide exists to address that gap.

What this guide covers

It will help you decide which architectural model best meets your requirements. I'll discuss FHIR facades, hybrid FHIR systems and FHIR-native implementations, and I'll help you decide between them.

I will help you work out which FHIR server is most suitable or whether an open-source server might better fit your needs. And I will help you identify what you might need to build around your FHIR server so that it can safely work with live patient data.

The architectural decisions you make in the first three months will determine the success or failure of your project.

What this guide does not cover

This guide does not address EHR integrations, FHIR data modeling, specific regulations you need to meet or Implementation Guides you might need to support.

If your FHIR requirements are limited to making calls to another organization's FHIR API, you won't find anything in this guide to help you.

Chapter 1
The Role of the FHIR Server

The expectation in the early days of FHIR was that each organization would implement a FHIR API themselves and populate FHIR resources from their existing databases. That's not how things turned out.

Commercial FHIR server vendors emerged to feed the new market, delivering products that combined a FHIR API with an underlying database. These "off-the-shelf" servers – and their open-source equivalents – now dominate the FHIR server landscape. It's rare in 2026 for an organization to code a complete FHIR API of their own.

What a FHIR server actually does

A FHIR server is a combination of a FHIR API – as defined in the FHIR standard – and a database, all wrapped up into a single product. Most organizations working with FHIR use versions R4 or R5, which contain 145 and 157 different resource types. The API documents many hundreds of search parameters for these resource types and a small number of "operations". Operations are endpoints that carry out specific logic.

Commercial FHIR servers typically support one or both of these versions of FHIR and will normally implement most – if not all – of the search parameters. The degree to which they support "official" operations varies considerably and is a key point of differentiation.

While a server combines a FHIR API and a database, that database is normally concealed and inaccessible to users – even to developers working with the server. In practice, this means access to data stored in the server must always be made via the API. No SQL queries can be run.

This can be problematic for some business use cases and has given rise to different solutions from different vendors – another major point of differentiation.

Figure 1.1 illustrates what a "bare-bones" FHIR server looks like. A consumer makes a GET, POST or other request to the server to access or update particular resources, sometimes using search parameters to filter the response. For "write" operations, the server is capable of performing some validation.

At last count, there were upwards of twenty commercial FHIR server vendors and eight open-source server projects. These include managed FHIR servers in the cloud as well as independent vendors selling "enterprise" servers that organizations host and manage themselves.

Whichever direction your FHIR implementation takes you, it is likely that you will interact with a commercial or open-source FHIR server like these in some way, either as a core component of your project or as a way to shortcut your technical implementation.

It should be clear already that not all FHIR servers are equal. There is real diversity in the FHIR server landscape, which means real care needs to be taken when selecting a vendor.

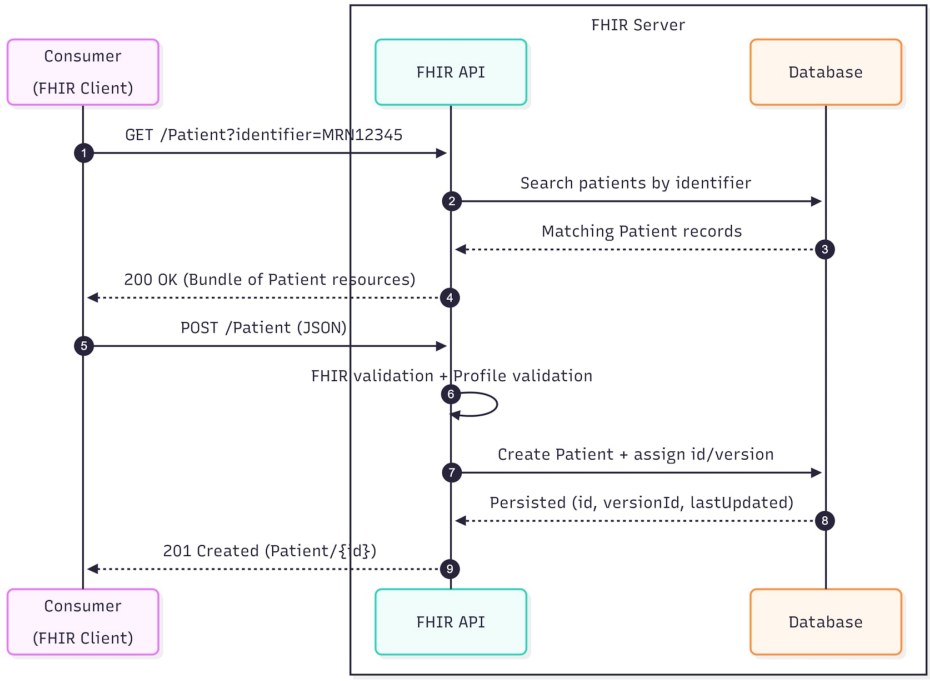

Figure 1.1

What it does not do

Let's return to what a FHIR server does. It receives a request, validates it as best it can, carries out the request, and returns a response. That's it. That's all the FHIR server is responsible for.

Everything else that you might be expecting it to do is the responsibility of another application, process or service. Usually, you will have to configure, set up or build these other things yourselves. Some vendors provide ways to handle these. Others do not.

Secondly, FHIR is not a database.

The term "FHIR Database" is one that pops up again and again when the wider business starts talking about FHIR. There is a lack of understanding beyond the technical level of what a FHIR server is and what it is not.

You cannot run labor-intensive queries using a FHIR API. Queries that are commonly run against SQL databases or data warehouses cannot be run against a FHIR server. If the business wants to populate dashboards using aggregated metrics drawn from your FHIR server, this cannot be done *by* the server or directly *from* the server.

A misunderstanding across the business of what a FHIR server is and what it is not is one of the most common problems on FHIR implementation projects. It leads to unmet expectations, code rewrites and project delays.

Why the server decision is so important

The FHIR server needs to meet the business requirements.

Not all FHIR servers are suitable for all projects and use cases. Choose the wrong one and you may find your patient data residing in the wrong country. You may find applications are unable to retrieve the data they expect. You may find that two years into your project you have to change vendors because you cannot meet the needs of more than one client.

The server decision influences your wider architectural decisions. Vendors take different approaches to key functionality such as multi-tenancy, intercepting re-

quests, SMART on FHIR, RBAC and more. This means that once you begin working with a particular server your systems will become more and more reliant on that server provider's "ways of doing things."

Vendor lock-in is real, though rarely intentional. While FHIR data can be moved relatively easily from one server to another, in many key areas FHIR architecture is vendor-specific. Switching server providers after implementation is completed then introduces major rewrites and structural changes at the architectural level.

The consequences of choosing the wrong server are severe.

The "platform illusion" trap

Businesses often misunderstand what a FHIR server is. This can lead to expectations that a commercial FHIR server from a big-name company can solve a wide range of problems and meet broad business needs. That the server is essentially a "platform" that fulfills many roles.

This causes business leaders – and sometimes technical leaders – to underestimate what they need to build around the server and to overestimate what the server itself actually does.

Examples of "platform" features that a FHIR server does not fulfill:

- Business validation
- MPI and EMPI
- Management of master data and "golden records"
- Data governance
- Comprehensive auditing
- Fine-grained access control
- Producing analytical data

Responsibility for these does not lie with the FHIR server. If required, you will need to manage and implement them yourselves as separate projects.

A second problem caused by the "platform" misunderstanding is that it leads some organizations to adopt a FHIR-native approach without understanding what building and maintaining one actually requires. I'll discuss this in later chapters.

A FHIR server has a clearly defined role. It allows you to expose and receive patient and clinical data via a universally understood API. Don't expect more from it than this.

Chapter 2
Three Architectural Models

These three models describe the role FHIR can play in your system architecture. No *one* is inherently better than the others. Your individual project requirements and limitations should dictate which one you lean toward.

1. The facade model

A FHIR facade is an implementation of a FHIR API that draws data from existing databases or data warehouses, dynamically converts it into FHIR resources, and returns them to the consumer. The bulk of the work is mapping from the core data structure to FHIR resource types.

Performance is often a problem with facades, as gathering all the data required to construct FHIR resources on the fly can involve multiple queries, sometimes to many different databases or services.

A key advantage of the facade is that it allows organizations to expose their data to external consumers without duplicating it in a FHIR server. It also allows existing data structures to remain unchanged.

Facades are often used as interim measures when regulatory or other requirements demand that data be made available via FHIR and the organization may not want to invest significant time or resources in a FHIR server.

2. The hybrid model

"Hybrid" describes a FHIR architecture where data from existing systems is kept in sync with a FHIR server. Consumers use the FHIR API to access a copy of the data, while the legacy systems remain the source of truth for all data.

Hybrid systems deliver high-performing FHIR APIs and require few – if any – changes to legacy systems. They enable incremental migrations to FHIR, as data from different systems can be synced one system at a time. It's an attractive model because it preserves existing systems without requiring wholesale replacement.

But they are difficult to build and maintain. Complexity arises when legacy data structures and terminologies are converted to FHIR resources, and when errors occur during the sync process. These need to be carefully managed.

While most hybrid systems are read-only, some handle write requests via a facade that transforms and routes them to the legacy systems.

Whenever the word "hybrid" comes up in the FHIR context, it's important to understand precisely what is meant. Online discussions often focus on single implementations and ignore the variation across hybrid systems – no two are ever quite the same.

3. The FHIR-native model

The term FHIR-native came into regular use around 2020. At its heart, it means the FHIR server is the source of truth for some or all of its data.

It's particularly popular for greenfield projects, where no legacy systems or data exist. In these cases, read and write requests for all resource types flow into the FHIR server directly, without involving any other system. Domain-specific projects often use FHIR-native where master data remains in legacy systems but transactional data does not. The FHIR server becomes the source of truth for some resource types, while legacy systems maintain ownership of master data, which is kept in sync with the FHIR server.

In both cases, the FHIR server becomes the source of truth for some data.

A key advantage of FHIR-native is that all requests from consumers and data providers are FHIR requests. A second advantage – particularly for new projects – is that FHIR resources become the primary data model. The FHIR documentation becomes their documentation.

A weakness is that components traditionally handled by legacy systems may need to be built. These include data governance, MDM and MPI. This is a lot of work.

The FHIR-native model has been adopted at scale by some national programs as well as by a number of private organizations. Many of these implementations are more complicated than my brief description might suggest.

Before you choose

The three models outlined here are not abstract patterns. They define where data lives, how it is accessed and which systems are responsible for it.

To consumers, they appear identical. In practice, each one comes with constraints that directly affect performance, scalability, data governance and long-term cost.

Before going deeper into each model, you need to be clear on what your system actually needs to do. The next chapter focuses on that – defining the requirements and capabilities that will determine which of these models is viable.

Chapter 3
Clarifying Your Requirements & Capabilities

The most difficult time to make key architectural decisions about FHIR is at the start of a project when the technical and business teams' knowledge of FHIR is at its lowest and requirements are incomplete. This is unfortunate but it's the reality for many new FHIR projects.

Your project and organizational requirements will dictate which FHIR architectural model you decide on and which FHIR server you choose. The nine questions in this chapter cover the key areas that will impact these decisions. You need solid answers for each of them.

1. Why are you exposing data via FHIR?

Organizations adopt FHIR for three main reasons: regulations, interoperability or structured data. Most projects fit into at least one of these.

Regulations

This is the most common reason for organizations to begin working with FHIR. Future use cases may deviate from this as the benefits of FHIR become apparent, but meeting a specific regulation is often the trigger. The regulation may or may not mention FHIR but FHIR is the mechanism you're using to meet this requirement. Regulations are usually explicit about the precise data that should be exposed via FHIR and they are often accompanied by Implementation Guides further dictating the expected values.

Ask yourself this: are you aiming to meet the minimum requirements as outlined in the regulation or are you planning to exceed them and deliver the spirit of the regulation as well as the letter?

Interoperability

You are seeking to share data with others – internal or external consumers – and have chosen FHIR because it's expected by your consumers or because it's a globally understood format that can be used with little onboarding effort by those same consumers. As with regulations, you rarely need to expose everything at once – it can grow as your consumers and requirements grow.

Structured data

FHIR is a structured data format that is easier for AI systems to process and understand. For this reason it's becoming the preferred way of making healthcare data accessible to LLMs and AI algorithms. 2025 marked the point where AI use became a common standalone reason for adopting FHIR – especially by private businesses. Existing Implementation Guides such as Structured Data Capture standardize FHIR further, making it even more useful for AI ingestion. Unlike regulations and interoperability, the more data you can make available in FHIR format, the better AI performs.

Decision impact

If your reason for using FHIR is because you are obligated to do so by regulation, be aware that regulations often say little or nothing about performance, which means a slow-performing facade may be acceptable if all you're doing is "checking a box". Delivering high-performing FHIR endpoints – which is what all consumers want – might not be possible using a facade, which means you may need to consider a hybrid approach with a FHIR server.

Many server vendors highlight their servers as complying with specific regulations such as "Prior Authorization Final Rule" in the US. This is marketing language and should have no bearing on your server selection. The FHIR server itself does not "meet regulations." The data inside it and the fact that it's accessible via FHIR APIs does.

If interoperability is why you are adopting FHIR and you are sharing data with important consumers, then for reasons already mentioned a facade is often not

sufficient. You should only consider a facade if your consumers are internal and performance is not an issue – overnight batch jobs, for example – or if the scope of the API is small.

A structured data requirement, especially if AI is the consumer, will need fast access to as much data as possible. In this scenario expect scope to expand quickly. You should be looking at the hybrid and FHIR-native models.

2. Are you publishing data or building a source of truth?

This question comes down to one thing: who owns the data?

If your FHIR system receives and processes write requests that create or update data, then you own the data and are the source of truth. Your architecture must reflect that.

If you are publishing data that exists and is managed elsewhere, you do not own it. Write requests will likely flow into legacy systems where data governance already exists. Your job is to map and expose that data via FHIR.

Decision impact

Building a source of truth involves decisions beyond selecting an architectural model. You take on responsibility for implementing features such as data governance, MPI capabilities and master data management. You should choose one of the FHIR-native variants. If you are only publishing data, facade and hybrid are both viable options.

3. Where do write requests land and who has ownership?

You may not know the precise answers yet, but you should understand in broad terms where data is handled and which system is responsible for it.

Source of truth

Where write requests end up determines which system is responsible for which piece of data. For some data that may be an existing or legacy system. For others it

may be the FHIR server. Master data may be treated differently to transactional data. Lack of clarity on data ownership can lead to data drift and unreconciled changes.

Ownership boundaries

Ownership determines who validates data, who can reject an update request and who can resolve conflicts between systems. Without clear boundaries, hybrid designs in particular can quickly degrade as no one knows which data to trust.

Decision impact

The next three chapters go into detail on the many variants that exist within each of the three models. Clear answers here will help you determine not just which model best meets your needs but which variant of that model is the right fit. They will also help you determine what else you need to build: data governance and master data management processes, validation rules engines and data sync methods.

4. Are you sharing existing data or generating new data?

This question strongly influences which architectural model makes sense. If you are working on a greenfield project where the data does not already exist elsewhere, you are not constrained by legacy systems and can choose how the data is stored and made accessible. In these cases, FHIR-native should be a strong contender.

But fully greenfield projects are rare, especially in large organizations where new projects often rely on existing master data. Where significant data already exists in legacy systems, hybrid – and in some cases facade – are often more realistic options.

Decision impact

Entirely new data makes FHIR-native realistic. Existing data often favors hybrid or facade.

5. Who are your data providers and consumers?

You need a clear understanding of who is sending you data and who is reading your data. You need to know their expectations and what commitments have already been made.

Data providers

Is data coming from internal or external providers?

Internal providers usually give you a greater say in the data structure, the timing of requests and how they are handled. Nightly batch jobs, asynchronous workflows and planned downtime are often acceptable.

External providers tend to have stricter requirements. SLAs, uptime expectations and immediate processing become more important and data volumes can be higher.

Data consumers

Internal consumers may have fewer expectations in terms of performance and uptime, and may be more tolerant of imperfect syncing of data from legacy systems.

External consumers have much higher expectations. Your organization may have made commitments to them regarding performance and data accuracy. The volume of requests may be high, requiring scalable and high-performing endpoints.

Decision impact

The expectations of external consumers may rule out the facade model altogether. High request volumes and strict SLAs may also limit your choice of FHIR server.

Accepting data from external providers adds an extra dimension around data governance that may not exist for internal providers.

6. What are the project “deal breakers”?

This question focuses on requirements that directly affect which FHIR server you choose. The server landscape is diverse enough that even small requirements can rule out individual servers or entire classes of servers. Below are a few examples. Your project may have its own set of deal breakers.

Data location

Does it matter to your organization or to your customers where patient data is located? Does it need to be stored in a local server "on-prem" or can it be in the cloud? And if it can be in the cloud, does the country the cloud server runs in matter? Here's an example: if you spin up a new Microsoft Azure FHIR server in Europe, the likely location of that server today is Sweden.

Be careful not to draw conclusions based on a first customer or first market. Sweden might be ok today but not ok for future customers. This question requires business knowledge and foresight.

Authorization

SMART on FHIR is the expected authorization method for FHIR servers. As of 2026 the latest version is v2 but not all FHIR servers support it. You need to understand what your customers and users expect here, as well as any regulations that might apply.

Some projects require capabilities beyond Smart-on-FHIR, such as RBAC and element-level access control. A small number of vendors have done work in this area, but most have not.

Multi-tenancy

A "tenant" may refer to a country, a region, a hospital or a set of hospitals. There is no "best practice" for managing multi-tenancy in FHIR servers. Different servers handle it differently or not at all.

Regulations come into play here as well, with different countries favoring or insisting on different approaches when separating patient data from distinct organizations. Some lean toward logical separation, others insist on physical separation. This means one FHIR server or multiple FHIR servers – a decision with large architectural and cost implications.

FHIR Subscriptions

A FHIR Subscription notifies listening apps that a FHIR resource of a particular type has been created or updated. It's a key feature of FHIR that is relied upon by many consumers, especially in hospital settings.

Not all vendors and servers support Subscriptions. Some have implemented custom workarounds which are not based on FHIR at all and which require custom implementation work from each consumer.

Analytical queries

FHIR is not a database. You cannot run analytical queries that you might run on a PostgreSQL database using a FHIR API. Non-technical leadership often does not understand this, leading to unmet expectations. Ask yourself this question: is there a business expectation that data stored in your FHIR server can later be used for analytical purposes?

If there is, it needs to be brought to the surface at the outset, when server decisions are being made. Vendors have different ways of handling this, from syncing FHIR data with existing data warehouses to supporting open-source projects such as SQL on FHIR. These are non-trivial projects and often create dependencies on a vendor's wider platform.

Decision impact

If a FHIR server does not support a key requirement, you need to drop that server from your shortlist or plan to implement the missing capability yourself. In some cases this will not be possible – if Sweden is the only available data center and your data cannot be located there, you must choose another vendor.

Many vendors will attempt to reassure you that missing features are on their roadmap. Reassurance is not enough. Roadmaps change, planned features get dropped and priorities shift.

Building it yourself may be an option but you need to understand the effort involved and how well it might integrate with a vendor's platform.

7. What can your technical team realistically deliver?

This is a critical question and one that is frequently overlooked. Technical and business leaders often assume their teams can build anything. In reality, lack of expertise is one of the main reasons FHIR projects fail.

Depending on the model you choose, your team may need experience with event-driven design, healthcare data governance, scalable APIs and databases, distributed systems, scoped authorization and MPI – in addition to FHIR and terminology expertise.

An honest assessment of your team's capabilities is required, especially if you're considering FHIR-native.

Decision impact

If your team cannot confidently build it, you can't deliver it. Hybrid requires deep expertise in building and maintaining data sync pipelines, handling failures and resolving data drift. FHIR-native requires all of that plus the ability to build and maintain a full source of truth system for patient data.

8. What are your budget and timeline constraints?

Every project works under financial constraints and deadlines. There are real consequences for not delivering on time and on budget.

Budget

The project itself may have an overall budget, with individual components funded separately. You will likely need to buy a FHIR server, which comes with recurring costs. Additional resources may be required – FHIR and terminology expertise can be particularly difficult to find.

Timeline

Your delivery date may be driven by meeting a particular regulation's deadline, by the expectations of sales and marketing teams or by your own customers. The effort required will vary depending on how much data is exposed via FHIR, where

it resides today and how difficult it is to extract and transform. This applies whether you're building a facade or syncing with a FHIR server. Every additional requirement adds time.

Decision impact

If you choose an architectural model that requires a FHIR server, this will add a major annual cost to your project. Enterprise servers can easily run to six figures annually. Chapter 8 discusses pricing in more detail.

If your timeline is tight and delivering on time is more important than high-performing APIs, you may need to start with a facade. If time is on your side and the budget allows for it, a hybrid approach may be the better option.

9. What is your organization's tolerance for change?

"Tolerance for change" is hard to quantify. It's a difficult question to answer honestly, especially if you're documenting it in a public Architectural Decision Record (ADR).

Ask yourself this: how often are large projects spanning two or more years approved in your organization? And when they are, how often does leadership reverse course in response to a bad quarter or changes in key personnel?

Decision impact

If your organization is "low-risk" and favors smaller, incremental projects, you may want to start by looking at a simple facade – read-only if possible. Introducing a FHIR server and all the procurement and testing steps this entails should be avoided where possible. If your organization is comfortable with large projects and has a history of "staying the course," then you should have the freedom and confidence to look at more complicated hybrid solutions.

Conclusion

These questions are not theoretical. They determine whether your FHIR project succeeds or struggles from the start.

If you can't answer them clearly, you should hold off on making irreversible decisions until you can. Push back on the business to clarify assumptions, and do not commit to architectural models, FHIR servers or delivery timelines too early.

Most FHIR projects that fail do so because key decisions are made before the fundamentals are properly understood. Do not let that happen to yours.

Hard truths

"Tolerance for change" is a key failure point on large FHIR projects. Leadership gets cold feet eighteen months in and starts demanding shortcuts. I've seen this play out many times. If your organization does not have a track record of delivering high-impact, cross-team projects, do not expect this time to be different.

Chapter 4
The Facade Model in Practice

What is a FHIR facade?

A facade is a FHIR API that draws data from existing systems, dynamically converts it to FHIR resources and returns them to the consumer. It's a translation layer that does not store any data. For consumers, a facade and a FHIR server can look and behave the same. Under the covers they're very different.

The most common implementation of a facade is read-only, with selected resource types and search parameters exposed to consumers. A common misunderstanding when building a facade is that all search parameters and FHIR resource types need to be handled. This is not true. You get to choose which parts of the FHIR API you support.

I've seen facades that expose only a single resource type and a single search parameter, and facades that expose more than thirty resource types and a wide range of search parameters. The bigger the facade, the more work is involved in building it.

Figure 4.1 illustrates a simple read-only facade that exposes patient data. The source of the data is a single SQL database located in an existing system.

A consumer makes a FHIR query to the */Patient* endpoint to get a patient with a specific MRN. The facade validates the search parameters and passes the request to the "SQL Translator," which converts it into an SQL query and runs it against the database. The results are passed to the "SQL to FHIR Mapper," which constructs a FHIR Bundle and returns it to the facade.

This is a simple "happy path" request flow for a read-only facade. To implement this, you need to build variations of the following:

- An API with a /Patient endpoint exposed for GET requests
- An ability to accept any required FHIR parameters ("identifier", for example)
- An SQL translator to transform the request and get the data
- An SQL to FHIR mapper to construct the FHIR resources

A real-world implementation will be more complicated, depending on what data is being requested and where it's located.

The bulk of the work in building a facade lies in the FHIR Mapper that converts SQL (or other) results into FHIR resources. Complexity increases when multiple FHIR resource types are requested and must be retrieved from different systems. The SQL Translator may need to call multiple databases or services to fulfill a single query, and the FHIR Mapper must combine those results into what can become a large FHIR Bundle.

More complicated facades may also support write requests, routing them to different databases, services or APIs and returning the updated results as FHIR resources. As facades grow in scope, they begin to resemble FHIR servers. At this point, moving to a hybrid approach makes a lot of sense.

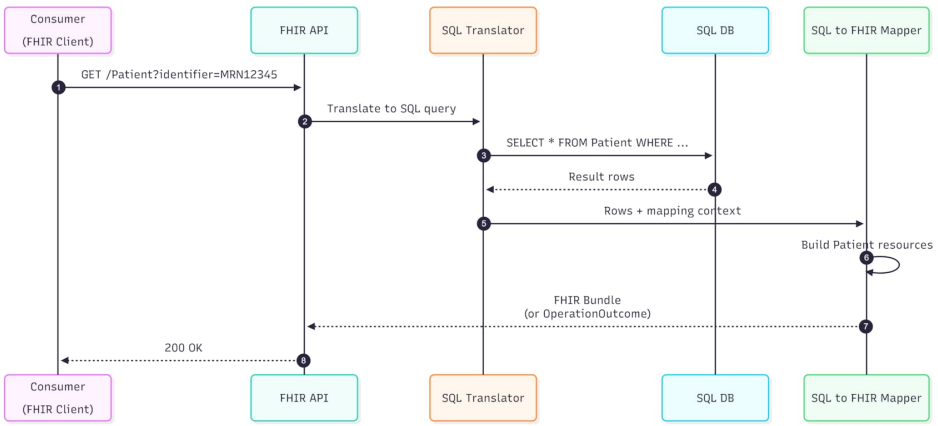

Figure 4.1

Where it fits (and where it doesn't)

A facade is not the right choice for most use cases, but it does work well in narrow, clearly defined situations.

Regulatory requirements

Meeting a specific regulation's requirements is a good reason to look at the facade model. Most regulations focus on consumers being able to read data via FHIR, not write data. They are also very specific about which data needs to be accessible,

while having nothing to say about which search parameters a consumer can use. This makes it possible to fulfill a regulation by handling a small number of resource types and search parameters. Regulations are often vague about performance, which means slower response times may be acceptable initially, especially if a richer hybrid model is planned later.

Interim strategy

A facade can be a good incremental approach when exposing data via FHIR. It allows a technical team to implement a working – if limited – API quickly and expand on it later when resources and time allow. In many cases the facade can be replaced with a hybrid or FHIR-native approach later with minimal impact on consumers.

Simple queries

Facades work best when the FHIR footprint is small – a limited number of resource types and search parameters. One example I mentioned earlier involved exposing only the Observation resource type with a single "encounter" search parameter. This rarely justifies a full FHIR server.

Low volume

Low request volumes are another good reason for choosing a facade. If requests are few and datasets are small, performance issues can be manageable. Handling larger volumes using a facade often involves caching or other architectural changes, which come with extra complexity and risk.

When NOT to use

Handling complicated search queries – especially those that use the "_include" and "_revinclude" parameters – can be difficult and sometimes impossible using a facade. A single FHIR query that spans multiple systems may take many seconds or even minutes to run. This is rarely acceptable.

When performance matters, a facade is rarely a good choice. Dynamically constructing FHIR resources for each request will always be slower than getting those

resources from a FHIR server. Caching can help, but that only works well for simple queries and comes with its own set of difficulties.

Finally, a facade is rarely a strong long-term plan, particularly when FHIR requirements are likely to grow. For most organizations it works best as an interim step.

Boundaries and operational reality

A facade translates FHIR requests into calls to existing systems but does not own any of the data it returns. Data quality, availability and performance all depend on those systems. If they are slow or unreliable, the API will be too.

Every request is a live query. If one system is down or slow, the whole request is affected. This becomes more noticeable as more systems are involved in a single query.

Most of the work is in the mapping layer. Converting source data into FHIR resources is where problems show up, especially with edge cases and terminology. Changes in upstream schemas or APIs can break the facade without warning.

Variants in practice

Facade architectures are often presented as a simple alternative to a FHIR server. In specific cases that simplicity is warranted, but in many implementations the complexity of underlying systems leads to facades that are as complicated as hybrid designs.

Variant A: Read-only facade	
Description	A facade that exposes a read-only FHIR API over one or more source systems.
Source of Truth	Source systems.
FHIR Persistence	None. Resources are constructed dynamically per request.
Read Flow	Client > FHIR API > Facade translator > Source system query > FHIR mapper > Return.
Write Flow	Not supported. Source systems are updated via their own native interfaces.
Typical Use Case	Regulatory consumer requiring read access to a legacy system.
Key Trade-offs	Fastest way to expose legacy data via FHIR, but handling search parameters adds significant complexity.

Table 4.1

Where it fits

A read-only facade can work well for meeting regulatory requirements where the **existence** of the API is the goal rather than its **performance**. Internal systems that can run as nightly batch jobs are also a good fit.

Primary risks

The performance of the existing system determines the performance of the API. If that system is a fast PostgreSQL database, your API will perform well. If that system is multiple poorly performing legacy services or databases, your API may be too slow to function under production loads.

Multi-system complexities

When data needs to be pulled from more than one source system to fulfill a request, potential problems begin to arise. Merging data from multiple sources to construct a single FHIR Bundle can be challenging. Pagination can be difficult and sometimes impossible depending on the FHIR query. Parameters such as "_include" or "_revinclude" may need to be restricted if they need to pull in resource types from different sources.

Performance problems can multiply as the number of data sources increases. A single under-performing database or API can degrade the entire facade.

The permitted FHIR search parameters and modifiers – especially those that affect underlying database joins – may also need to be restricted. A multi-system facade is generally only practical when search queries are simple and data is easy to bring together.

Operational reality

Most problems with facades are performance-related, which means they originate in systems outside the facade's control. For this reason, it's important that source systems are stress-tested in advance of choosing a facade model.

Monitoring production uptime and performance requires monitoring of the source system as well as the facade, which may not always be possible.

The initial workload in building the facade will likely focus on the legacy-to-FHIR data mapping. Key problem areas tend to be around terminology mapping and unexpected edge case data coming from the source system. Ongoing maintenance issues often revolve around changes to the legacy data that have not been communicated with the FHIR team.

Caching is problematic for anything but the simplest facade. Complicated search queries and combinations of multiple search parameters can negate the value of the cache, requiring fresh source queries for most requests.

Variant B: Read / write facade	
Description	A facade that exposes a read and write FHIR API over one or more source systems, translating data to and from FHIR.
Source of Truth	Source systems.
FHIR Persistence	None. Resources are constructed dynamically per read request. Write requests pass data through to the source system.
Read Flow	Client > FHIR API > Facade read translator > Source system query > FHIR mapper > Return.
Write Flow	Client > FHIR API > Facade write translator > Source system update > FHIR mapper > Return.
Typical Use Case	Exposing a legacy system via FHIR without syncing or migrating data. Ex: labs, medications.
Key Trade-offs	Complete FHIR wrapper for a system, but every read and write still depends on the legacy systems.

Table 4.2

Where it fits

Ideal for internal organizational use where read and write access would benefit from a fresh FHIR API, possibly for integrating with new apps and systems. Strong and consistent legacy APIs or easy database access would make integration easier.

Primary risks

The same performance risks as for a read-only facade apply, with extra concerns if write requests are delayed or asynchronous. Write requests may suffer from mapping problems with the source system, especially if FHIR extensions or unexpected terminology are used.

Multi-system complexities

Write requests that span different systems can lead to orphan data and incomplete data if individual system updates fail. The complexity of building a "transaction" that spans systems, when some of those systems are outside your control, can sometimes be impossible to overcome. In these cases you're pushing against the boundaries of facades and should be looking at a FHIR-native approach or maybe restricting write access via FHIR.

Operational reality

A key benefit of a read-only facade is that source systems do not have to change. With write facades this is rarely the case, especially if data is being received from external data providers. Comprehensive mapping from FHIR resources to existing databases or APIs may not succeed without additions to the source systems.

Changes to the facade or the source system can rarely be made in isolation. Both need to be kept in sync to prevent data loss on the way in or gaps in data on the way out.

Write facades may be practical when the data size is small but prove difficult to implement and maintain as the volume of resource types increases.

Implementation approaches

Many FHIR projects start small, usually as a response to individual regulations that affect only subsets of data stored in existing systems. This leads to "quick and dirty" solutions that begin as a custom-made facade. While this is workable, there are alternatives.

Custom facade – build your own

Earlier I mentioned an organization whose sole FHIR requirement was to expose a set of Observations for individual Encounters via FHIR. Here's the query:

/Observation?encounter=Encounter/12345

It was not difficult to implement. A single API endpoint, a single parameter, a single data source. It took a two-week sprint to complete and test.

This is a perfect use case for a custom facade. Requirements were small. There were no unanswered questions. Request volume was anticipated to be low enough that performance was not an issue.

If your own requirements are as small as this, then a custom facade is a viable option. As the FHIR API is the same regardless of implementation, you will not be locked in and can change your approach in the future if requirements change.

Commercial tools

A small number of commercial FHIR servers can function as facades. Instead of syncing data with the server's database, you map your existing data to FHIR resources and elements. The server – configured to act as a facade – accepts the incoming request, retrieves data from existing systems, builds the FHIR resources and returns them to the consumer.

The FHIR API already exists. Your workload involves data mapping and configuration – not implementation.

But this comes with an annual cost and it locks you into a specific vendor's implementation. Moving away from that vendor in the future would likely involve architectural as well as code changes.

Open-source alternatives

A middle ground between a custom solution and a commercial FHIR server is to use an open-source FHIR server as a starting point. The FHIR API is implemented, resource objects exist and hooks to retrieve data from a database are in place.

If you expect to handle a lot of resource types and search parameters, an open-source project can be a good place to start. Adapt it to access your own data sources and rewrite the internal mapping layer to fit your custom data structures.

Hard truths

Facades have their place, but that place is limited. They work best as interim steps, when you have immediate requirements and little time, or when the future role of FHIR in your organization is still unclear. Starting with a facade and evolving towards a hybrid or FHIR-native model is common.

Facades rarely hold up as long-term solutions. As data models and FHIR queries grow more complex, facade implementations become harder to maintain and performance becomes difficult to manage.

Chapter 5
The Hybrid Model in Practice

What is the hybrid model?

A hybrid FHIR system syncs data from existing or legacy systems into a FHIR server. The source of truth remains with the legacy systems, while the FHIR server stores a copy of the data. The sync is typically driven by Change Data Capture (CDC) or event notifications.

Legacy systems continue to run as before, accepting data via existing services, APIs or direct database updates. Consumers access the data via a FHIR API.

In some implementations, the FHIR API may also function as a write facade, exposing write endpoints to data providers and translating and routing the requests into the legacy systems. This is less common and far more difficult to implement.

The legacy systems may store data in a single relational database or in a collection of different databases and data stores. Syncing multi-system data to the FHIR server comes with its own problems, but the consumer is unaware of them. All they need to know is that a FHIR API is available for read requests.

The key feature of hybrid systems is that ownership and management of the data lie with the legacy systems. If your architectural diagram is leaning towards the FHIR server being the source of truth for some resource types, look at the FHIR-native variants in the next chapter.

Figure 5.1 illustrates a simple hybrid system where a single SQL database is synced with a FHIR server. The consumer reads patient data from the FHIR server, unaware of the true source of the data and the syncing process behind it. The diagram gets more complicated where more than one data source is involved.

In the context of FHIR architecture, the word "hybrid" can be ambiguous. It means different things to different people. When discussing hybrid systems, it's important to ensure everyone understands your meaning.

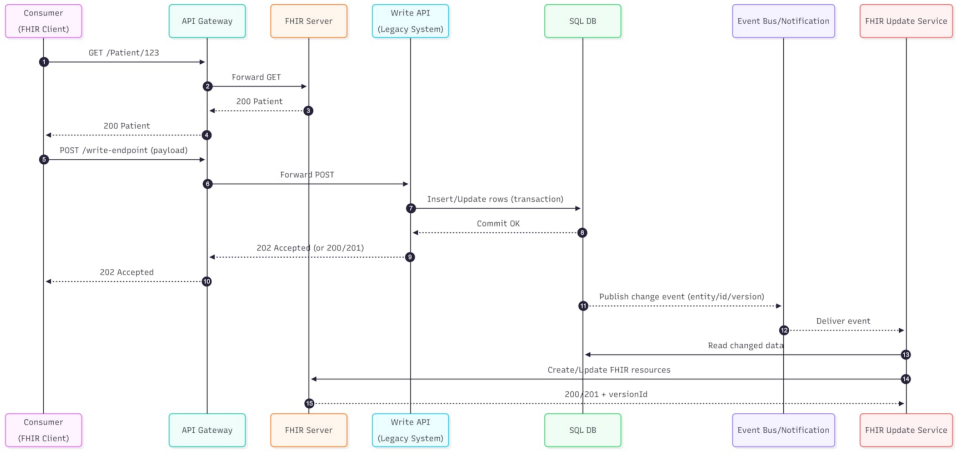

Figure 5.1

Where it fits (and where it doesn't)

Hybrid is a strong fit when your existing systems are staying in place and when you need to deliver a FHIR API to consumers without the performance issues common to facades.

Regulatory requirements

If you are required by regulation to expose some of your data via a FHIR API, hybrid is a good fit. As mentioned in the previous chapter, you could do this quickly using a FHIR facade, but if API performance is important – and it should be – then hybrid is a much better solution. Each request is served directly from the FHIR server, without routing to legacy systems and without dynamically building FHIR resources.

Maintaining existing systems

Unless you're building a truly greenfield platform, your patient and clinical data will already be stored and managed inside existing systems. These systems have evolved over time, and will likely include data governance and master data management. Apps and processes may have been consuming and updating this data for many years. It's the source of truth, and there may be no plan or need to change this. A new FHIR requirement on top of this is well served by a hybrid design.

Incremental modernization

Moving to FHIR piece by piece or domain by domain is common. This often begins with meeting a specific FHIR regulation – pathology data and reports, for example – and then expanding by exposing more domains via FHIR. A hybrid model fits this incremental approach very well, as you can expand out your data syncing steps into more domains as required.

Consistent FHIR API

Onboarding new apps, consumers and data providers to existing systems has always been time-consuming and difficult. Moving to FHIR makes this easier. The FHIR API and FHIR standard replace proprietary APIs and data models. Together, they provide a consistent interface for new consumers and remove the need to learn multiple proprietary APIs.

When NOT to use

If you're building a truly greenfield project that has no legacy dependencies, then hybrid is not a good fit. Otherwise you might find yourself designing a new legacy system, then syncing with a FHIR server as an extra step. FHIR-native may be a better approach as it lets you bypass that legacy step entirely and jump straight to FHIR.

The syncing process of the hybrid model is rarely real-time. Depending on your legacy systems, there could be substantial delays in new or changed data being reflected in the FHIR server. Use cases that require immediate access to changed data are unlikely to be met by a hybrid model.

If you are considering syncing only some resource types to the FHIR server – master data, for example – and making FHIR the source of truth for domain-specific transactional data, this is not hybrid. You should look at one of the FHIR-native variants.

Boundaries and operational reality

Hybrid systems expose data through a FHIR server but do not take ownership of it. Data governance, data quality and master data management remain with the existing systems.

Most of the risk is in the sync process. Delays, failures, upstream schema changes – all show up in the FHIR server as bad data, sometimes without being obvious straight away. Data drift and partial updates are common, so monitoring and correction processes are needed.

Consistency is not guaranteed. Consumers may see old or incomplete data depending on when the last sync ran. The maintenance work is in keeping the sync reliable and dealing with issues when it breaks.

Variants in practice

When deciding between the two variants, the key question to ask is whether your FHIR API is read-only or read/write.

Variant A: Sync-only, read-only FHIR server	
Description	FHIR server populated via CDC or event-driven sync for selected resource types. No write access exposed to external consumers.
Source of Truth	Source system.
FHIR Persistence	Full for synced resource types.
Read Flow	Client > FHIR API > FHIR repository query > Return.
Write Flow	Not supported. Source systems are updated via their own native interfaces.
Typical Use Case	Any legacy system that wants to expose all or some of its data via FHIR.
Key Trade-offs	Simple to implement, but syncing process and legacy-to-FHIR data mapping can introduce difficulties.

Table 5.1

This is the most common implementation of a hybrid FHIR system. In its simplest form, data is synced from a single source system. It gets more difficult when multiple source systems come into play, each providing different data that needs to be synced. Sometimes multiple systems may need to be accessed when creating or updating a single FHIR resource; sometimes there may be inconsistencies or duplicate data across the different systems that need to be resolved as part of the sync process.

Where it fits

This variant works well for read-only use cases where write control rests entirely with the existing system. This includes regulatory consumers and internal systems that need access to structured data. It's also a good fit for organizations planning a wider migration to FHIR over time. A read-only hybrid implementation is a strong first step, allowing data to be exposed via FHIR one system at a time.

Primary risks

Data drift between the FHIR server and the source system is a real possibility, as are sync delays and incomplete data. Schema or terminology changes in the source system might not be reflected in the translation layer of the sync process, leading to sync failures or incorrect data. Consumers can be overconfident in the quality of the data if they are unaware of the issues with the syncing process.

Multi-system complexities

Constructing interconnected FHIR resources from data coming out of more than one source system can be complex. Example: data used to populate Encounter, Observation and Patient resources may legitimately be stored in different databases in the source system. Source of truth may be unclear, and duplication may be common. The mapping component inside the syncing process will need knowledge of all these native dependencies and will need to be able to resolve conflicts and inconsistencies.

Issues arise if one source system is unavailable or if expected connected records are not found across systems. Is a FHIR update not made at all or should a partial update occur? If a partial update is allowed, how is the consumer notified?

Multiple systems often mean multiple identifiers, some or all of which may need to be reflected in FHIR, adding confusion around which system is the true source of truth.

Operational reality

Careful monitoring of errors during the sync process is required, with failures or partial updates reported in sufficient detail that action can be taken to correct them. You will need a manual process for this – especially in the early months after a first production release. This may take the form of a "retry" method for individual updates or screens to manually correct problematic FHIR resource types.

Wider data quality issues are not always noticeable immediately. Incorrect terminology in very specific instances, for example. Fixing these sorts of issues often involves bulk updates, which are not easy to run – if at all – on some FHIR servers.

Consumers may notice asynchronous updates to the FHIR server, and may report missing data issues while waiting for the sync process to kick in. To handle this, expectations need to be managed. For example: SLAs that specify 30-minute windows instead of 30 seconds.

"Temporary fixes" to FHIR data – often required when something goes wrong in production – can put the FHIR server data out of sync with the legacy system's data and can cause problems for updates during the sync process.

Variant B: Sync with write-back facade	
Description	Reads from a synced FHIR server. Writes accepted via FHIR API, translated by a facade, and sent to the legacy system. FHIR server updated asynchronously via CDC or event-driven sync after legacy write completes.
Source of Truth	Source system.
FHIR Persistence	Full for synced resource types.
Read Flow	Client > FHIR API > FHIR repository query > Return.
Write Flow	Client > FHIR API > Facade write translator > Source system update > * > Return. * FHIR mapper may be unable to function here
Typical Use Case	Legacy system that wants full FHIR exposure without needing to change existing systems.
Key Trade-offs	Full FHIR interface for consumers and data providers, but facade translation to native data model and calls to legacy systems can introduce real complexity.

Table 5.2

This variant is not common in practice. Where it is implemented, the write facade tends to be kept small, with only very specific data handled and mapped to the legacy data model. There can be many complications, especially when multiple source systems need to be updated.

Where it fits

Variant B can work for systems that need to accept data via FHIR but do not want to make the FHIR server the source of truth for that data. It can also function well as a "next step" in an incremental approach if migrating to FHIR-native. Start

with a read-only server, later allow selected write requests via the write facade, moving to fully FHIR-native as write requests shift from legacy systems to FHIR. It's also a good fit for organizations whose legacy systems cannot be exposed directly.

Primary risks

There is a real danger of write failures or partial updates across legacy systems, with a knock-on effect on the data in the FHIR server. Race conditions are a possibility where multiple source systems need to be updated and the sync process kicks in too early. There is complexity in mapping incoming FHIR data to legacy schemas and data models, especially around terminology.

Multi-system complexities

Transforming FHIR POST requests into updates that span multiple systems is challenging as true transactions are not possible. If one update fails, how does this affect the remaining and previous updates? A process needs to be put in place to manage this.

The subsequent sync process may be problematic if different systems update in different ways: some synchronously, others asynchronously. A sync could fail if it is triggered before one of the source systems has processed its update.

Operational reality

Ownership of the legacy system's write processes often does not lie with the FHIR project team. This can lead to insufficient access to error logs and reporting when writes fail, and an inability to re-run updates. Rollback or delete endpoints may not be exposed, limiting your ability to undo a failed transaction, leading to partial or orphan data in the FHIR server.

Mapping of incoming FHIR data to existing schemas and terminologies can fail when data providers are out of sync with the latest updates to legacy systems. Handling this requires clear advance warning of any legacy schema changes and an ability to push back on them if data providers cannot make corresponding changes in time.

Implementation approaches

Implementing the hybrid model requires a combination of components that your team builds along with a FHIR server – commercial or open-source.

Build your own

Typical components include the following, each one a significant project in itself:

- Custom sync pipelines using CDC, event notifications, messaging or batch jobs
- Mapping layers to convert legacy data models to FHIR resources (and the reverse if you are supporting a write facade)
- Monitoring and reconciliation processes to manage sync failures, data drift and data quality issues
- A proxy to intercept and route write requests through a reverse-mapping layer to legacy systems

Commercial & open-source servers

Some server vendors offer support with data mapping from legacy systems and have existing modules to integrate with CDC and other data flows. Your choice of vendor may impact the size and scope of what you need to build yourself.

With open-source servers you're very much on your own. An advantage of open-source is that it's easier to fully integrate with a server when you have access to its source code and can make changes.

Chapter 7 discusses the FHIR server decision in more detail.

Hard truths

Variant A is one of the strongest models for exposing data via FHIR. It delivers a fast FHIR API without impacting existing systems, but it comes with an ongoing server cost that often deters organizations from choosing it at the start. Many organizations begin with a facade, encounter performance problems and then move to a hybrid model.

Variant B rarely works well in practice, especially when more than one legacy system needs to be updated. Problems increase during development as the complexities are often underestimated. It's common for organizations to stop, limit or roll back a write facade.

Chapter 6
The FHIR-Native Model in Practice

What is FHIR-native?

In FHIR-native systems, the FHIR server is the primary data store and the source of truth for all or some of the data it contains. All queries are FHIR queries, with no requirement to translate or run them against legacy systems. In practice, it's less common than many assume.

FHIR-native implementations come in two variations. In one, all data is stored and managed in the FHIR server with no external dependencies. In the other, transactional or domain-specific data is created and stored in the FHIR server while other data – such as master data – remains in existing systems and is synced as needed.

If you're unclear whether a system is FHIR-native, ask this: are some resource types created and persisted only in the FHIR server? If they are, the system is FHIR-native. If all data is copied or imported from elsewhere, it's hybrid.

Figure 6.1 illustrates a simple FHIR-native system where all data is persisted in the FHIR server. Search and update requests flow from the consumer through an API Gateway to the server where they are processed. Search queries return resources stored in the FHIR server, and POST requests create new resources inside the same server.

Anyone who has built or maintained a real-world healthcare data system knows that many components are missing from this diagram. In Chapter 3 I talked about the "platform illusion trap," where organizations have expectations of their FHIR server that are not met. Key features such as data governance, MPI, MDM and golden records are often required but do not come with a FHIR server.

Organizations considering FHIR-native need to understand the limitations of FHIR servers and the additional components they will need to build or buy alongside them.

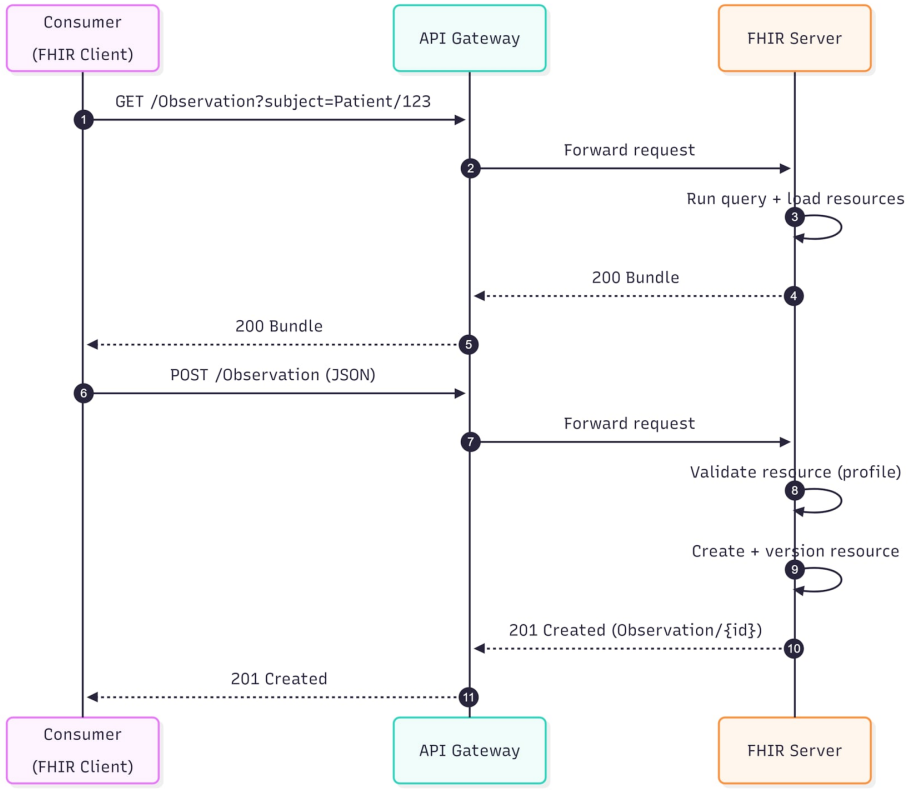

Figure 6.1

FHIR-native examples

Greenfield FHIR-native projects are becoming more common. COVID testing centres are a good example. Many operated independently of legacy systems, managing registration, testing and results without requiring external patient identifiers.

A more complex example is a diagnostic imaging provider, which receives patient and other data from legacy systems. HL7 v2 messages are converted into FHIR resources and stored locally. As scans are completed, transactional data such as Observation, DocumentReference and ImagingStudy resources are created and stored in the FHIR server.

FHIR-native can lead to rapid development, but it can also lead to systems that are as complex as the legacy systems they are designed to replace.

Where it fits (and where it doesn't)

FHIR-native can be a good approach for organizations that want FHIR to be more than a translation and interoperability layer. It requires a shift in thinking, especially for technical teams used to relational databases. The speed of setting up a cloud FHIR server makes choosing FHIR-native easy, but it's not always the right choice. In practice, a number of conditions should be met before FHIR-native makes sense.

Greenfield projects

A new project with no legacy data dependencies or requirements makes FHIR-native a real possibility. Starting with FHIR as the primary data model removes much of the schema design and translation work and introduces standardization from the start.

Platforms that generate their own data

Domain-specific platforms that generate and manage their own data internally can benefit from FHIR-native. Transactional data is created and managed within the platform, while more broadly used resource types are synced from existing systems.

AI, analytics and secondary use

Secondary use of structured data is central to Europe's EHDS regulations. Other use cases include AI and analytics – all of which benefit from FHIR's standardized structure, terminology and Implementation Guides. For projects like these, going FHIR-native reduces the need for translation layers and makes clinical data easier to access for heavy data-centric use cases.

When NOT to use

Organizations that are heavily dependent on legacy systems rarely succeed with FHIR-native approaches. Migration and syncing can quickly become unmanageable and data ownership and governance often becomes unclear.

If FHIR requirements are small – as they often are when the work is regulation driven – going all-in on FHIR-native is overkill. Facade or hybrid approaches are often a better fit.

Boundaries and operational reality

FHIR-native owns and manages clinical and operational data directly as FHIR resources, reducing or removing the reliance on legacy systems and proprietary APIs.

It's often seen as an opportunity to start fresh with few legacy constraints. In practice, misunderstandings about what a FHIR server does and does not do can become costly. Capabilities such as identity management, data governance, terminology services and data correction processes sit outside it. These need to be built or integrated separately. Without them, the system will not hold up under real-world use.

Variants in practice

The key question when designing a FHIR-native system is: where is the source of truth for each FHIR resource type?

Which system maintains the Patient record? Which system creates Observations, MedicationRequests and other transactional resources? Document this for each resource type your project uses. Once you've done this, the variant often becomes obvious.

Variant A: Single source of truth – single FHIR server	
Description	A single FHIR server persisting all data in supported resource types. The main platform for storage, access and updates.
Source of Truth	The FHIR server.
FHIR Persistence	Full.
Read Flow	Client > FHIR API > FHIR repository query > Return.
Write Flow	Client > FHIR API > FHIR repository update > Return.
Typical Use Case	A greenfield project with no data dependencies, or a major modernization of an existing system.
Key Trade-offs	Cleanest and most high-performing FHIR model but rarely achievable where legacy systems remain active.

Table 6.1

Startups in healthcare data often default to a single FHIR server as their primary data store. FHIR-native can seem like the obvious answer when faced with interoperability requirements, and FHIR's resource types offer a comprehensive data model out of the box. It can be hard to say no to, but it's often the wrong answer.

Where it fits

It can work well for greenfield projects. Examples include: a travel vaccine clinic, an online telehealth platform, an airport test center. In each case, a patient record can be created without relying on external systems.

Organizations willing to undertake major rewrites to gain the advantages of FHIR may also benefit, as can platforms expecting significant AI or analytics use. It can also work well as a POC starting point, if the limitations are understood.

Primary risks

Project scope is often underestimated, especially when experience is lacking on the technical and product teams. Entire sub-projects get missed. (see Appendix I)

When legacy systems maintain ownership of some data, the source of truth can become unclear, as multiple systems maintain overlapping data.

Operational reality

FHIR API limitations often come as a surprise, especially to those who assumed FHIR would deliver all the strengths of a relational database. These limitations can lead to the implementation of custom FHIR operations to fill gaps in functionality and to parallel projects to sync data from the FHIR server to data warehouses where it can be queried more aggressively.

Problems specific to individual FHIR servers are common. Examples are poor bulk data import capabilities, server performance and concurrency issues, and cost overruns as request loads increase. This reinforces the need to choose the right FHIR server at the start.

While this variant may be the correct decision, you need to be sure you're choosing it for the right reasons. If your team has never built a healthcare data project before, you likely don't have that experience.

Variant B: Mixed source of truth – single FHIR server	
Description	A FHIR server owns and persists some data while master data is synced from external systems.
Source of Truth	The FHIR server for transactional data. External systems for master data.
FHIR Persistence	Transactional data is fully persisted. A copy of master data is persisted.
Read Flow	Client > FHIR API > FHIR repository query > Return.
Write Flow	**Transactional data:** Client > FHIR API > FHIR repository update > Return. **Master data:** External System > CDC/Message feed > FHIR API > FHIR repository update
Typical Use Case	Domain-specific systems such as imaging, lab or procedure platforms that receive ADT or ORM messages and generate new clinical data locally.
Key Trade-offs	High performance, but introduces complexity around data syncing and data governance boundaries.

Table 6.2

This is the most common FHIR-native implementation. The source of truth for master data is a system that already exists. The FHIR server is the source of truth for transactional data such as Observations, Claims and Immunizations.

Where it fits

This variant is a good fit for systems that own domain-specific data but do not own or control master data. They rely on that master data in order to trigger their own workflows, but all data created as part of those workflows belongs to them.

Some examples:

- A glucose monitoring platform that generates large volumes of Observations while relying on identified patients from an EHR.
- An imaging or radiology solution that documents scans for existing patients, creating FHIR resources such as ImagingStudy, DiagnosticReport, Observation and DocumentReference, while maintaining a copy of Patient, Practitioner and ServiceRequest resources supplied by an existing system.
- Care management platforms that generate CarePlans and associated resources for referred patients and send reports to existing practitioners.

These examples illustrate how the FHIR-native system owns some of the data but not all. The hospital system owns the patient record, not the FHIR server that makes use of the Patient resource.

This variant also fits modernization programs where processes and systems are being migrated domain by domain. Master data systems are often the last to migrate to FHIR.

Primary risks

Confusion over data ownership is a common problem, with data being changed in the FHIR system to meet immediate requirements instead of being changed in the legacy system. This sometimes leads to backdoor or jury-rigged solutions which cause data drift and confusion across systems.

Operational reality

Data syncing is never perfect. CDC, event notifications, HL7 v2 messages – even in the best of implementations these systems are never bulletproof. Problems occur that need to be fixed, sometimes in a hurry with a patient on the table. Processes and screens need to be built so that data can be corrected and conflicts resolved.

Clear data governance rules should address this but often do not. Insisting that master data FHIR resources be read-only is often impossible in practice. While

early versions may assume a "happy path," production realities quickly lead to workarounds and raise the need for making these resource types less read-only than they were expected to be.

There often comes a time when two-way syncing of master data is required.

Variant C: Distributed source of truth – multiple FHIR servers	
Description	A FHIR platform that spans a number of servers, divided by resource type.
Source of Truth	Each FHIR server is the source of truth for its assigned resource types.
FHIR Persistence	Fully persisted. Different resource types in different FHIR servers.
Read Flow	Client > FHIR API > FHIR repository query > Return.
Write Flow	Client > FHIR API > FHIR repository update > Return. [Cross-server transactions may require splitting and merging of request payloads and results]
Typical Use Case	Systems with large data volumes that need different performance and governance rules depending on resource type.
Key Trade-offs	Enables scaling and governance by domain, but comes with routing complexity and cross-server query, transaction and data sync challenges.

Table 6.3

This variant is most commonly implemented as a separation between master data and transactional data, each managed in different FHIR servers. This is similar to variant B, but replaces the legacy system with a second FHIR server.

Assume two FHIR servers. Server 1 contains master data (Patient, Practitioner, Organization). Server 2 contains transactional data (everything else). Updates to master data on server 1 are synced to server 2 via event notifications. This approach allows for the separation of master data into its own system and fast querying of all data on server 2. Write requests to master data resources on server 2 are usually restricted except during syncing and data reconciliation processes.

In larger systems or national programs, separation may be along clinical domain lines, with master data synchronizing across multiple servers.

Where it fits

Large organizations where patient and practitioner identity is managed separately from daily transactional data, and where different data governance rules apply to different data domains. This separation justifies the overhead of running multiple FHIR servers.

Primary risks

If data is synced from one server to another – as is the case in the master data example – it will suffer from the same sync issues and the same ownership confusion as variant B.

Operational reality

Different FHIR server vendors handle this in different ways, or not at all. Some recommend multiple FHIR servers, others lean towards partitioning of the underlying database, while others maintain a single FHIR API that accesses multiple underlying databases. Expect a higher cost, as most commercial server vendors charge by production instance.

As with variant A, the master data server will require many extra processes and systems to be installed, built or configured for proper patient and identity management. The workload here should not be discounted.

If you're looking beyond a two-server "master data/other data" split, think long and hard about the trade-offs. Depending on the request complexity and how resource types are separated across servers, you may need to impose restrictions on

the FHIR API, not allowing certain parameters to be used at all. Restricting FHIR search parameters due to architectural decisions negates much of the power of FHIR. Is it worth it in order to separate data by domain? I don't believe it is, but this is open to debate.

Implementation approaches

If legacy systems exist containing master data, consider variant B as a starting point. This allows you to maintain existing governance and master data processes and focus on building the transactional systems. As those systems mature and confidence in the FHIR-native approach grows, ownership of master data can be moved to its own FHIR server, removing the reliance on legacy systems.

Coordination layer

With two or more servers, things can get complicated quickly.

You may need a proxy layer in front of your FHIR API, intercepting requests on the way in and responses on the way out. This allows you to route traffic to a specific FHIR server depending on the request, and it's a good place to trigger data governance processes.

You need an effective strategy for syncing data from the master server to the transaction server. This may be an event-driven process, with changes queued for processing in the transaction server, or you might decide to use FHIR Subscriptions as the trigger for change requests.

In systems where multiple domain servers exist, cross-server requests can be problematic. You may need to implement a query management step to split queries and route them to specific FHIR servers, then combine the results as they come in. The proxy mentioned earlier is a good place to initiate this.

Commercial & open-source servers

Chapter 8 focuses on the FHIR server decision, including the differences between commercial and open-source servers.

Hard truths

FHIR-native is a lot of work. It requires the same skills, expertise and experience needed to build and maintain legacy systems. Too many organizations forget this and assume all the old experience is no longer necessary. If you and your team have never built an enterprise scale healthcare data project before, FHIR-native should not be your first choice. If you don't understand why this is the case, look at Appendix I.

Chapter 7
Selecting the Right FHIR Architecture

After reading the last three chapters, you've probably made up your mind. You understand your requirements, your team's capabilities and how things get done in your organization. One model should have risen to the top. That doesn't mean it's the best model, but it may be the best for your project right now.

Mapping requirements to models

If you answered the nine questions in Chapter 3, you should be able to identify the model that fits your project. **Table 7.1** maps those questions to models and highlights which is the best fit.

Strong fit: recommended for this scenario.

Viable: acceptable but with limitations.

Weak fit: not recommended.

Question	Facade	Hybrid	FHIR-native
Q1: Why are you exposing data via FHIR? Assumed: regulatory or interoperability requirement	Viable	**Strong fit**	Weak fit
Q2: Are you publishing data or building a source of truth? Assumed: publishing existing data	**Strong fit**	**Strong fit**	Weak fit
Q3: Where do write requests land? Assumed: write requests go to legacy systems	**Strong fit**	**Strong fit**	Weak fit
Q4: Sharing existing data or generating new data? Assumed: significant existing data in legacy systems	Weak fit	**Strong fit**	Weak fit
Q5: Who are your consumers? Assumed: ex-	Weak fit	**Strong fit**	**Strong fit**

Question	Facade	Hybrid	FHIR-native
ternal consumers with performance expectations			
Q6: What are the project deal breakers? Not applicable.			
Q7: What can your technical team realistically deliver? Assumed: limited FHIR experience, moderate technical capability	**Strong fit**	Viable	Weak fit
Q8: What are your budget and timeline constraints? Assumed: moderate budget and time to deliver a mid-sized project	**Strong fit**	Viable	Weak fit
Q9: What is your organization's tolerance for change? Assumed: moderate, with some	**Strong fit**	**Strong fit**	Weak fit

Question	Facade	Hybrid	FHIR-native
appetite for multi-year projects			

Table 7.1

A warning based on experience: don't bite off more than you can chew. There is a workable migration path from facade to hybrid to FHIR-native, but you can't easily move in the other direction. Choose a model that makes sense today and that you can build on over time. All three models work. All three come with limitations and implementation headaches.

Moving between models

You may have to choose a less-preferred model. Financial, time and other constraints often necessitate quick delivery, forcing you to ship a working system today instead of the best system tomorrow. In these situations you may have approval in principle to improve the system later once more resources become available.

This often means starting with a facade and moving to hybrid over time, accepting the performance problems that come with it.

Migration is less common than you might expect, and almost always comes later than it should. Once a system is in place and working, it can be difficult to convince the business to allow a rewrite, even if problems are occurring.

Facade to hybrid

If your facade is implemented well and is fully aligned with the FHIR standard, migrating to hybrid should have little or no impact on consumers. A hybrid FHIR server can be installed and pre-loaded with data while the facade remains in place. Once syncing between legacy systems and the server is working, you route all FHIR requests away from the facade to the FHIR server. Ensuring that an API Gateway sits in front of your facade from the start should make the migration process seamless.

Hybrid to FHIR-native

This is more complicated. Moving to FHIR-native means your hybrid system becomes the source of truth for some or all data it contains. It may assume ownership of master data and the processes around it. Migration can be a single step or a phased implementation. You may decide to maintain write requests in legacy formats initially, intercepting incoming requests and transforming them to FHIR. This would reduce the workload on data providers and keep all changes in-house. Migration could occur one system at a time, with master data systems and all of their surrounding complexity left until the end.

FHIR-native to other models

Moving away from FHIR-native is far more complicated than moving in the other direction. You would need to introduce new APIs and databases – essentially building an entire system from scratch – and then layer your FHIR facade or hybrid system in front of it. This is not a direction to consider and is one of the reasons you must be as confident as possible if you choose FHIR-native as your architectural model.

When you can't make the decision yet

You may not have all the information required to make an informed decision, but you might need to get started regardless. Deadlines exist. Your team may be in place and eager to get started.

Whichever model you decide on, you will be returning data to consumers in FHIR format. If legacy data models exist, you will have to transform those models into FHIR resources. You can start this work immediately.

If you're not in a position to settle on an architectural approach, you're not in a position to choose a FHIR server either. You may not even need one. Your teams can get started by using Docker instances of the HAPI open-source server, and by beginning to work with a FHIR SDK. The FHIR API and the resource type objects are the same regardless of model and server provider. This work is required re-

gardless of the model you choose. Once you have determined which model you will be using, switching approaches should be straightforward.

Do not be rushed into choosing a model when you're not ready. The cost of a wrong decision increases over time.

Why teams choose the wrong model

Lack of FHIR knowledge leads to poor decisions.

It's common for teams to choose FHIR-native when hybrid is a better fit for their project. They discount the complexity of existing systems and think they can easily move everything to FHIR. Technical people love working on pure greenfield projects, and sometimes go out of their way to ensure new projects are as greenfield as possible.

The facade model is often chosen because it's fast and cheap. Teams with little FHIR experience can put a limited FHIR API in place and meet immediate requirements. These systems can then fall over under even moderate production loads. The rush to meet regulations can lead to APIs that are not fit for purpose. This is far more common than many realize.

Teams often choose hybrid without understanding the complexity involved. New systems get released to production without data correction abilities, leading to data drift between the two systems. Overconfidence in syncing processes leads to data that is not trusted by consumers.

Don't be afraid to start small and build up your FHIR system over time.

Choosing the model

You've made the decision.

Eliminate the other models and move forward. Keeping options open at this stage slows everything down. Document what you will build and what you will not. Ensure the wider organization understands the limitations of your selection: what can't be done, what will be slow, what resources will be required to build and maintain it.

Expect trade-offs and imperfections, especially when legacy systems bump up against your FHIR system. Data transformations may be imperfect, terminologies may not align fully. Expect this and allow for it.

Start the FHIR server selection process. You don't need to choose a vendor yet, but you should settle on a server category so you know what you'll need to build yourself and what may come with the server. Chapter 8 goes into this in detail.

Hard truths

The strongest implementations I have seen are hybrid, where the source of truth rests with the legacy systems and write requests flow directly into those systems. You do not want to assume responsibility for the data governance complexities of a master system unless you have to.

Facades that become hybrid systems over time are common in theory and in documentation, but are rare in practice. If you're a technical architect reading this book, you will know from experience that tomorrow may never come and you may be stuck with what you build today for many years.

Data syncing is never as perfect as we expect it to be. There is always a manual process required to fix incorrect data or syncing failures. I've seen systems released without this that came apart within weeks because data issues couldn't be fixed easily. If this is not on your roadmap, it needs to be.

Chapter 8
Choosing a FHIR Server

Once you've chosen an architectural model, you can choose a server.

The wrong FHIR server decision can set your project back by many months. If the mistake is not discovered until after you release to production, fixing it may take years. For many organizations, the server decision is made at the worst possible time – early in the project, when the technical team has the least knowledge of FHIR and when the wider project requirements are not always properly understood.

Chapter 3 focuses on understanding your own requirements. In that chapter I addressed nine questions that you need to be able to answer before making key decisions like choosing a FHIR server and vendor. If you still can't answer those questions, you should not be making this decision.

Before looking at the factors that may influence your decision, I want to highlight something I've experienced many times across organizations: choosing a FHIR server for the wrong reasons.

How NOT to choose a FHIR server

The default decision

For the most part this chapter is not going to address vendors by name, but I am going to make an exception here to highlight a common problem when organizations choose a server provider. I call it the "existing platform problem" and it looks like this.

An organization is heavily invested in one of the big cloud platforms like Microsoft Azure, AWS or Google Cloud. All or most of their infrastructure is hosted on this platform – databases, web apps, identity management, Kubernetes clusters, AI and machine learning. There may be a company policy that says if the

platform provides a service, it must be used. This applies to FHIR servers just as it does to any other service.

The server choice then becomes a box-ticking exercise. Other vendors may be looked at half-heartedly, but the decision is already made. The Architectural Decision Record (ADR) documenting the decision will probably focus on the "easy integration" between the FHIR server and existing apps and services on the same platform, ignoring other key requirements.

Let me be clear: this is the worst way to choose a FHIR server.

There are real differences between servers. Making a default decision for something so crucial is a mistake. If a policy like this exists in your organization, you need to push back on it and insist that you have greater autonomy when choosing a FHIR server.

Vendor publicity and marketing

A small set of vendors are over-represented in FHIR server discussions. They attend – and in some cases run – many of the public FHIR events around the world, and they are highly visible in Google search results, LinkedIn posts and the FHIR forums. These are good companies delivering strong products, but this does not mean they should automatically be at the top of your shortlist. Good marketing is not one of YOUR requirements.

In Appendix II, I provide a list of vendors sorted by category. This list will likely include companies you are not familiar with or were not aware operate in the FHIR server space. Once you've made decisions around which category of server you need, this list should help you filter the vendors down to a shortlist.

The choices available

Server vendors broadly fall into four categories. If you're clear on your requirements, you'll likely narrow this to one or two categories. There is some crossover between them, usually where managed server providers also sell containerized instances of their servers for you to manage locally. Most servers are similar in what they support. The differences that matter are how much control you have over the

server, what comes bundled beyond the core FHIR functionality and how tightly the vendor's architecture integrates with and affects your own.

Your goal is to eliminate the wrong categories quickly.

1. Managed, cloud servers

The vendor owns and manages the server and infrastructure. Spinning up a new server can usually be done in under an hour. This category features brand-name companies such as Microsoft and AWS, as well as a number of smaller companies and startups. There is wide variety in functionality among these providers, with many of the smaller companies showing more innovation than their larger counterparts, rolling out feature-rich updates regularly.

Where the vendor runs large cloud platforms, there is often a path to integrate the FHIR server deeper into their wider infrastructure. Examples are low-code tools, data warehouses and Terraform scripts.

When something goes wrong at 2am, it may be caught by the vendor's monitoring tools. But you will still need your own uptime monitoring to ensure data is flowing and accessible.

Pricing

Pricing tends to follow the pattern for cloud services: a mix of fixed hourly charges and request volume and storage size charges. Unless you have a good understanding of your data and request volumes, this can lead to higher bills in production than expected. Price hikes and pricing structure changes do happen. I know of one business who had to change server providers on the back of pricing changes as their model was no longer viable. You need accurate data projections and confidence that the price won't change dramatically if you're going with a cloud provider.

Weaknesses

There is often little choice in data location – unless you bypass the managed aspect and run a containerized instance of the server yourself (not always possible). Earlier I gave an example of an Azure server in Europe being hosted in Sweden. Expect similar limitations from all managed, cloud servers.

New releases tend to be rolled out across the board. While you may receive notification of an upcoming release along with a list of changes and bug fixes, you won't have control over whether or when it happens.

When this is the right choice

When you're looking for a FHIR server your team can start working with straight away with no long term commitments. When you do not want responsibility for maintaining the server and managing updates. When you want out-of-the-box integration with existing platform services.

2. Enterprise FHIR servers (self-hosted or vendor-managed)

Some of the most high-profile names in the FHIR space fall into this category, and many of them have been around since the early days of FHIR. I use the word "Enterprise" here in the software sales context. For these vendors, the FHIR server is the core product.

The key differentiator of servers in this category is that you configure and manage them yourself, either on-prem or in your own cloud infrastructure. The servers are generally more feature-rich than their managed counterparts, as they tend to be the primary product of the companies involved, not one product of many.

When something goes wrong at 2am, your technical team is responsible for identifying the problem, fixing it, or rolling back to a more stable version.

Pricing

Expect the sales process to span many months, involve multiple meetings and possibly a paid-for POC. Servers are generally sold per production instance, with support and possibly consulting sold as part of the package. Unlike managed servers, they tend to be priced annually, with pricing starting in the high five figures or low six figures. You won't find prices on their websites and probably will not hear a figure until you're well into the sales process.

Weaknesses

Your technical team will be responsible for maintaining the server once it goes live. Whether this is a strength or weakness depends on the capabilities of your team. Ownership of rolling out updates is a positive, but only if you have the internal resources to test and then release new versions.

The additional functionality that enterprise servers offer beyond the core FHIR spec can create dependencies that make it difficult to migrate to a new server provider in the future. Your architecture becomes tightly entwined with the unique server architecture, making partial rewrites necessary if you decide to move.

Expect a slower start compared to managed solutions.

When this is the right choice

When you need full control over the configuration, performance and deployment of the server. When you need flexibility in your architecture. When your DevOps and infrastructure teams can manage complex production systems.

3. End-to-end healthcare platforms with FHIR capability

This category brings together vendors in the healthcare data space who bundle a FHIR server as part of a much larger package. They sell end-to-end platforms, often incorporating HL7 v2 and DICOM data flows, mapping and conversion to and from FHIR, and downstream transformation to analytical formats such as OMOP. Many have built-in features for data governance, EMPI and master data management.

When something goes wrong at 2am, it should be handled by the vendor's operations team.

Pricing

Expect high licensing costs and long contracts. These platforms are not cheap. But you're getting a lot more than just a FHIR server.

Weaknesses

Vendor lock-in is significant. You're committing much of your data flow to systems built and managed by another company. Migrating in the future would be a major undertaking.

When this is the right choice

When you need more than just a FHIR server. When you want a fully integrated solution for data flows and integrations. When your requirements are large and your technical teams are not set up to deliver them. And when you have deep pockets.

4. Open-source FHIR servers

National and regional programs have been built using open-source FHIR servers. They are not a lesser alternative. At last count there were eight fully-featured open-source servers, all with permissive licenses and available in different programming languages. Making use of one of these servers gives you complete control over configuration and customization. You can use it as is or extend it to meet your exact requirements.

When something goes wrong at 2am, it falls to your technical team to fix it. While there may be some help available in online forums, and there are consultants who work with these servers, don't expect them to be on call.

Pricing

The server is free, but you should expect significant setup and running costs. Budget for the infrastructure to run the server, a technical team to maintain and enhance it, and possibly some expert consultants on retainer – at least until you build up experience in-house.

Weaknesses

It's common for organizations to misunderstand the work involved in configuring and maintaining open-source servers.

A number of "extras" that sometimes come with commercial servers may need to be built by your own teams – many of these are projects in themselves. See Appendix I for a non-exhaustive list.

There is no guarantee that open-source servers will be maintained long-term. A server used by multiple organizations was abandoned by its maintainer a few years ago. One of the few open-source DICOM servers was also shut down last year, though the same company still maintains an open-source FHIR server – for now.

When this is the right choice

When cost is the primary decision maker. When you have strong technical teams capable of supporting the server, even if it is no longer maintained or updated by the owners. When you need full flexibility in how the server runs.

Organizational and delivery constraints

Team capabilities

The capabilities of your technical teams have a large role to play in choosing a server category. Are your DevOps people capable of managing a FHIR server 24/7? Do you have architects and developers who can build a system around a local FHIR server, and build services that interact with the server?

A strong technical team makes open-source and enterprise servers a viable choice. If your teams are weak or non-existent you should be looking at a managed server, or even an end-to-end solution – if your budget allows.

Tolerance for change

Is your organization willing to invest resources and time to manage a FHIR server, and to extend it when necessary? Regulations change, consumer expectations change, AI requirements change. Your FHIR server and its surrounding functionality will likely have to change to keep up. This requires ongoing resources and dedicated teams.

Low organizational tolerance for change means you should be looking at a managed server, possibly with one of the smaller vendors who are known for moving

fast and releasing new features frequently. If your organization is comfortable with large projects that span many years, an enterprise server may be more appropriate.

Delivery times

An immediate requirement can force your hand and rule out certain categories. Managed cloud servers and open-source are the fastest to get up and running, but you have to accept their limitations. Enterprise servers come with long sales processes and a more regimented POC. Expect a few months before you can even get started.

Budget

If your budget is small, open-source may look attractive – but only if you have the team to support it. Allocating a strong dev team with architects and product people for the duration of the project can quickly get expensive.

Managed servers – with their "pay as you go" pricing models and lack of long-term contracts – can look appealing on paper and prove cost effective if the requirement is short-term or likely to be deprecated in the future.

Enterprise servers and end-to-end platforms are generally more expensive – at least initially – and come with long contracts and commitments.

Data location

Where the data can and cannot be located is often a major factor in the server decision. National regulations and customer expectations both come into play.

On-prem vs cloud

Despite what some suggest, storing patient data in the cloud is not a battle that has been won. Many hospitals – even in 2026 – will still not sign-off on patient data residing anywhere but a VM owned and controlled by their IT teams. Expect variation here across different countries. This is why it's so important to have early insight into who your customers will be in a few years' time.

If you go all in on a cloud solution and later find it's not acceptable to a large portion of your target market, you are facing a major rewrite.

An on-prem requirement rules out most managed, cloud providers. Some of the small vendors may allow containerized deployments for on-prem use, but these may be missing plug-in components and often come with reduced support.

Enterprise servers excel in on-prem solutions. This is one of their key strengths, and it may turn out to be your only choice.

Specific country

European patient data stored on US servers can fall under US legal jurisdiction. This is a scenario that keeps many European data leaders up at night. Different countries have different regulations and expectations regarding where patient data can be stored. I mentioned earlier how Azure FHIR servers in Europe default to Sweden today. This may or may not be acceptable. It comes back to knowing who your customers are and what their requirements are.

It's not just where data is stored, but who controls it and who has legal access to it.

Country-specific data requirements can rule out most cloud providers.

Existing infrastructure

At the beginning of this chapter, I warned against making a default decision on server selection due to existing platform and infrastructure commitments. While the warning is valid, sometimes easy integration with existing apps and processes is more important than the FHIR server itself. You may have apps built on the Microsoft Power Platform that want internal access to a FHIR server, or you may want seamless integration with Google's BigQuery data warehouse. In these cases, choosing a specific platform because of your existing relationship may be a valid choice. But be careful not to downplay the server's limitations.

Architectural constraints

By now you should have decided between a facade, hybrid or FHIR-native architectural approach. How does this influence server selection?

Facade

In most cases no FHIR server is required.

A facade's read requirements can often be met via a custom API built in-house. A small number of servers in the enterprise category can function as a facade, mapping and transforming data from relational databases to FHIR resources. If your facade spans a large or complex data model, or you lack confidence in your technical team, this is worth exploring.

Hybrid

All stand-alone FHIR servers can fill the server role in hybrid designs. The only category that should be excluded here is end-to-end platforms, where you're buying an entire system, not just a FHIR server. Expect to front-load a lot of data with hybrid systems. This means server performance and bulk data ingestion become important.

FHIR-native

With FHIR-native, everything flows through the server. You need a server that delivers strong performance and is operationally easy to manage. Expect to enhance the server a lot over time, adding new operations, processes and transformers. You need as much control over the server and its environment as possible.

I recommend an enterprise server. If you decide on a managed server, you may quickly find yourself bumping up against functional and architectural limitations.

Performance

Once you've narrowed your options, performance becomes a deciding factor.

It's difficult to find accurate metrics that help you compare one server to another. Where data does exist, it is often out of date, focused on a small number of vendor comparisons or clearly biased in favor of one vendor. Example: Azure vs Google from 2020. This data has no value today.

This is why you need a POC. Create a test suite that reflects real-world scenarios, pay the money for the POC, and run your tests. You need to compare like-to-like, so ensure the exact same tests are run against each server on your shortlist. Use realistic data volumes and structures. Simplified datasets will hide real issues.

If vendors respond to shortcomings with promises to fix things later, don't believe them. You're buying a server for what it does today, not what it might do in the future.

What doesn't matter as much as you think

Vendors are quick to hype up specific features of their servers – even if those features are shared by every other server provider. Here are examples of things that don't matter as much as you think.

- Feature checklists. Most FHIR servers support almost all of the FHIR spec. Not supporting something usually happens because it's not heavily used. An exception is FHIR Subscriptions – if you need this, make sure it's supported.
- "*We support Regulation X.*" A very common sales pitch, but meaningless. The server validates and stores what you send it. Meeting the regulations is on you.
- Specific FHIR operations. There are real differences between servers regarding built-in support for operations. Examples are common ValueSet operations such as $expand. This should not be a deciding factor, as most common operations are relatively easy to build. Don't spend $$$ to save yourself $.
- Built-in extras such as MDM, analytics, golden records. Unless you're going all-in with an expensive end-to-end platform, you should not be tying yourself to one vendor's implementation of key functionality like this. Source these externally from dedicated providers.
- Polished demos and white papers. These are marketing tools. They demonstrate happy paths and rarely show up server deficiencies. Ask for a POC instead.
- Performance claims, especially when they come with vendor comparisons. Vendors compare the best of themselves against the worst of others. Unless the comparison is unbiased and recent, ignore it.

Conclusion

The "best" server is the one your team can run, support and afford over time.

Ask yourself how much control and responsibility you want over the server. With managed FHIR servers, you have none; with open-source you have it all. With enterprise servers you have some, but is it enough?

Lastly, do not make this decision without a POC that tests production loads and real-world data models. Make the decision based on your requirements and constraints, not vendor claims.

Hard truths

The best server for your project is unlikely to be the one with the biggest brand behind it, especially when it's not the vendor's primary product. Many of the smaller server companies are more innovative and address bugs faster than their larger competitors.

If you're considering an open-source server, HAPI should be at the top of your list. It's been around a long time, it's well maintained and it's backed by a commercial company that upsells consulting and other services. It's low-risk. Open-source projects are abandoned or closed all the time – even when they're owned and run by large companies. Priorities shift and directions change quarter to quarter. Don't be seduced by a big name attached to an open-source FHIR server. Stability is more important.

Startups fail often. Be careful investing your data's future with a company that doesn't have a history of stability or growth. A quick way to check for this is to look up the company's page on LinkedIn, navigate to the "Insights" tab and look at the "Employee count" over time. It's a premium feature, but it gives you real data about the state of the company.

Finally, get references. Don't settle for vendor-approved references, reach out yourself or have your team do it. You can find lead architects working with specific platforms via LinkedIn. If things have gone well or badly, they will be surprisingly honest about it.

Conclusion

FHIR is fifteen years old, but it's only beginning to reach architectural maturity. Until recently, implementations were driven by regulations. Organizations moved to FHIR because they had to, not because they wanted to. There were no "best practices" for building complex FHIR systems. Everyone made it up as they went along.

AI and its need for structured data is changing that. More and more organizations are turning to FHIR because they want to. Detailed Implementation Guides are being adopted because they improve data quality, not because they are mandated. Architectural approaches are becoming more standardized as delivering quality FHIR-enabled systems becomes more important than merely ticking a FHIR checkbox.

Variations of the architectural models outlined in this guide are in use all around the world, from startups to national programs. We know what works, where the problems and pitfalls lie, and which model best suits which use case.

It's important to understand the trade-offs in each model. Choose the one that works for your project and organization, but don't underestimate the difficulties and complexities involved. Be aware that a FHIR server is only the beginning. It's one part of your architecture, and not the most important part. Understand where ownership of the data lies and where the source of truth is for each FHIR resource type.

The success of your system will depend far more on these decisions than on the vendor you choose.

About the Author

Darren Devitt

I'm an independent healthcare technology advisor specializing in FHIR architecture and strategy. I've spent six years working on FHIR projects across a range of organizations and sectors, from large enterprise implementations to national programs. Prior to this, I spent twenty years on large-scale data and integration projects.

I'm 100% independent and have no commercial relationship with any vendor or company you work with or depend on. My advice is unbiased. I've worked on FHIR projects for companies like Olympus, Optum and BCG.

That work – designing and delivering large-scale FHIR systems in real environments, where decisions have real consequences – is the basis for everything in this guide. The advice here is not theoretical. It comes from projects that succeeded and projects that failed.

If you're working on a FHIR project and need senior architectural or strategic guidance, I'm available for hire. Engagements range from a focused piece of work to longer-term advisory.

darrendevitt.com

Appendix I
Implementation Components Beyond the Server

A FHIR server is only one part of a working system. The components below are commonly required to handle real patient and clinical data in production. Whether you need them depends on which architectural model you're using. A facade may require a few, hybrid more and FHIR-native most of them. It's not an exhaustive list. Many of these are underestimated or discovered late in projects.

Access and security

1. **API gateway**
   Handles routing, rate limiting and sometimes authentication and authorization. Almost always required and typically bought (cloud or commercial).
2. **Identity and access management**
   Manages user and application authentication using OAuth2 and SMART on FHIR. Usually required and integrated from existing platforms. Not always straightforward.
3. **Consent and access control (advanced)**
   Manages patient consent, and access rules that derive from consent. Often required and typically built in-house.
4. **Security labels / attribute-based access control (ABAC)**
   Enables fine-grained data access decisions at the element level. Advanced requirement and usually implemented with custom logic. Some server vendors have done work in this area.
5. **Multi-tenancy management**
   Separates data between customers or organizations. Sometimes supported by server vendors in unique ways, but usually requires significant custom code and architectural work.

Data movement and transformation

1. **Data mapping layer**
   Translates between existing data models and FHIR resources. Essential for facade and hybrid, and almost always built in-house. Commercial and open-source libraries may help, but expect a lot of customization.
2. **Proxy / intercept layer**
   Intercepts and modifies requests and responses (examples: validation, transformation, routing, de-identification). Typically built in-house or implemented via FHIR server add-ons.
3. **Data sync / ingestion pipelines (CDC, events, batch)**
   Moves data from source systems into the FHIR server. Central to hybrid architectures and typically built using messaging or ETL tools alongside custom transformation code.
4. **Event bus / messaging infrastructure**
   Enables asynchronous communication between systems (examples: Kafka, Service Bus). Central to hybrid architectures and usually bought or integrated.
5. **Data correction interfaces**
   Enables manual review and correction of data issues when automated processes fail. Always required in production systems and typically built in-house.
6. **Error handling and retry mechanisms**
   Handles failed writes, retries and partial processing. Essential in production systems and typically built into pipelines.
7. **Bulk data processing (import / export)**
   Handles large-scale data ingestion and export. Frequently required and often beyond server capabilities. Expect a combination of server functionality and custom code

Data quality and governance

1. **Data reconciliation and drift detection**
   Identifies mismatches between source systems and FHIR data over time. Critical in hybrid systems and some FHIR-native systems. Usually custom-built.
2. **Data quality and validation layer**
   Enforces business rules beyond FHIR validation. Usually built using rules engines or custom logic, and often requires a proxy layer.
3. **Master patient index (MPI / EMPI)**
   Identifies and links patient records across systems to reduce duplicates. Commonly required for source of truth systems and usually bought or heavily customized.
4. **Master data management (MDM)**
   Manages golden records and resolves conflicting data across domains. Often required and typically implemented using commercial tools. Some FHIR server vendors are moving into this space.
5. **Data governance framework**
   Defines who owns the data, who can change it and what the rules are for resolving conflicts and correcting errors. Always required. Primarily a "people and processes" problem with tools to support them.
6. **Provenance tracking**
   Captures where data came from and how it was created or modified. Important for audit, trust and multi-source systems. Usually involves custom code.

FHIR-specific infrastructure

1. **Terminology service**
   Manages CodeSystems, ValueSets and terminology validation. Often required and commonly bought or integrated. Usually missed in early product releases.

2. **Implementation Guide management**
   Management process for necessary Implementation Guides (profiles, extensions, terminology bindings). Requires internal ownership, though some servers make it easier.
3. **Custom operations**
   Implements non-standard FHIR operations and business logic. A common requirement and typically built in-house. Some server vendors have specific ways of doing this.

Operations and observability

1. **Audit logging and traceability**
   Records who accessed or changed data and when. Required for compliance and typically built using platform logging infrastructure or in-house.
2. **Monitoring and alerting**
   Detects system failures, uptime issues and data flow problems. Always required and usually implemented with existing tools.

Secondary use and testing

1. **De-identification / anonymization services**
   Removes or masks sensitive data for secondary use. Often required and implemented using specialized tools or custom code.
2. **Data transformation for analytics (examples: OMOP, data warehouse sync)**
   Moves FHIR data into analytical models and data warehouses. A common requirement in larger systems. Some vendors have pre-built ways of doing this, others require custom implementations or use of open-source libraries.
3. **Test data generation and synthetic data tools**
   Generates realistic datasets for development and testing. Useful for

early-stage projects. Open-source tools exist to help, but it's typically built in-house due to unique data requirements.

Appendix II
FHIR Server Vendor Landscape

The list below is a snapshot of vendors who provide FHIR servers. It was current as of March 2026, and includes all the providers I'm aware of along with open-source FHIR servers. There is some duplication across categories, where vendors do not neatly fit into a single group.

I don't endorse any of the companies listed and cannot vouch for them individually. Due diligence is required. Do your homework, insist on a POC and seek out references.

Managed, cloud servers

1. **Microsoft – Azure Health Data Services**
   https://azure.microsoft.com/en-us/products/health-data-services
2. **Google – Healthcare API**
   https://cloud.google.com/healthcare-api
3. **AWS – HealthLake**
   https://aws.amazon.com/healthlake
4. **Health Samurai – Aidbox**
   https://www.health-samurai.io/aidbox
5. **Medplum**
   https://www.medplum.com/
6. **Oystehr**
   https://oystehr.com/
7. **SAP – Health Data Services**
   https://www.sap.com/products/technology-platform/health-data-services-for-fhir.html
8. **Aigilx Health**
   https://aigilxhealth.com/managed-ai-powered-fhir-server/

9. **Better**
   https://www.better.care/operational-data-repository/

Enterprise servers

1. **Smile Digital Health**
   https://www.smiledigitalhealth.com/
2. **Firely**
   https://fire.ly/products/firely-server/
3. **InterSystems – IRIS for Health**
   https://www.intersystems.com/resources/intersystems-fhir-server/
4. **Edenlab – Kodjin**
   https://kodjin.com/
5. **Health Samurai – Aidbox**
   https://www.health-samurai.io/aidbox
6. **Medplum**
   https://www.medplum.com/
7. **Blacktusk**
   https://blacktusk.eu/en/b2bProducts
8. **Helios**
   https://heliossoftware.com/helios-fhir-server/
9. **Carefluence**
   https://carefluence.com/products/fhirone-platform/
10. **Exafluence**
    https://fhir.exafluence.com/
11. **Trifork**
    https://trifork.com/work/digital-health/trifork-health-platform/
12. **Pine IT**
    https://www.pineit.at/produkte

End-to-end solutions

1. **InterSystems – IRIS for Health**
   https://www.intersystems.com/resources/intersystems-fhir-server/
2. **Infor – Cloverleaf**
   https://www.infor.com/resources/cloverleaf-fhir-server
3. **Enovacom**
   https://www.enovacom.com/en/solution/enovacom-data-repository-the-fhir-data-warehouse-to-enhance-your-data/

Open-source servers

1. **HAPI**
   https://github.com/hapifhir/hapi-fhir
2. **Medplum**
   https://www.medplum.com/
3. **Microsoft**
   https://github.com/microsoft/fhir-server
4. **Blaze**
   https://samply.github.io/blaze/
5. **Spark**
   https://github.com/FirelyTeam/spark
6. **Helix**
   https://github.com/icanbwell/fhir-server
7. **Ballerina**
   https://github.com/ballerina-platform/module-ballerinax-health.fhir.r4
8. **DirectFHIR**
   https://github.com/metacareforyou/directfhir

Final Note

When large FHIR projects fail or run into difficulty, the root cause can usually be traced back to decisions made in the first few months. The wrong architectural approach, the wrong FHIR server, not enough attention paid to what sits beyond the FHIR server.

I'd love to say we're getting better at all this, but my experience over the past few years is that we're not. Companies who should know better, who've been in healthcare for decades, make the same mistakes as startups.

If you need help making these early decisions, get in touch and we'll see if we can work together to make your project a success.

darrendevitt.com