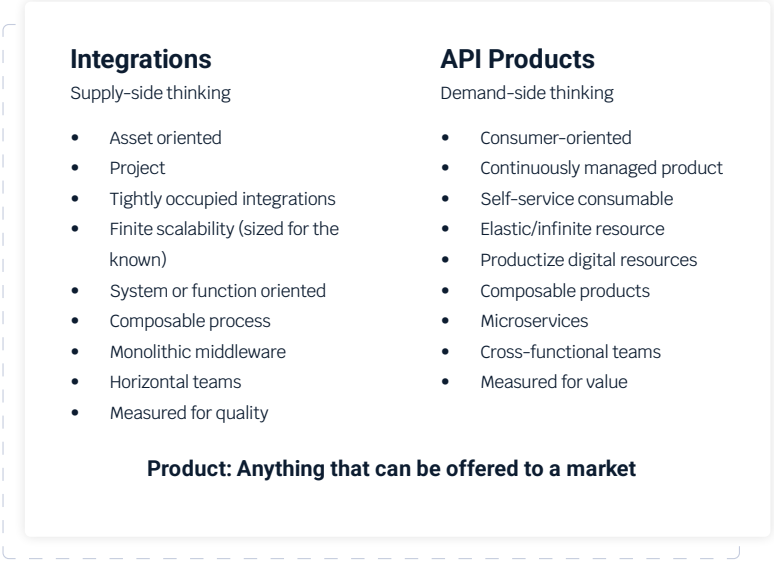

OpenLegacy ebook
Automate Digital Service Creation from Legacy Systems
From integration mindset to product mindset with an API factory

Connecting legacy systems to digital services raises a number of challenges. Unfortunately, enterprises often create even more unnecessary obstacles by failing to approach it with the right mindset. You need to support the agile and swift creation of APIs to meet every business use case. This ebook examines the benefits and best practices of pursuing an "API product mindset" through the creation of an API factory.

Enterprises approach APIs the wrong way

Enterprises are far too slow at generating critical APIs connecting digital services to legacy systems, mainly because they persist in treating APIs as assets rather than products. An asset-based mindset focuses on the supply and internal systems side and inherently limits growth. Product-based thinking focuses on the demand and consumer side to deliver increased value.

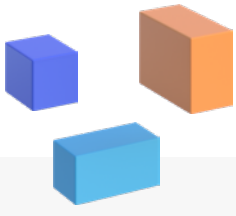
Companies should approach APIs as a product, asking themselves customer-centric questions like "what do users need?" rather than asset-centric questions like "What can I do with the resources at hand?



An asset primarily benefits business operations, but it might not drive profit in any direct manner.

APIs allow various computer programs to use each other's capabilities, forming the bedrock underlying all digital services. Consumers take for granted services like checking your bank balance on your phone, or comparing quotes from different insurance companies.

But generating APIs is easier said than done. It's a problem that touches technology, company culture, and collaboration between teams.



Too many teams drag creativity down

If you talk to a digital services developer about mainframes, AS400, or other legacy systems, they wouldn't know how to program or access data from these systems. This leaves them unable to access content from legacy systems. If you talk to a legacy developer about programming microservices, they'd look at you funny.

You need people with both these skill sets, coming from different backgrounds and generations and working in tandem, but that doesn't always happen. Different teams have differing priorities, and people, processes, and tools can form severe bottlenecks.

It gets worse. Most processes require an integration team between the other two. The integration team creates a facade for data translation or updates ESB middleware layers. Even if the legacy team identifies the right areas in the system quickly, the extra handovers and intervening changes add a great deal of time and inefficiency to the process. Frequently, the number of people who understand the middleware is limited, and they are busy, so schedules get lengthened.

All of these hand-offs increase the time to move from idea to fulfillment, and slows down your creation of digital offerings. Sluggish delivery handicaps your organization's creativity at birth. An awareness of the many months of headaches it takes to deploy a new service prevents teams from even thinking "Hey, wouldn't it be great to offer consumers a new banking service?"

> *Generating APIs touches technology, company culture, and collaboration between teams.*

Speeding up the process requires a change of approach

The most efficient way to speed up digital creation is by applying automation while decoupling the digital from legacy teams. This way the two separate teams can work in parallel, each doing what they do best. The automation allows custom code generation for each legacy asset, so there isn't a need to build a facade or use an ESB.

This whole process increases the speed of creation without requiring the integration team. Building assets quickly shifts your thinking from an asset-centric to a customer-centric model, allowing you to consider adding digital services more easily, just as you would any other product to better serve your customers.

Four modernization strategies

Before we delve deeper into the twin concepts of automation and parallel working, let's review the typical modernization strategies available to enterprises.

Rehost

This option moves your legacy system to the cloud, but keeps it running as-is. The shift to the cloud doesn't impact any of your API connections, so your assets would remain in different formats and languages. IT teams still have to expose APIs before your digital teams can make use of them. This also requires specialist employees to service your legacy systems, which can be expensive and time-consuming.

Rewrite, refactor, and reengineer

Businesses commonly take this set of approaches when they need to transition to the cloud. To do this successfully, you'll need to migrate your legacy functionality to modern applications in a gradual manner with a phased migration strategy. First you'll need to provide value by delivering APIs and extended functionality, while using migration methodologies behind the scenes to rewrite monolithic applications as microservices.

This can help reduce risk, but it adds its own obstacles. Providing access to data is difficult when some of your apps are in the cloud, and others are on your on-prem servers. Rewriting your legacy systems takes a while, but everyone needs access to your apps and data immediately. Plus, you still have to choose the technology you wish to use for it, such as microservices or serverless functions.

Replace

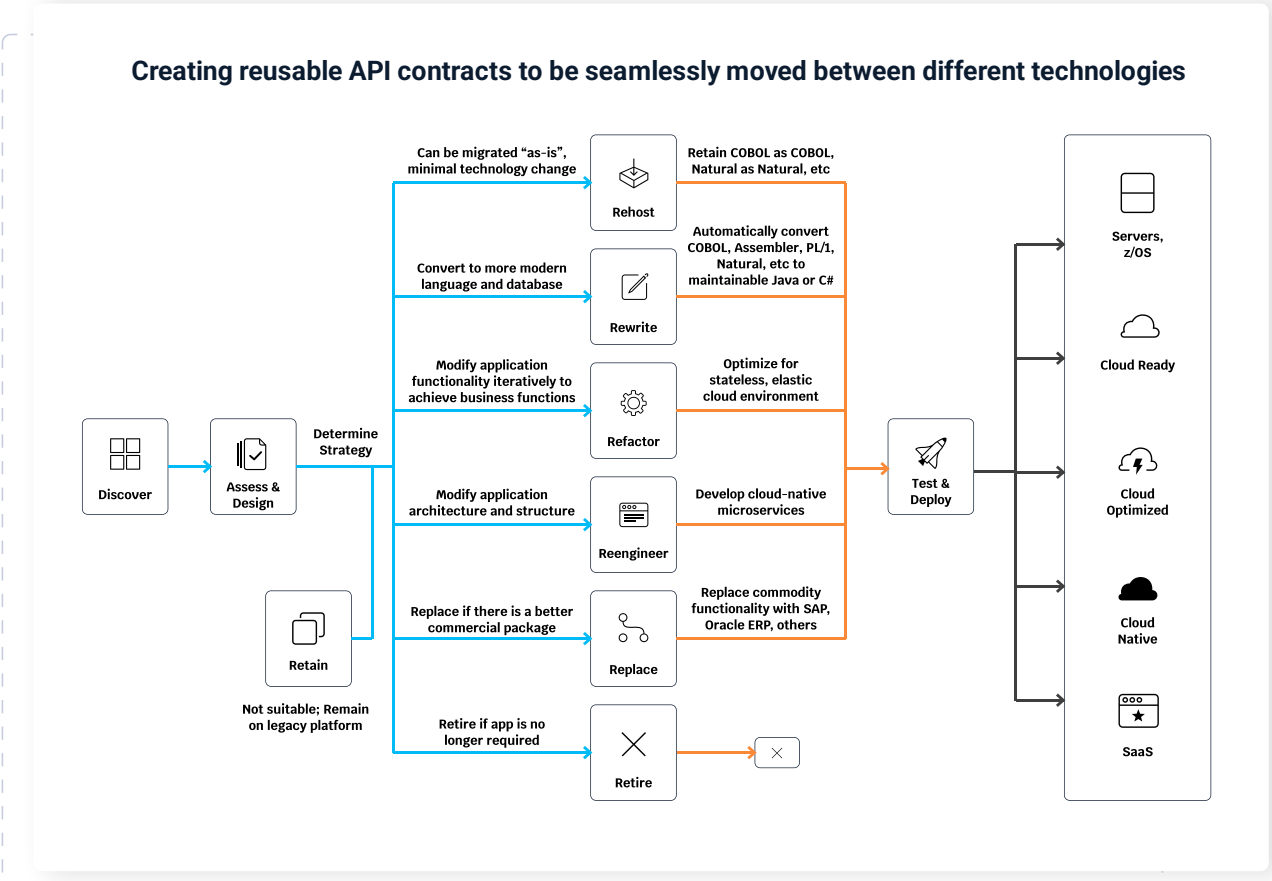
The next option is to replace all the third-party software that's running in your legacy systems with new, digital infrastructure. However, it can take years to evaluate your needs, research options, purchase a new set of tools, and implement them into your ecosystem, and you don't have years. Your employees and customers need data access right now.

Plus, you would probably find that your old and new systems need to coexist for quite some time while you complete the process. You'll still have to work out how to make connections with your legacy systems today while you're researching their replacement.

It's preferable to build the tools your users need, and then map any missing parts from third-party products back into the system.

Retire

Often, businesses consider retiring an app that's still in use, even if it's only by a handful of customers. This raises the challenge of continuing to meet their needs without the burden of outdated, legacy infrastructure. For example, many companies have custom or third-party apps that they need to rewrite and replace. It's common to resolve this by building APIs to support their short term needs while deciding the best long-term solution.



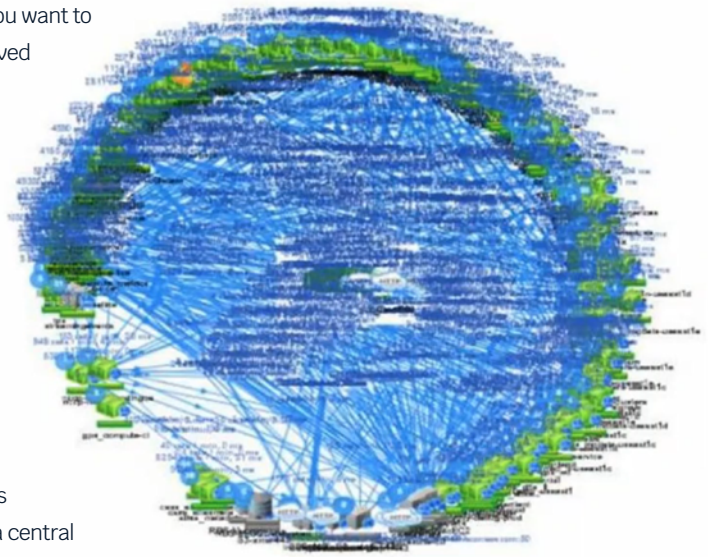
All these cases share common drawbacks that can be addressed by an API Factory mindset. An API Factory approach helps you complete your most pressing APIs quickly so you can move on with your strategies.

The benefits of an API factory mindset

An API factory mindset brings two main benefits: automation and speed. Ideally, you want to produce APIs on demand, as you see a consumer need, and that can only be achieved through automation.

We refer to it as a "factory" mindset because the process should be as automated as a car factory. Car factories produce cars constantly in an automated process, even when they switch between models. An API factory needs a fully-automated software factory for creating requirements, APIs, testing, deployment, and iterations, all with access to legacy-based systems.

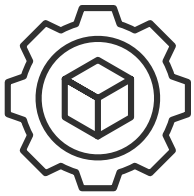
An automated approach dramatically reduces the time, cost, and effort required for digital transformations by making the many handoffs between the steps of the development process frictionless. These include the need to: design, build, test, develop, modify, ensure governance, verify security, establish the network, carry out QC, deploy, etc. It also helps remove human bottlenecks from the process by automating their tasks. Anyone with specialized knowledge can transfer it into a central repository. This prevents waiting until that individual is free every time you need their knowledge. You can automate even highly complex tasks, like building a complex microservice architecture and ensuring that it all works together correctly.



Netflix's highly complex microservices architecture or service mesh.

> *An automated approach dramatically reduces the cost, time and effort required for digital transformations by making the many handoffs between the steps of the development process frictionless.*

However, if you stick to supply-side thinking, you're liable to miss out on business opportunities. An automated, fast-moving API factory mindset enables you to create APIs the moment your customers need them, rather than developing assets and nurturing them slowly over time.



How to build an API factory

To shift your API creation process to an iterative, flexible API mindset, there are certain practices you need to introduce into your organization.

Catalog your assets

The legacy team is involved in an ongoing project of cataloging existing assets in the legacy system. This is best practice, to end up with a full understanding of all the assets, including databases, applications, transactions, and screens, as well as which programs reside on each one. If the team receives an urgent demand for data from a legacy asset, they can shortcut to finding and creating that module, and then map the information afterwards.

Even with multiple backend systems, a unified repository of JSON-based modules enables everything to look the same for your digital team. They won't have to worry about how to connect to mainframes, Oracle, VSAM, SAP, and more, because the process is automated for each one.

As your legacy team exposes and maps assets into a central repository, it gives digital teams the freedom to pull the relevant asset to generate a digital service. Digital teams gain the capability to create APIs in response to customer demand, without needing to make a specific request from the legacy team each time.

An API factory allows you to tap into the magical flexibility of APIs. Your developers can start creating digital services easily and with simplicity, whenever they are needed, by utilizing the artifacts in the repository or sending a request to the legacy team.

Collect your tribal knowledge

At the same time, you need to collect all your company's "tribal knowledge" into a single archive alongside your mapped asset base. Your legacy programmers make up an older generation of employees who are beginning to retire, but the business logic residing in the legacy systems isn't going anywhere, so who will remain to answer all your questions?

There are a myriad of connections and masses of business logic built into legacy systems: documents related to each program; which fields a program contains; relevant test data for each item; etc. By collecting the knowledge of these legacy experts, you won't be left high and dry when you need it at a later date and you won't have to bother the legacy experts each time you need something. This includes documentation that is gathered and collected together with the legacy information.

Store test data

As the legacy team exposes legacy assets, they are automatically tested and the results stored together with the documentation and the modules. This will go a long way towards enabling test-driven design in the future, when a digital developer comes to use it. Legacy asset code can change even after exposure, so periodic testing helps ensure that all assets are ready to be used by the digital team at will. Equally, it permits the legacy team to fix issues at their leisure, since everything is catalogued and stored.

The ease of use of an API factory

An API factory allows you to tap into the magical flexibility of APIs. Your digital developers can start creating digital services easily and simply, whenever they are needed, by using the legacy artifacts in the repository or sending an urgent request to the legacy team. Let's look at some of the elements you'll need besides the legacy content in the repository.

Templates help you meet corporate standards

When developers generate code, there's a risk that it won't meet corporate standards, such as placing brackets below the line, etc. The advantage of templates is to standardize code generation so it meets company requirements every time. The template is part of the generation process, so every microservice or API uses the template or set of templates and has the necessary corporate customizations.

For example, if your company requires the use of a particular security tool, a template ensures each new API calls the security tool as part of its standard behavior. Although developers could write the code directly into the digital service, they would do so individually for every single microservice. This doesn't just add time, it also increases the chances of manual coding errors or alternatives. Using templates means they won't have to rewrite the API code in the future if they need regeneration for any reason.

Templates ease the process of maintenance too, if you wish to make any changes to APIs in the future. There's no need to rewrite any code or check modifications; you can just regenerate and go. The more you can automate, the faster you can churn out high quality APIs for digital services.

Building using test-driven design

When you import legacy assets, you should test them to make sure the asset you've imported into the repository meets your needs. You'd generally begin by passing some data first to make sure that it works, before you start generating any code from the legacy assets. Test-driven development is known to reduce bugs and increase overall software and code quality.

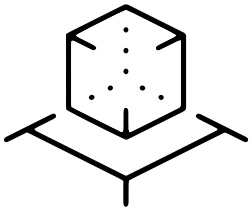
"Test early and often" is an important mantra, but it requires you to write the tests before you've even started writing any code. How can the digital team do that if they don't have extensive knowledge of the legacy systems they're connecting with? By drawing on the test routines and test data that are stored in your repository and bundled together with the legacy programs.

Turning legacy logic into a model allows you to run tests from it, so that you can produce JSON tests to verify the connection to the legacy system before the code is written, and then also check that your connection to the legacy systems is working the way you intended.

If you do this the old way using middleware, tests have to run through the middleware, which means everything needs development before testing. If the test fails after you've already put in all the work to prepare the system, you'll have to figure out where exactly in all the layers something went wrong, and that, in turn, requires checking with all the teams involved. Needless to say, this is a very time-consuming process, so it's valuable to test as you go along, to ensure that you've got the code you need.

Turning legacy logic into a model allows you to run tests from it, so that you can produce JSON tests to verify the connection to the legacy system before the code is written, and then also check that your connection to the legacy systems is working the way you intended.

Legacy teams may generate dozens, or even hundreds, of legacy modules in parallel with the digital team. Six months later, the digital team may pluck any one of those files out of the collection to use in generating a digital service, but they want to be sure it still works with the current legacy system. When you store tests together with the service, each service can automatically be tested periodically using tools like Jenkins. This way, you'll be certain it works when you pull the service, and you have the tests on hand if you want to cross-check. It means the legacy team can test frequently, allowing them to catch and fix problems before they reach the digital team, thereby boosting performance.



Retain flexibility for your high-level digital strategy

Finally, you'll need to decide how to deploy your API factory and how to connect it with your digital strategy, which can change over time as new technologies get introduced or business/technology needs change.

CIOs are rightly concerned with choosing the wrong technology. Tech moves very fast and tech stacks can change swiftly. In the time it takes to develop your digital service, the market may have changed. For example, solutions like Docker and Cloud Foundry are taking the digital world by storm at the moment, but the market has a way of changing direction unexpectedly. You need an API factory with enough flexibility to grant you the agility to change course at will.

Templates help you meet corporate standards

You may need to accommodate several different strategies in parallel, depending on your users' needs. These can include:

- Regulatory needs that demand a particular type of API.
- The need to expose an API as data in your own company, according to a microservices strategy.
- Choosing to go serverless if you need to deploy to DynamoDB (AWS), for example.

It's a strategic decision for your company.

> *Choose a universal approach that can pivot to work with any deployment environment, and be prepared to change to a different deployment one if necessary.*

Adapt to changes in digital strategy

With an API factory approach, you can transfer the necessary assets from legacy into the repository, store them in standard JSON, and then worry about digital strategy later. If your digital strategy changes for any reason, you can adapt to it by simply regenerating the code for a different deployment environment.

Once your legacy information is archived in a repository together with tested artifacts, it's easy to adjust your digital strategy as changes occur, such as if your company is acquired, your business strategy alters, you partner with a different company, etc.

Adapt to different deployment environments and tools

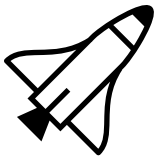
It's also advisable to avoid betting everything on a single deployment environment. You should keep your options open so that the way you deploy the solution is the last item on the agenda, and should be easily switchable. You don't want to reengineer the entire stack to generate your solution.

You should also make sure that you can develop in whatever modern tools you want, such as Java, javascript, serverless, microservices, Kubernetes, docker, openshift, kafka, Node.JS, .NET, etc.

The bottom line is that you need a universal approach that can pivot to work with any deployment environment, and be prepared to change to a different one if necessary. When you have a comprehensive repository of all these assets, it becomes easy to generate whatever you need, whenever you need it.

Overcome culture clash

Producing APIs requires close cooperation between both legacy and digital teams, but their cultures and methods of working are very different, making it difficult to support an integrated legacy-digital team. An API factory allows the two teams to work in parallel rather than in partnership, removing the need to overcome culture clashes and conflicts of mindset. As each team pursues their own workflows, they still support speedy, connected API development.



Automating digital service creation is the way forward

In summary, it's clear that switching to a factory mindset for generating digital services from your legacy systems will help your enterprise become more efficient, more productive, and more customer-oriented. When you make this shift, you'll split legacy coding from digital coding, so your two teams can work in parallel without wasting time on back-and-forth handovers or needing to rely on clumsy middleware.

Additionally, an API factory mindset future-proofs your legacy systems, since it saves your crucial expert knowledge even if the legacy experts themselves retire. You'll easily deploy and redeploy your digital services in different environments, regardless of any changes to your digital strategy.

About OpenLegacy Hub

OpenLegacy is an award-winning software company specializing in rapid, seamless integration solutions. Our digital-driven integration enables organizations with legacy systems to release new digital services faster and more efficiently than ever before. It connects directly to even the most complex legacy systems, bypassing the need for extra layers of technology.

The innovative **OpenLegacy Hub** lets organizations graduate from ad hoc APIs to be able to use and reuse any legacy module for any new project, in any standard deployment environment. The Hub preserves legacy wisdom while enabling giant steps forward in digital transformation. With OpenLegacy Hub, you can adopt an API factory mindset, releasing new apps, features, and updates in days instead of months. This lets you truly become digital to the core.

Contact us

We'd love to answer any of your questions about OpenLegacy Hub or chat with you about your digital transformation. Drop us a line at hub@openlegacy.com or check out our web site.