By this point in the conference you've probably had a pretty full day of maths and code, so I'm going to change the pace a little bit and not give you any more of either of those.

Instead, I want to present to you an idea. An idea that might at first seem like it's not achievable, but which I'd love you to try, because it's an incredibly powerful tool to have at your disposal.

—

LICENSE

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
http://creativecommons.org/licenses/by-sa/4.0/

To start off though, I should probably tell you a little bit about myself.

I build bridges.

Not the real kind, but rather bridges between data science and IT or software engineering.

I've been working in technology for over 25 years, and for about the last 20, I've been building bridges between these sometime disparate communities.
I started out building bridges from IT to data teams, but ended up flipping that on it's head and helping data teams bridge outwards into the IT and software engineering spaces in order to deliver their vision.

That's why I want to talk about APIs today. Data science APIs are the perfect manifestation of this bridge building as they have elements of infrastructure, software engineering and data science.

Photo by Cody Hiscox https://unsplash.com/@codyhiscox

WHAT IS AN API?

sellorm.com

So, lets start off with a little level-setting and make sure we all know what an API even is.

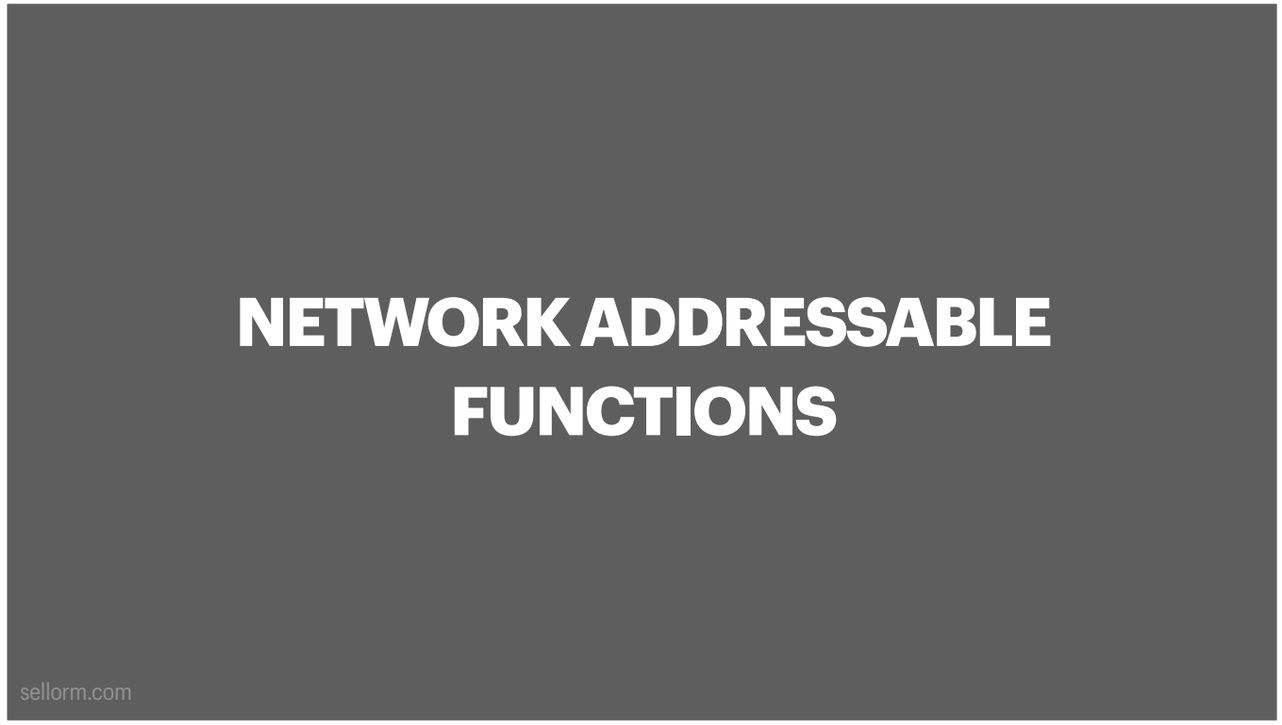
TIDYVERSE:

The 'tidyverse' is a set of packages that work in harmony because they share common data representations and '**API**' design.

— Hadley Wickham

sellorm.com

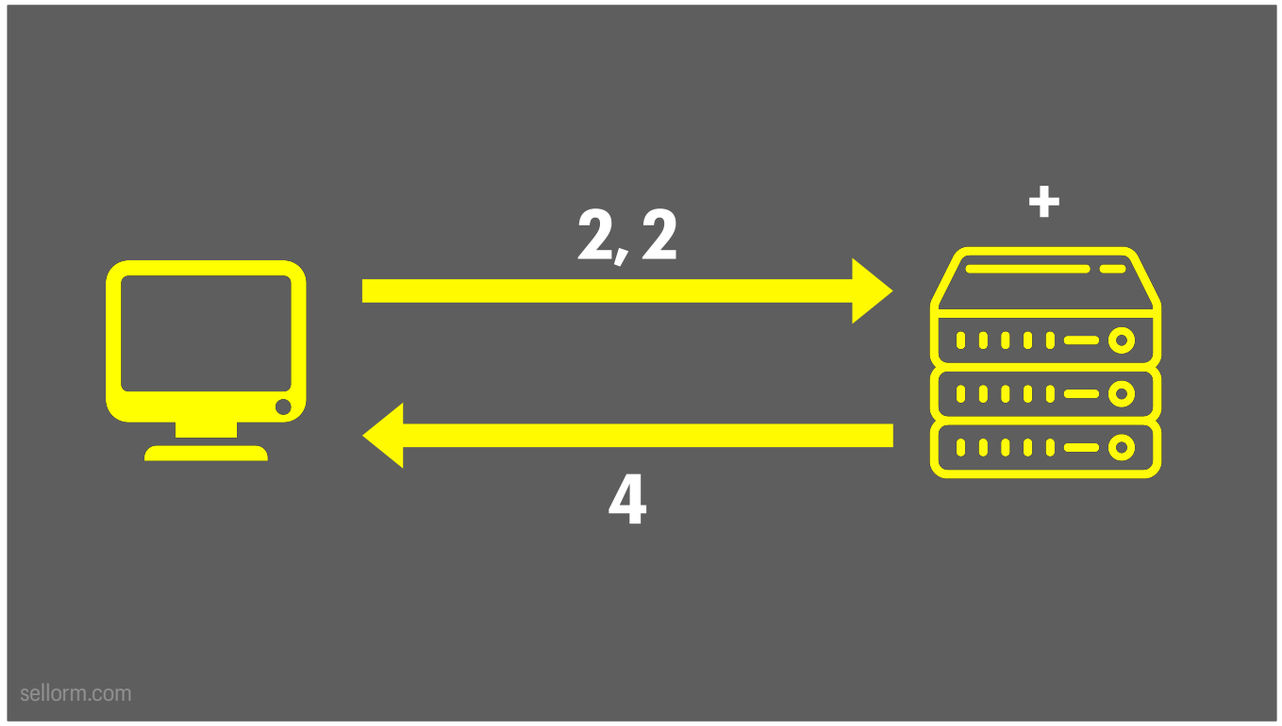
This quote from R's tidyverse package description by Hadley Wickham hints at the overarching definition of API. It's essentially the "design" of the user (developer) interactions within the software. There is however, another definition that's become increasingly common over the last decade.



The broader definition that we discussed is the overarching definition, but the APIs I'd like to talk about today are actually a subset of that definition.

The APIs that I'm going to talk about today are essential network addressable functions.

Literally, functions that can be called over the network.



In this toy example above, the left hand system — a client or consumer of the API — sends two numbers to the “+” API — network addressable function — on a remote system. That remote system computes the answer and sends that as a response back to the consumer.

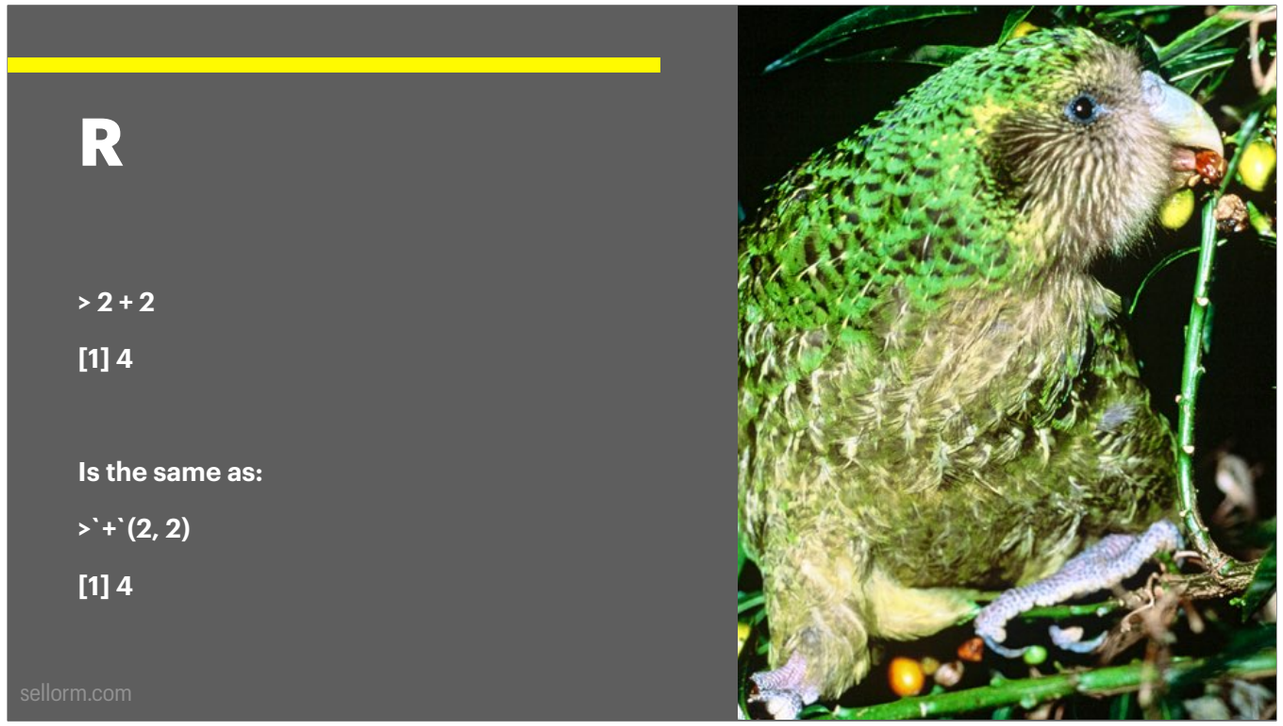
Consumer -> Input -> Function -> Output

Just as a little aside, Infix operators like “+” can in most cases, be expressed as an equivalent function.

In Python, the equivalent is “operator.add”.

The snake in this picture is not even a Python, but rather an emerald tree boa

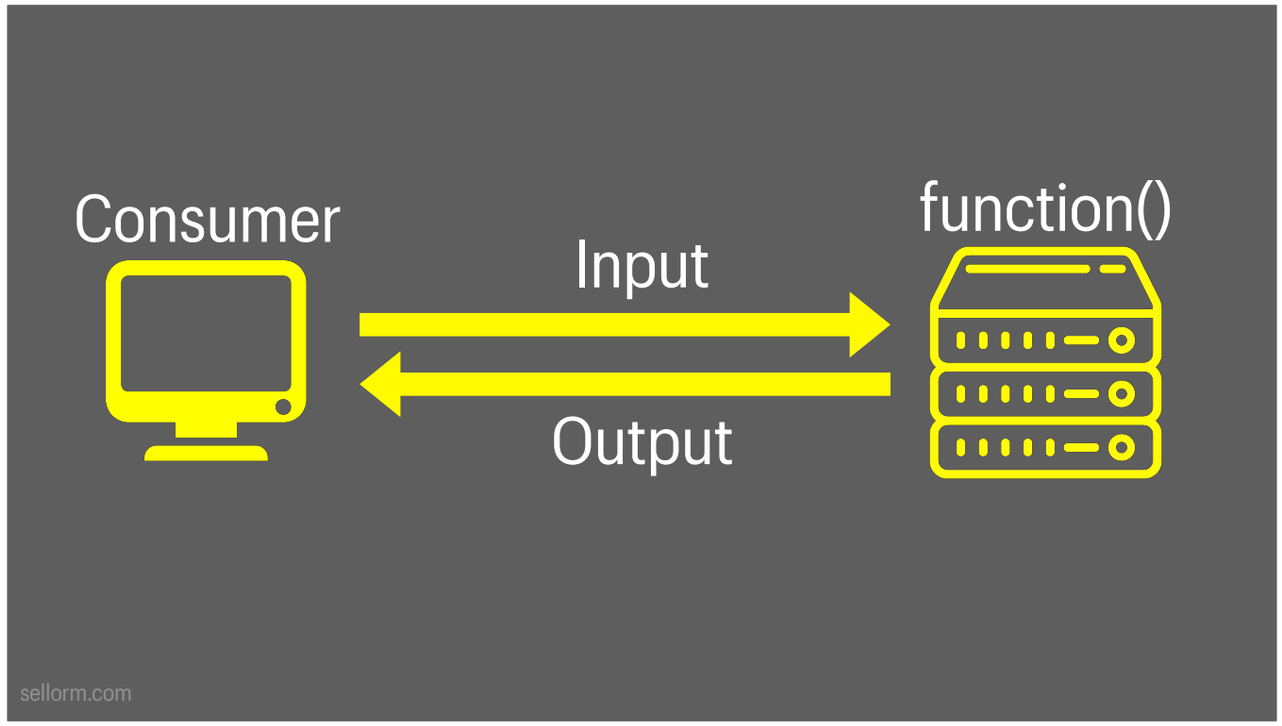
Photo by David Clode (https://unsplash.com/@davidclode)



And the R equivalent is to use the + symbol surrounded by backticks as the function.

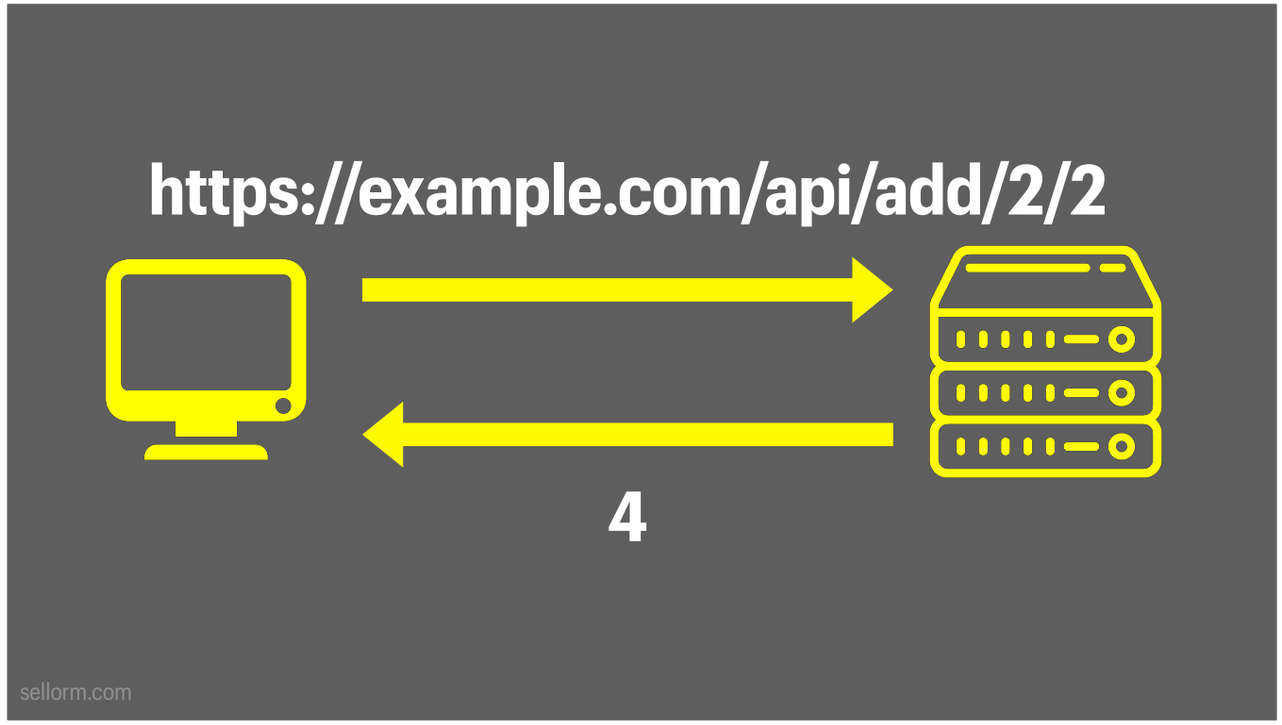
This picture is a kakapo in reference To Garret Gromelund and Hadley Wickham’s book, “R for Data Science”, since R doesn’t have an animal like Python does.

Photo: NZ Department of Conservation, CC BY 2.0 https://creativecommons.org/licenses/by/2.0, via Wikimedia Commons



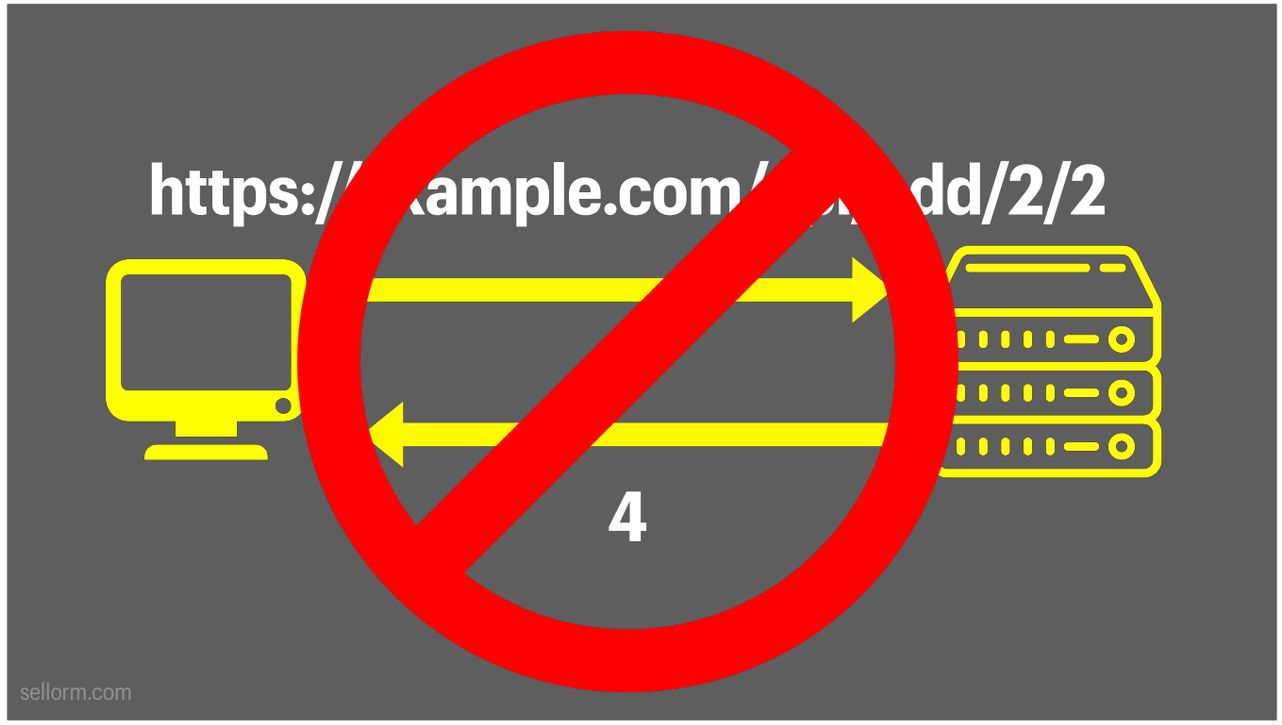
Remember the flow, because it’s going to come up a lot as we go on…

Consumer -> Input -> Function -> Output



In our 2 + 2 case, the way that this would work is that our API software would generally start a small web server for us, and then we would in some way send our input (2, 2) to the function (api/add in the example above) and it would respond with the result, in this case 4.

This is just one way to design this API. API design is a whole topic of discussion of it’s own and we’re not going to go too deeply into that here. The example above is one way it could be implemented, but it’s definitely not the the only way.



This example is also a toy example and I don’t want you to give it much thought beyond that.

It’s a kind of “hello, world” example that we see a lot in programming exercises.

Once you’ve done a few “hello world’s” of your own, you’ll end up building much richer APIs that enable you to do interesting things, like fraud detection, pricing optimisation, lifetime value prediction and so on.

A DATA SCIENCE API SHOULD BE THE INTERFACE TO YOUR INSIGHTS

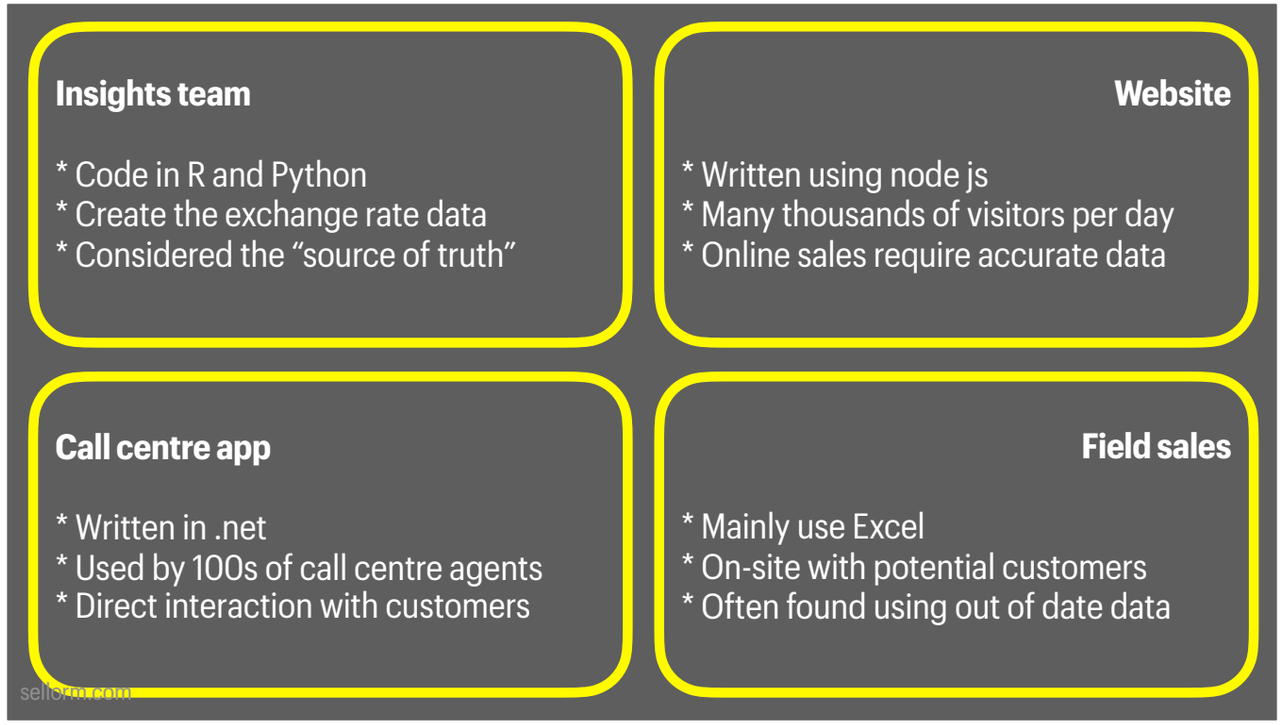
sellorm.com

A data science API, and where the power of APIs really becomes apparent, is when the API is the interface to your insights.
So if you’re building models and doing predictions, you can turn the functions that power those predictions into APIs and expose your insights to consumers on your corporate network.

EXAMPLE: EXCHANGE RATES

sellorm.com

I’m going to talk through a simple example. It’s not specifically a data science example, but it should hopefully help us illustrate why APIs are so powerful and give us some ideas about where that power can be leveraged.



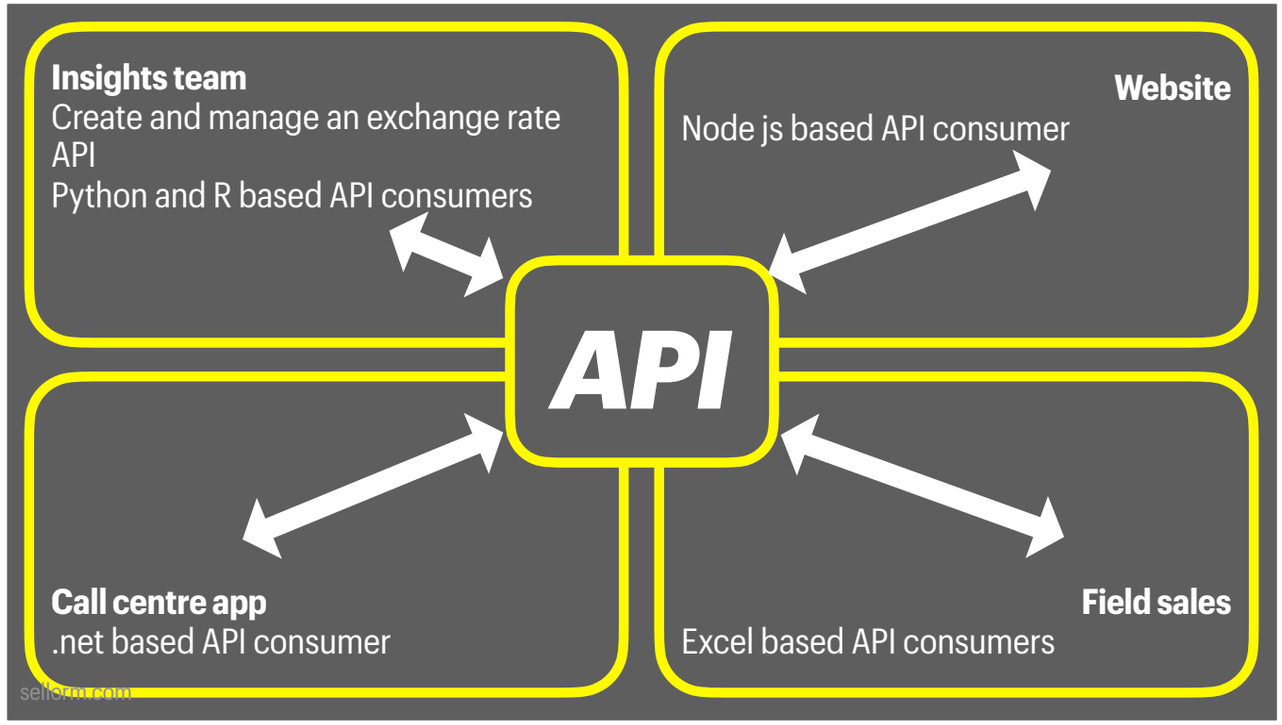
We have an imaginary business that relies on exchange rate data.

There are four groups in the business that need up to date exchange rate information for a variety of purposes.

The insight team are supposed to be the source of truth for this information, but in reality each group is gathering and using their own data from a variety of sources.

Naturally, the information each group is using is rarely in sync and even when it is, it doesn’t stay that way for long.
Think about the best way for the insight team to regain control of this situation.
They could try to communicate the information face to face in the office, but that’s difficult because not everyone is always in the office and there’s too much information to convey effectively.
They could try sending a report around with the new data once a week, but the exchange rates can update at random times and each user group would still need to manually update to the new rates.

So, what should they do?



In this case, it would be smart of the Insight Team to create an API that can be consumed by themselves as well as the other groups.

* There's now only one place to update and everyone gets the same information at the same time.
* The API becomes the single source of truth
* With this automation and "truthiness" comes a rise in confidence - all of the potentially disparate teams can be certain the information they're using is as up-to-date and as accurate as it can be.
* Notice that the key difference here is that the calculation is now uncoupled from the application. Functions provide tight coupling to their parent application where an API provides a loose coupling that can be shared across many disparate applications and many disparate programming languages.

WHEN SHOULD WE USE AN API?

sellorm.com

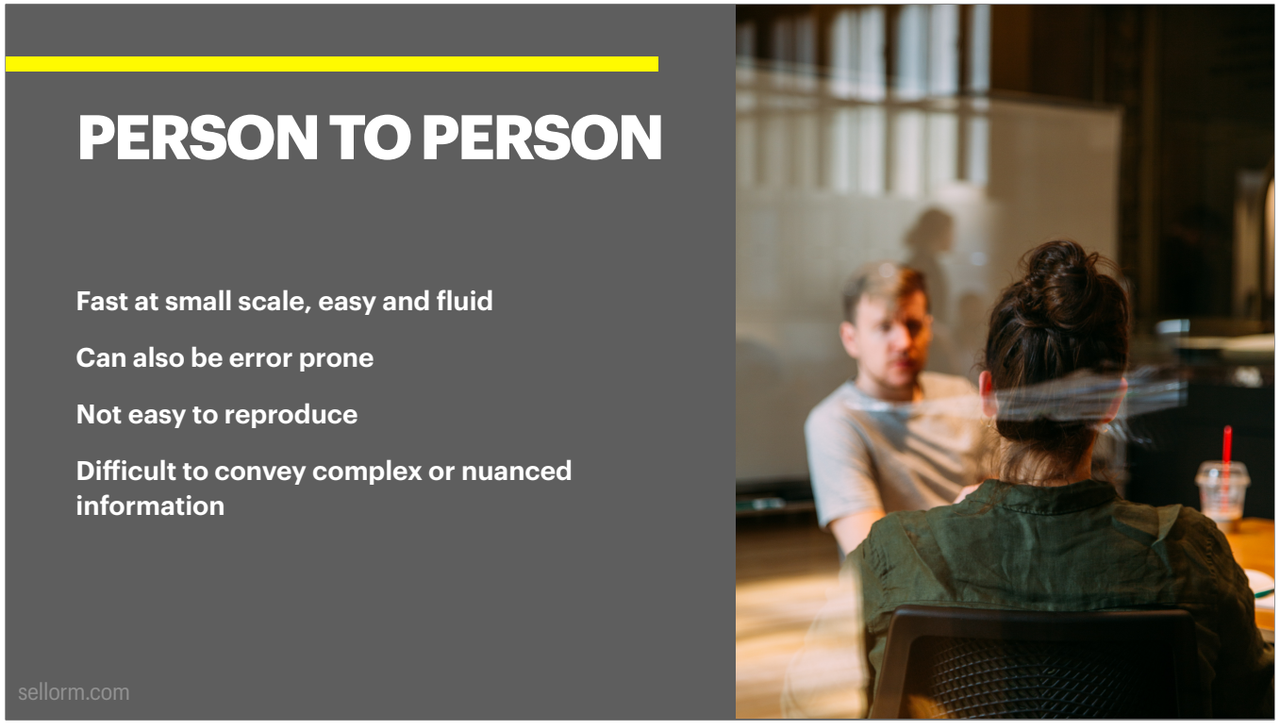
APIs aren’t great for everything though, so we should probably discuss *when* to use an API instead of another method of insight communication.

These are the different methods a data scientist might use to communicate the insights they’s generated.

<go through the list>

The way that the insight you generate is communicated is critical.

We must tailor the comms to the needs of the consumer.
If we look at each one in more detail, we’ll see the strengths and weaknesses of each



- Fast at small scale, easy and fluid

If the insight is small (think “buy more widgets!” size) we can communicate it easily in person in the lift up to the office or perhaps on a quick phone call.

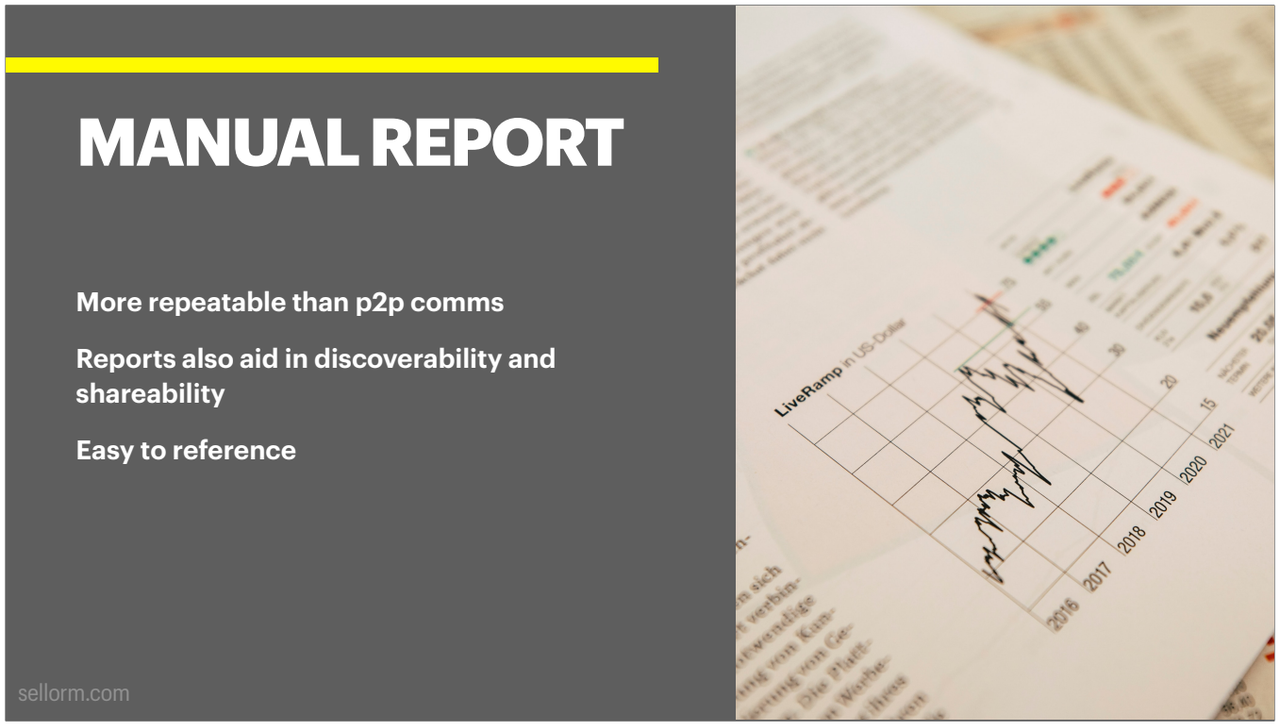
- Can also be error prone
- Not easy to reproduce

Without proper documentation, this sort of communication is of limited utility to data science. The broken telephone game can come into play if people are passing information around in this way and it’s not easy to be sure we have the correct information.

- Difficult to convey complex or nuanced information

When we communicate only in this way it is incredibly difficult to convey complex information accurately. Which is why this form of communication is rarely appropriate for communicating the sorts of insights that data scientists might generate.

Photo by charlesdeluvio https://unsplash.com/@charlesdeluvio

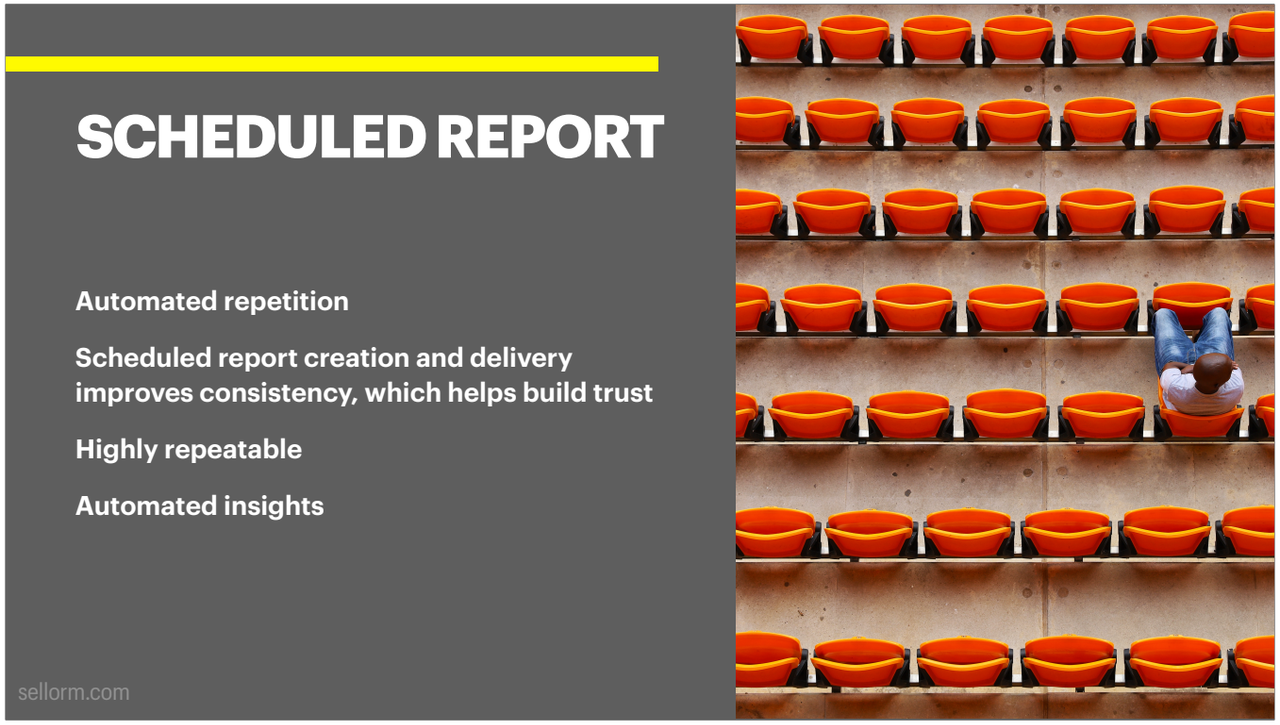


Manual reports are a mainstay of data science. Complex insight can be communicated effectively and accurately using a combination of text and charts making it easy for consumers to digest.

We can create templates for reports so that reporting is consistent. They also easy to share, via email or on shared drives, as well as easy to reference - “Did you see the forecast on page 4?”.

You can also refer to previous reports easily - “I need to check last years’ report for those figures”

Photo by Markus Spiske https://unsplash.com/@markusspiske



- Automated repetition

Automated reports are highly (often fully) automatically generated.

- Scheduled report creation and delivery improves consistency, which helps build trust

When a manager knows that the weekly sales report will be in their inbox first thing Monday morning, there tends to be an increased trust in the insight generation process. Consistency helps build confidence in the insight itself.

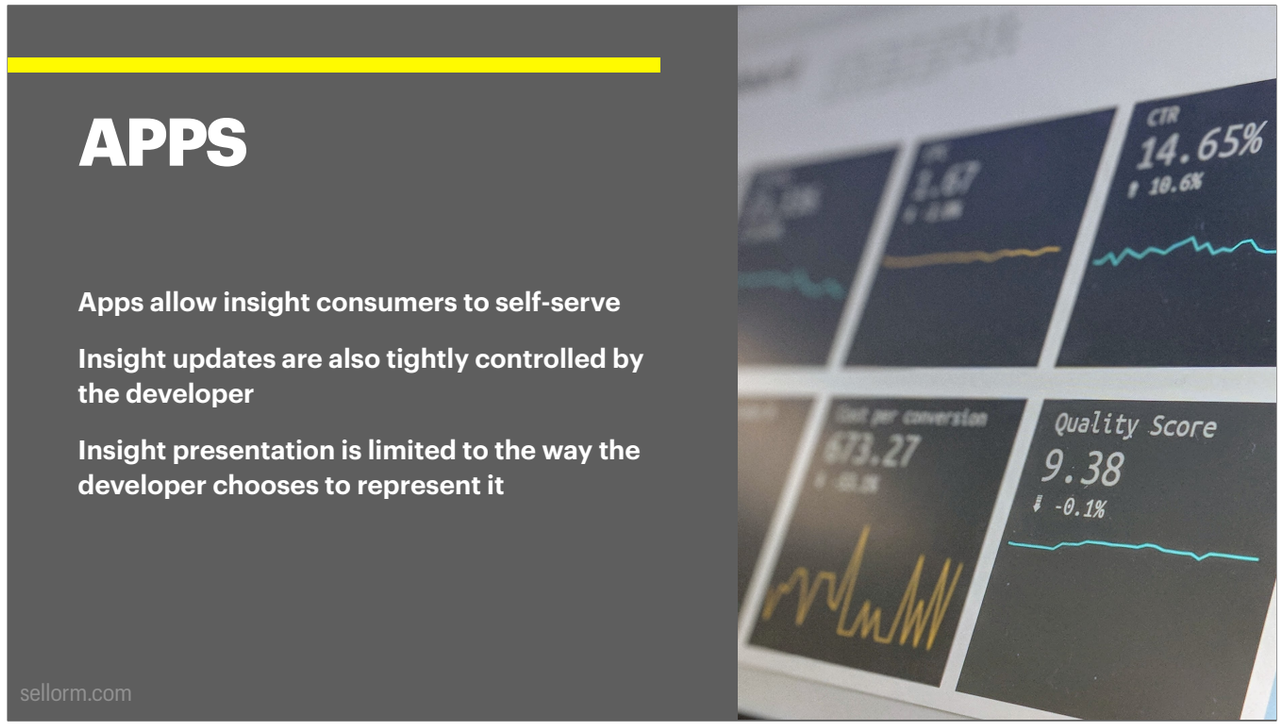
- Highly repeatable

Since reports are automatically generated, they are highly repeatable. It is possible to schedule reports for any interval, though we should be cautious about providing reports too frequently as this can work against us, reducing confidence by eroding any particular reports' worth.

- Automated insights

Automation is always appreciated as it frees people up to work on other things while reports are generated and sent automatically. This can help a data science team appear highly productive.

Photo by Nicolene Olckers https://unsplash.com/@nicolene_olckers



- Apps allow insight consumers to self-serve

Consumers can go to the dashboard whenever the mood takes them and get up-to-date insight

- Insight updates are also tightly controlled by the developer

Insight updates (model re-training etc.) are controlled by the app developer, so it could only be out of date if the app author let it get out of date.

- Insight presentation is limited to the way the developer chooses to represent it

We're only limited by our own imaginations, but this limit may preclude us from considering a method of presentation that improves the insight in some way - for instance by combining it with another data set.

Photo by Stephen Dawson https://unsplash.com/@dawson2406

PACKAGES

Packages allow other developers to leverage your insight in their own applications

You (the original insight creator) cannot control how that information is presented

Impossible to control the versions people use once they‘re out in the wild

sellorm.com

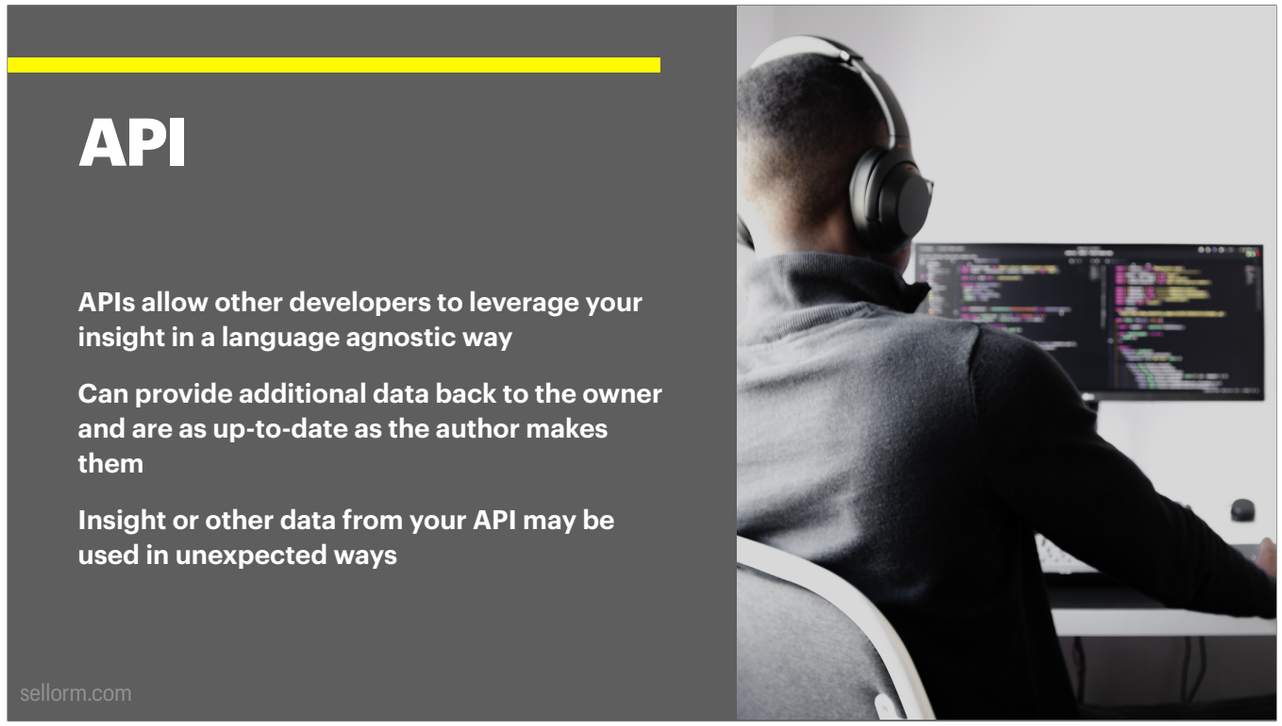
- Packages allow other developers to leverage your insight in their own applications
- You (the original insight creator) cannot control how that information is presented
- Impossible to control the versions people use once they’re out in the wild

Packages are perhaps more useful for code

Insight often updates too quickly to make packages a sensible approach

Limited to one language - do you make just an R package, or Python, or both etc.

Photo by RoseBox https://unsplash.com/@rosebox



- APIs allow other developers to leverage your insight in a language agnostic way

You don’t need to worry about what language the consumer is using. Consumers can use languages you do not yourself know and this adds to their flexibility.

- Can provide additional data back to the owner and are as up-to-date as the author makes them

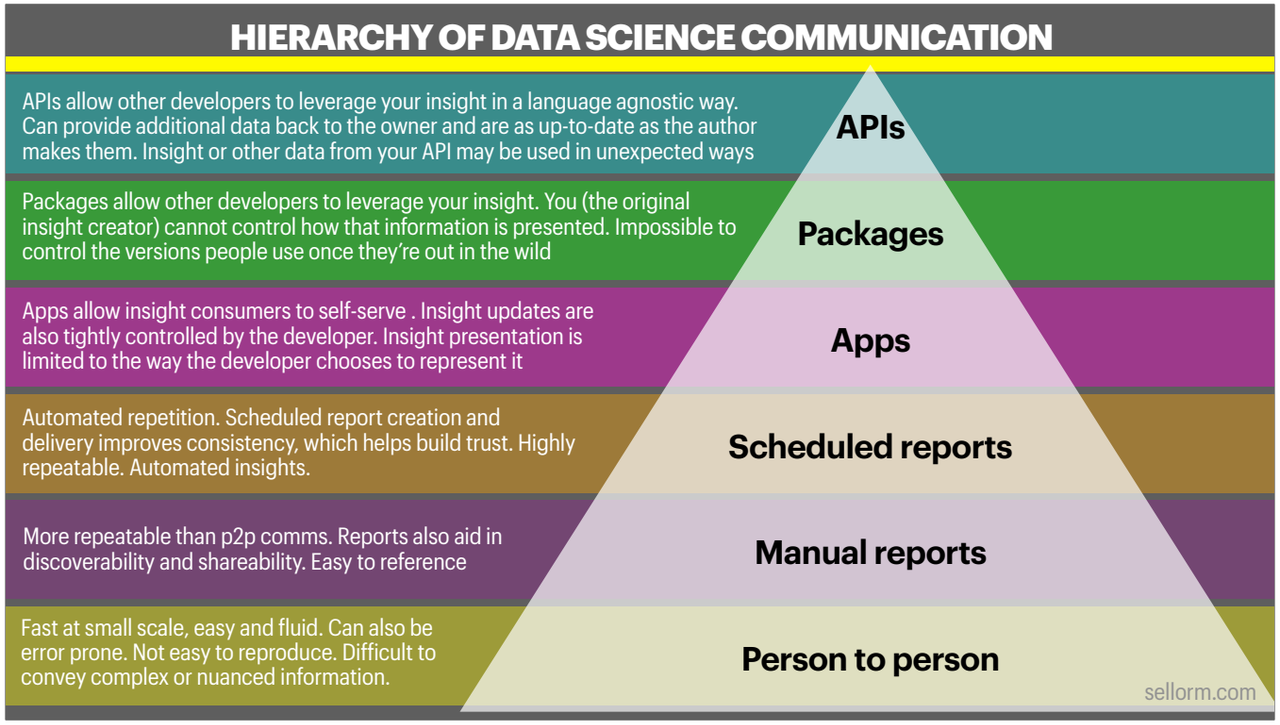
With appropriate logging, you can see who is using your API and what information they’re passing into it. This can be useful for improving the application in the future, or understanding when an API could be retired and so on.

- Insight or other data from your API may be used in unexpected ways

API consumers may have entirely different mindsets to yourself and might, therefore, consider ways of using the insight that you had not yourself considered.

“Other developers” could also mean yourself working in a different context as the consumer of your own API.

Photo by Nubelson Fernandes https://unsplash.com/@nublson

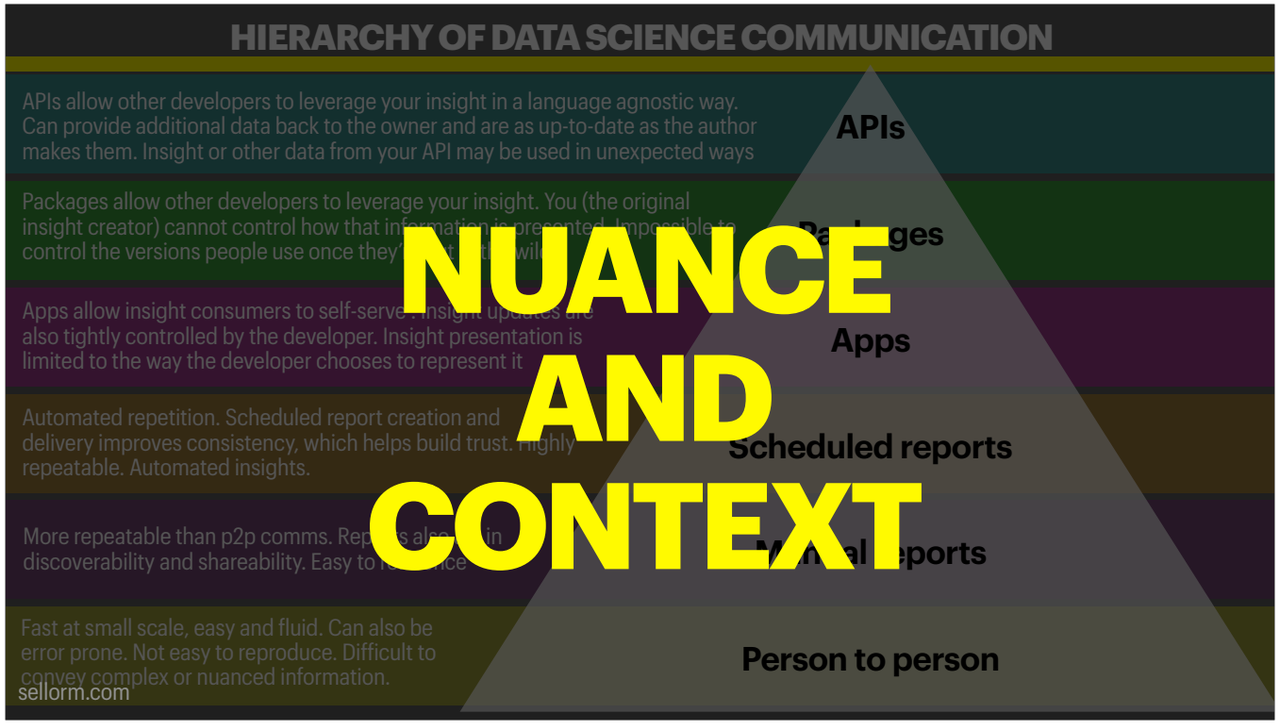


So we end up with this… A hierarchy of data science communication

- At the bottom there, we have person to person - it's fast, it's fluid, but it also doesn't scale and isn't repeatable
- Then we have manual reports that bring in that repeatability and improve discovery and so on along the way
- Scheduled reports automate the report creation allowing consistent and repeatable insight communication
- Apps allow customers to self-serve giving them new power to answer their own questions
- Packages allow other developers to leverage your insight, but in a language specific way
- And then finally, APIs allow other developers to leverage your insights in a language agnostic way

This hierarchy is arranged in terms of flexibility and reliability/consistency.

I'm not suggesting that APIs are best in all cases, but they can certainly be more flexible that other types of communication.



Any information presented like this must be taken with a pinch of salt and there's a load of nuance and context specific caveats and so on in all of this.

For example, I wouldn't give the CEO an API

And I wouldn't deliver insight to another development team in a report

We need to understand the tools and where they're strong so that we can choose the right tool for the audience.

It’s also important to remember that APIs (and the hierarchy more generally) are not a proxy for data maturity.

The mature data science organisation knows about APIs and when to use them, but that also means knowing when not to use them. A company that never creates an API may be the most mature in the world if they don’t actually need one.
It’s useful to test this stuff for its own sake, but creating APIs without a specific need is rarely a good idea.

—

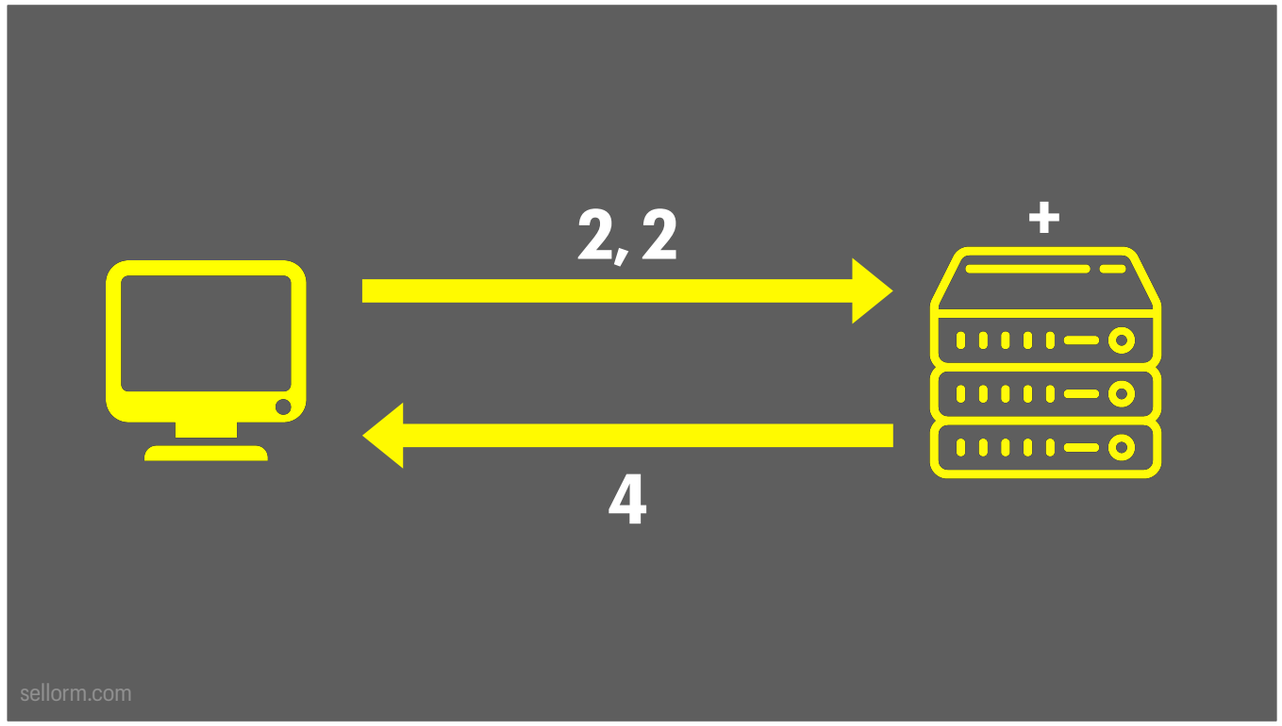
Read more about data maturity here - https://www.dataorchard.org.uk/what-is-data-maturity

Photo by Gustavo Fring: https://www.pexels.com/photo/happy-elderly-man-riding-scooter-outdoors-4975308/

YOUR FIRST API

sellorm.com

Let’s talk about your first API…

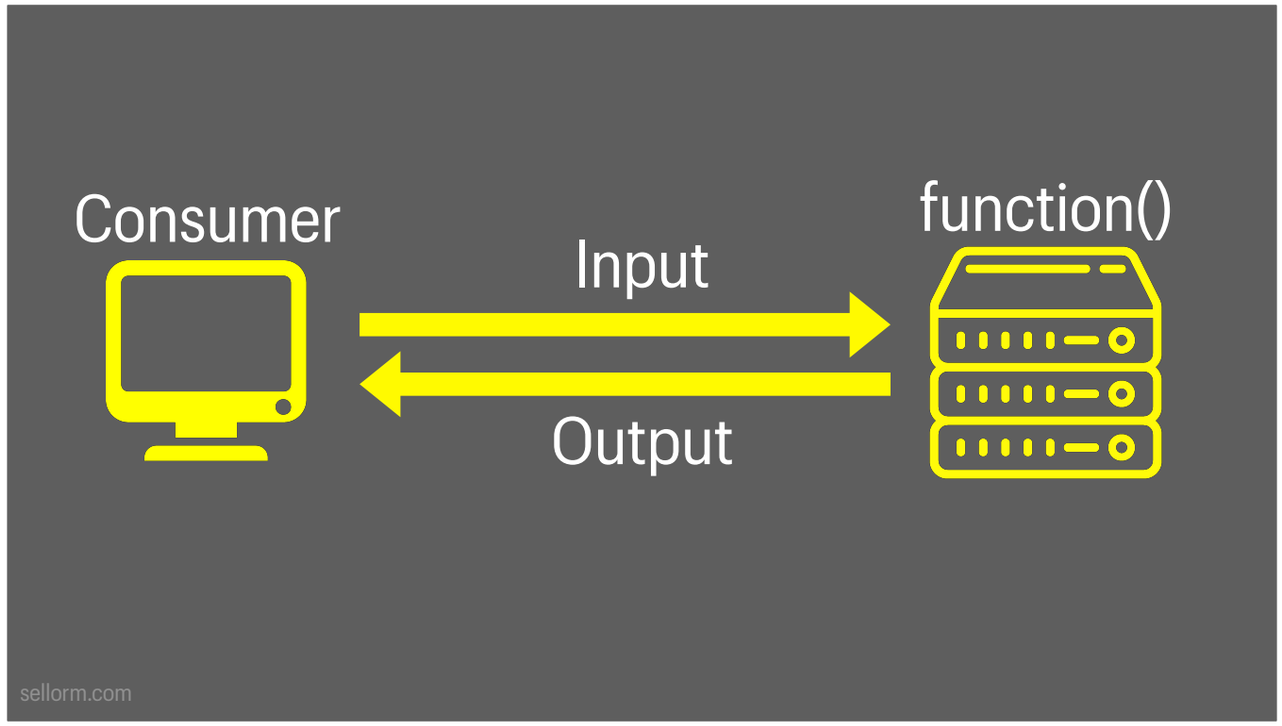


It’s the same diagram from earlier.

These kinds of APIs are the “hello, world” of API creation, but they won’t keep you entertained for long.

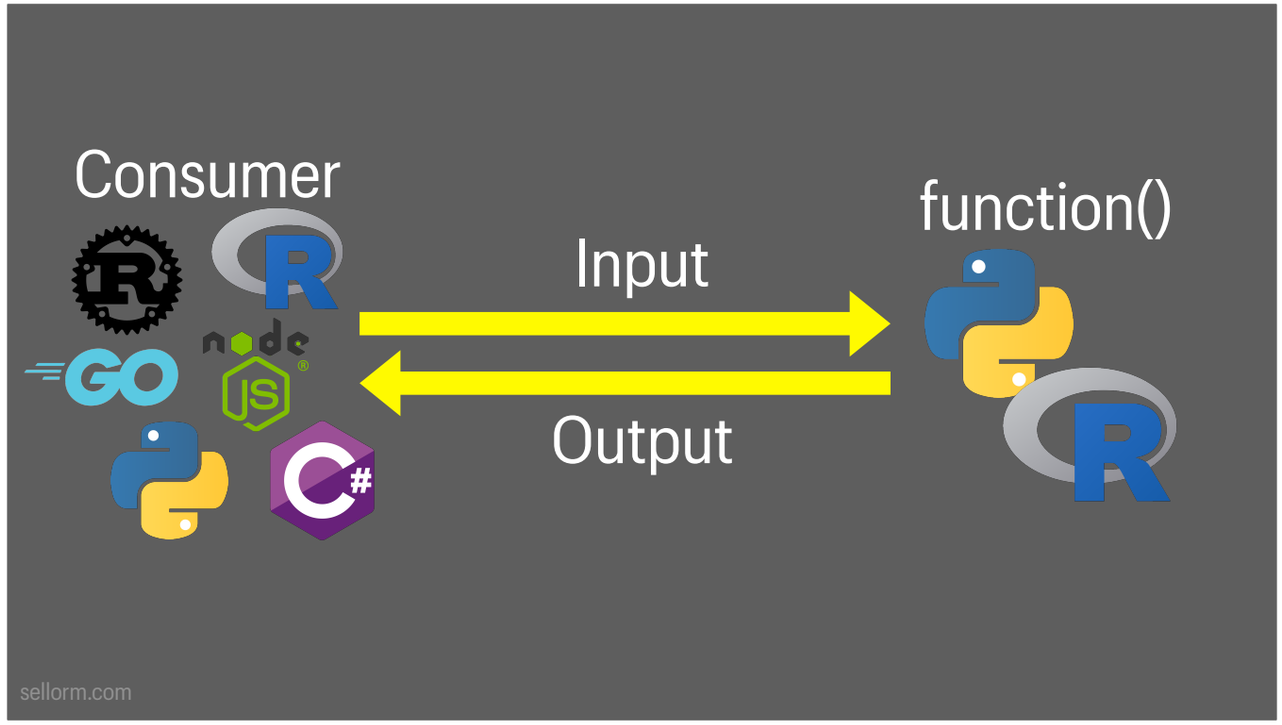
—

“Hello, world” programs are explained here - https://en.wikipedia.org/wiki/"Hello,_World!"_program



Remember our flow…

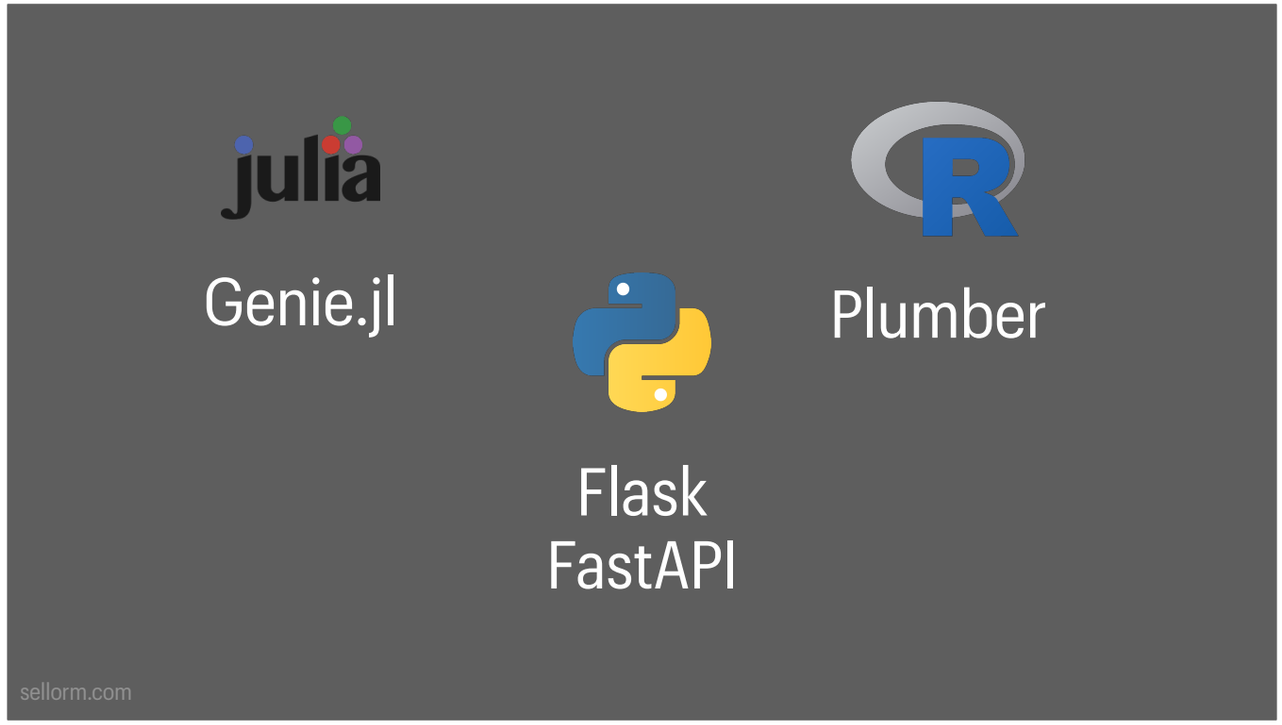
Consumer -> Input -> Function -> Output



Completely language agnostic and there are lots of languages to choose from.

I've emphasised Python and R on the "function()" side, since they're most common in the data science space.

I've only included six examples on the consumer side, but more-or-less every language you've heard of from the last 30 years can interact with an API.



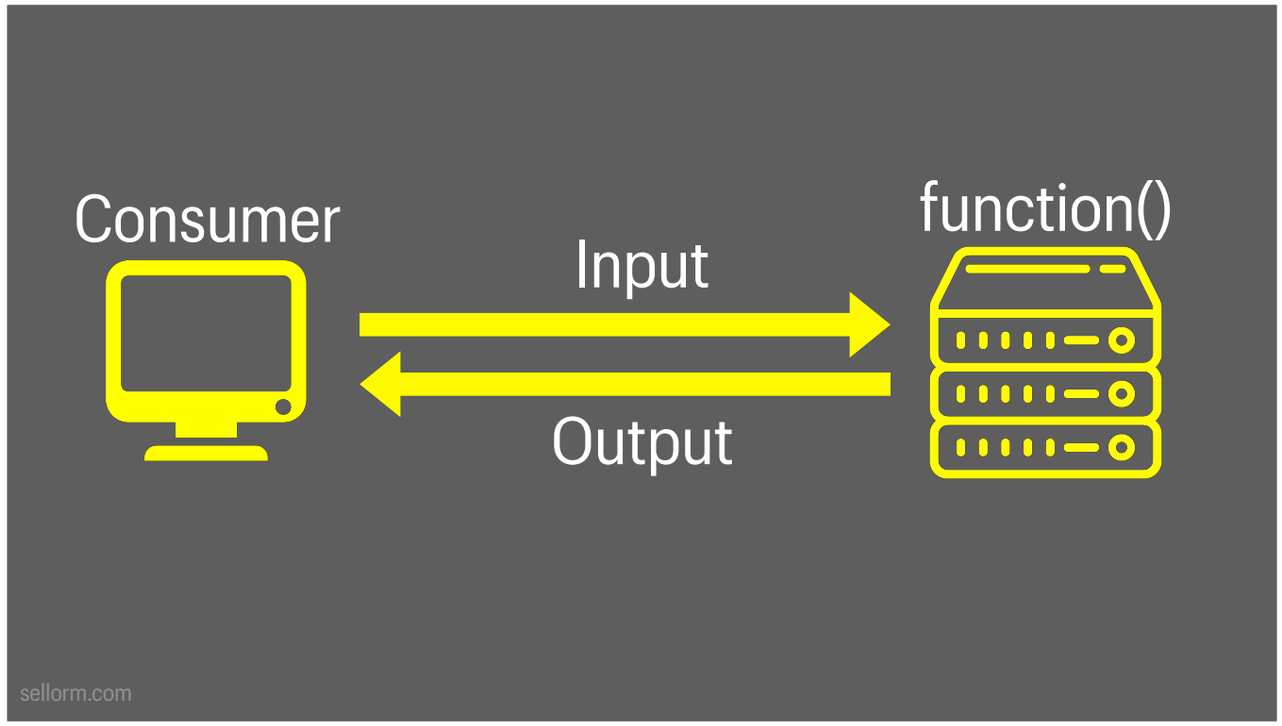
I've not done a great dal of work in Julia, so I can't vouch for it, but Genie seems to be popular.

For R we have the plumber packages

And In Python we have (among numerous others) Flask and FastAPI.

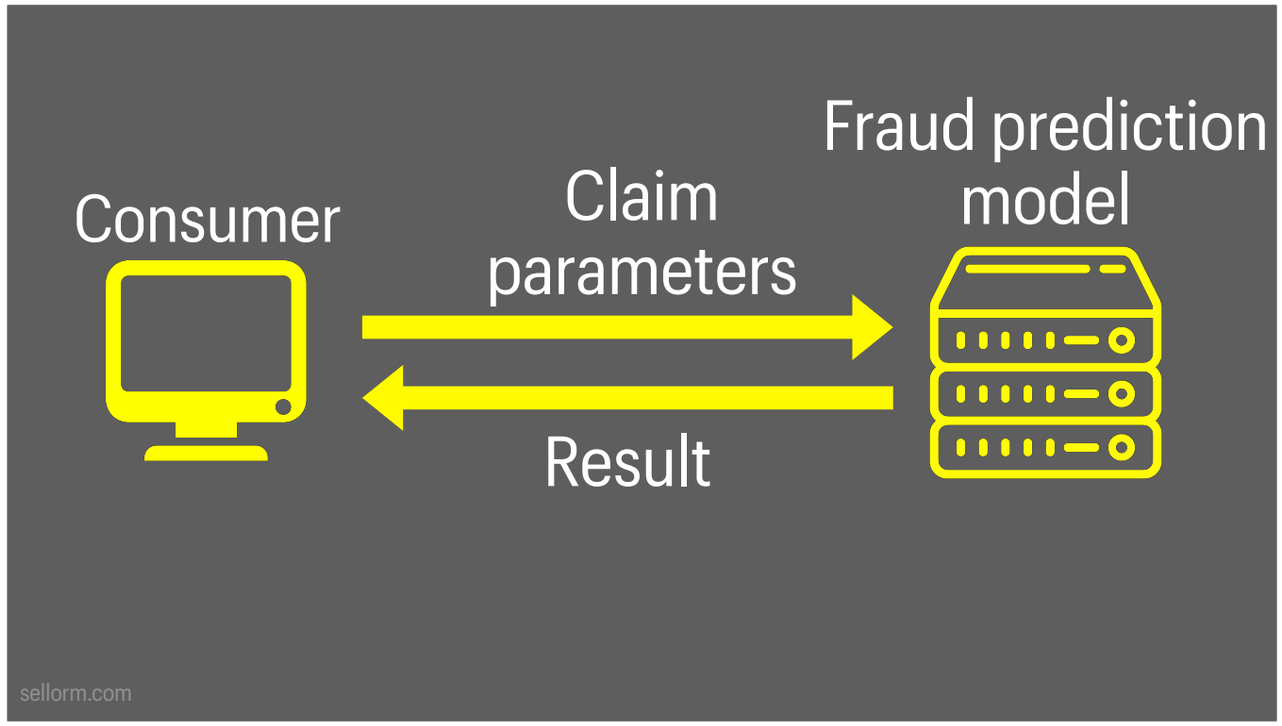
You can read more about them on the respective projects websites including lots of example code.

Flask - https://flask.palletsprojects.com/en/2.3.x/
FastAPI - https://fastapi.tiangolo.com/lo/
Plumber - https://www.rplumber.io/
Genie - https://genieframework.com/



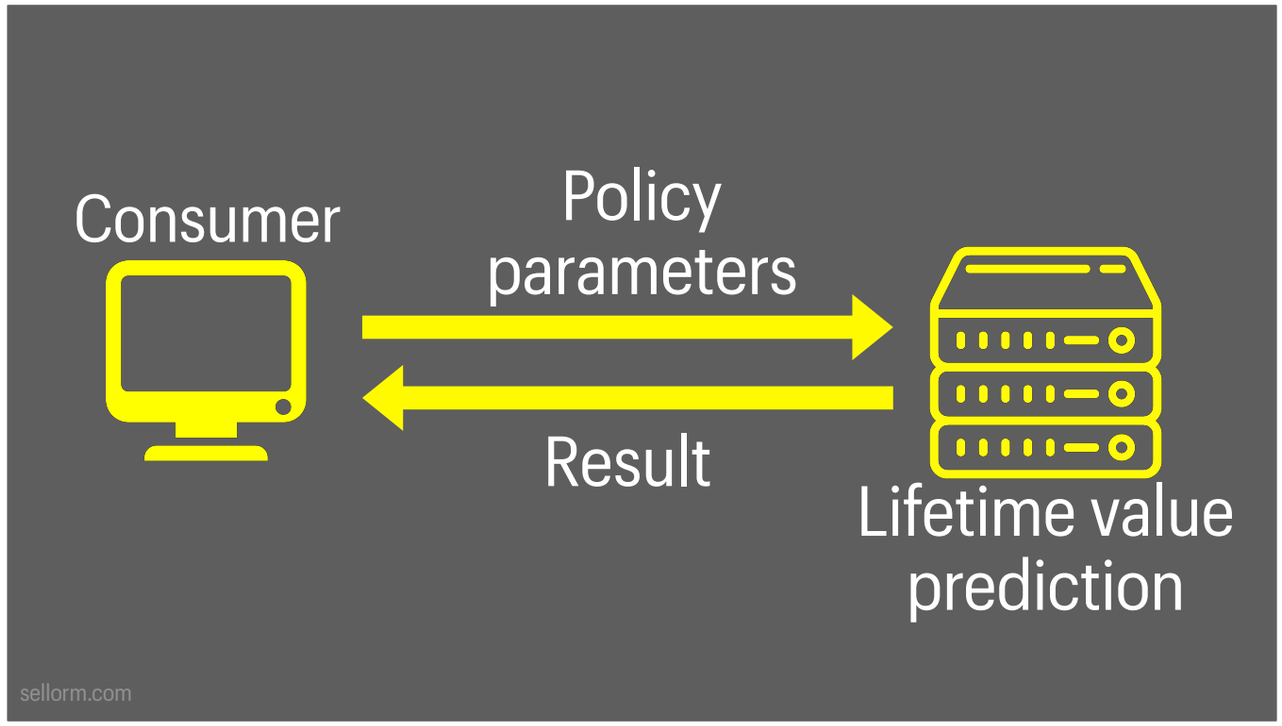
Let’s take a look at some industry specific examples for a little bit of inspiration.

Consumer -> Input -> Function -> Output



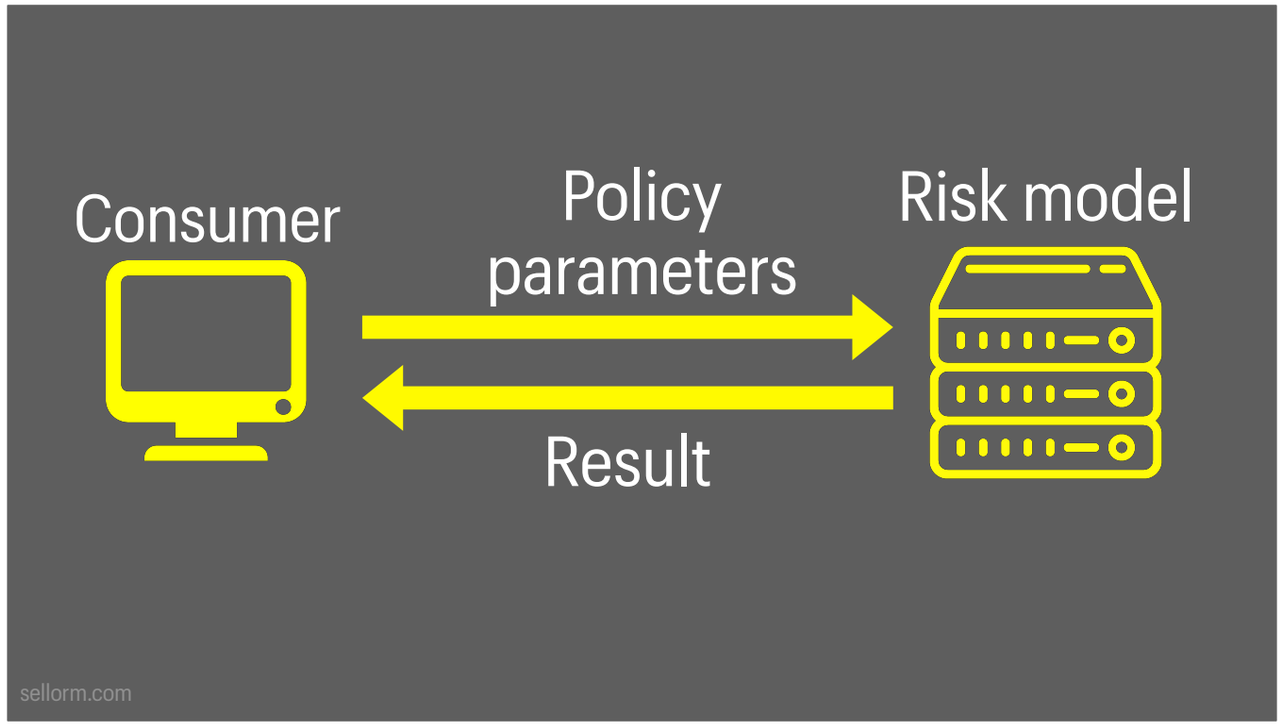
In this example, the consumer sends insurance claim parameters to a fraud prediction API.

The result could be returned as a score representing the likelihood that the claim is fraudulent.



In this one, the consumer sends policy parameters - probably along with policy holder parameters - to a lifetime value prediction API.

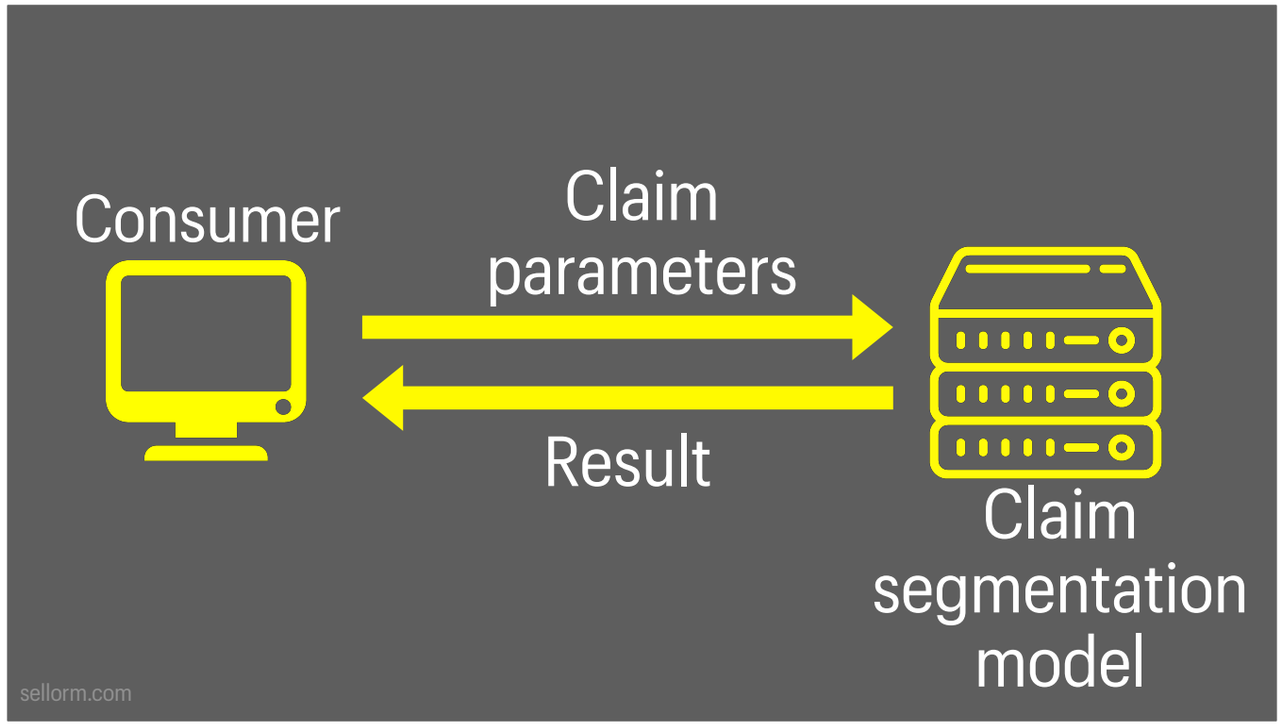
The result might be a dollar amount for the prediction, or could be bucketed (low, medium, high, VIP).



For a risk model, we'd send policy parameters - and probably the policy holder details - to the risk model.

The output could be a risk score predicting the likelihood of a claim from this policy holder.

You could also imaging this in a re-insurance setting as a portfolio risk API where the input is actually portfolio parameters and the output is the overall risk prediction for the entire portfolio.



In a claim segmentation model API, we could pass in claim parameters and use the model to automatically reject (based on sound, explainable reasoning), accept or — for the fuzzy middle ground — pass on to a human claims processing team for further consideration.

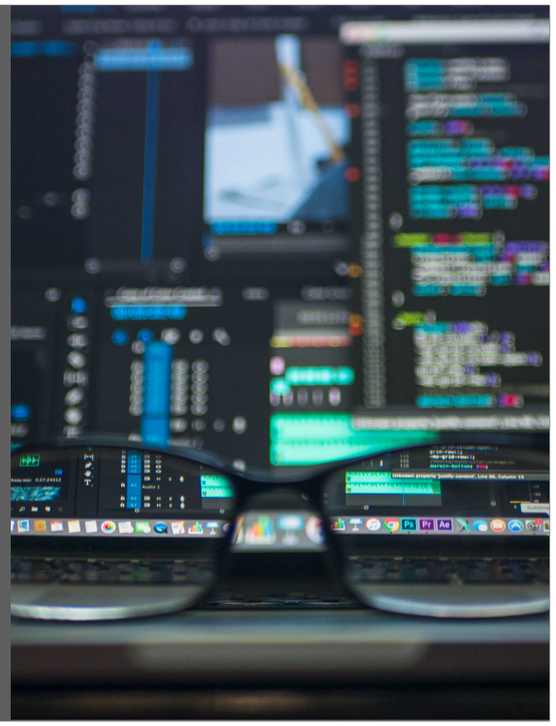
Or insight component delivery eg could return geoJSON if we’re working in spatial.

The key with all of these examples is that we’re delivering insight and in the majority of cases this is where data science should focus it’s API efforts.

Photo by Diego PH https://unsplash.com/es/@jdiegoph

DATABASE WRAPPER

sellorm.com



There is one other sort of API that can be extremely useful though, and that's a data wrapper API.

Database wrappers are a convenient way to retrieve or store information in a database.

I wouldn't use them for bulk operations or extracting large volumes of data in a single API call, but for smaller operations they can be really useful.

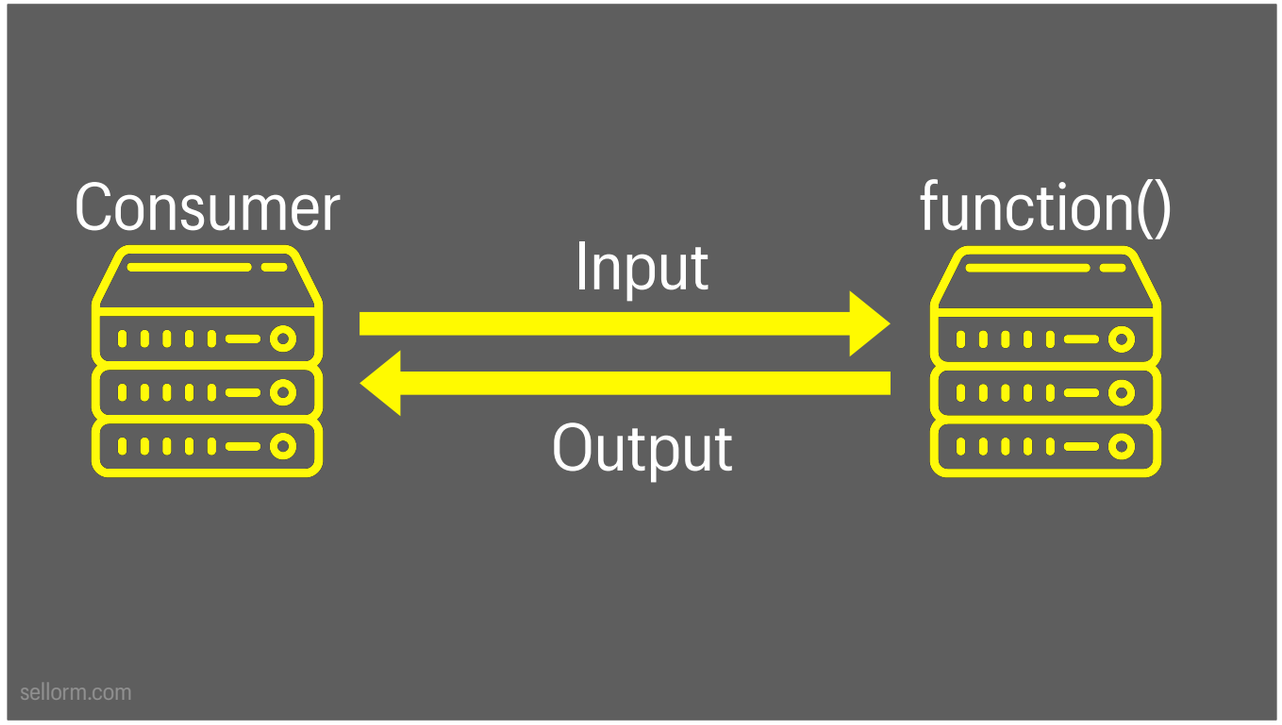
This is particularly when used with more insight based APIs.

For example it might be useful to combine a customer lifetime value prediction API with a data base wrapper API.

That way instead of sending all the customer details to the insight API, you could just send the customer ID and the lifetime value API could extract the details it needs from the database with another API call.

Creating this sort of API means that you don't need to mess about installing and configuring database drivers on all of your systems. That can be left to the system running the database wrappers alone.

Photo by Kevin Ku https://unsplash.com/@ikukevk



Now, what if the consumer is another system?

This is where things get really interesting
Because we end up with a decoupled, but highly composable insight architecture.
Which in turn can driver further insights.

Connecting machines to other machines in this manner allows you to create complex pipelines of operations.

DATA PIPELINES

sellorm.com

Using APIs in data pipelines allows you to increase the level of automation in your workflows.
Data can be pushed through your organisation from capture and storage to inclusion in an insight generation process.
You can enrich existing datasets using automated processes and increase speed, confidence in your results and trust in the data being used.

Imagine an insurance company in which the data, models and assumptions are connected via APIs, so that information can be trusted and flow seamlessness from one function to another, from business planning and capital setting, to pricing and underwriting, and claims handling and reserving, while at the same time each team might use different tools and applications.

Photo by 夜 咔罗 https://unsplash.com/@karosu

AUTOMATION

sellorm.com

All of this enables automation

Greater automation means greater speed and agility, more flexibility and the ability to do things quickly, but in a high-trust environment

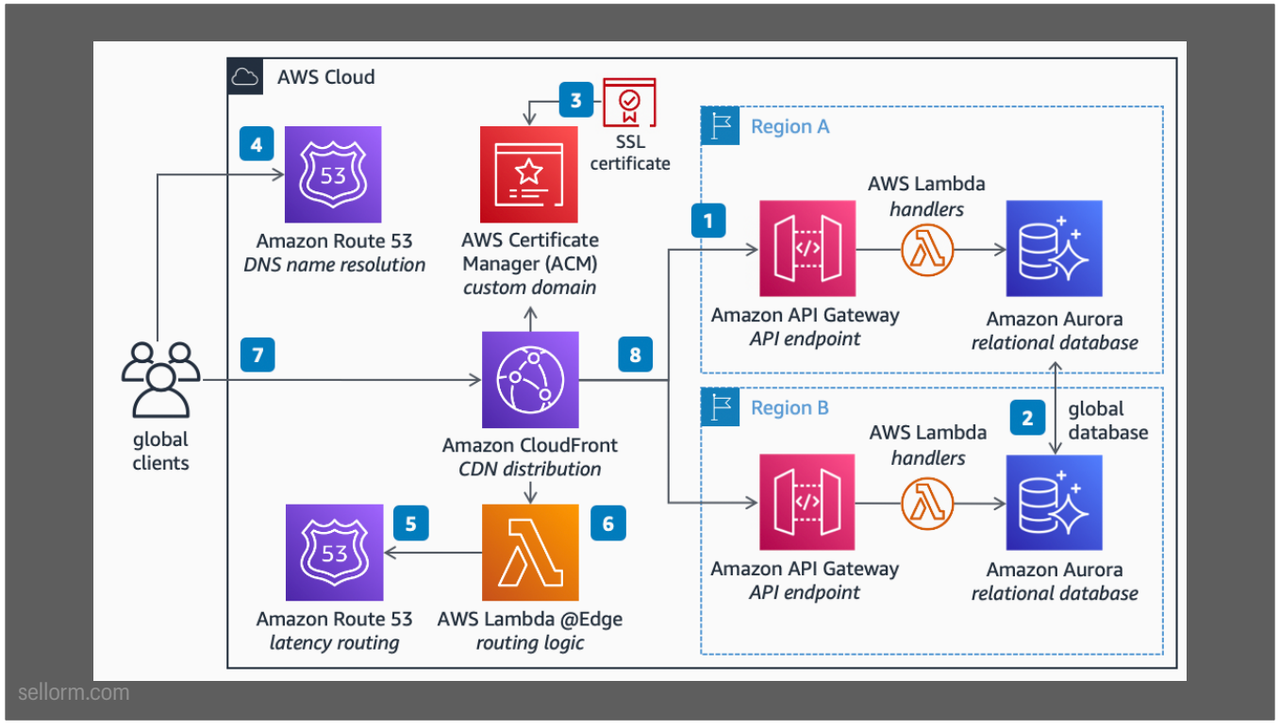
Photo by Adam Lukomski https://unsplash.com/ko/@anilinverse

API DEPLOYMENT ARCHITECTURES

There are a lot of ways you can deploy an API…

…and I want you to go away today with a reference architecture you can use in your own projects, especially when you're just getting started.

Photo by Donny Jiang https://unsplash.com/ko/@dotnny



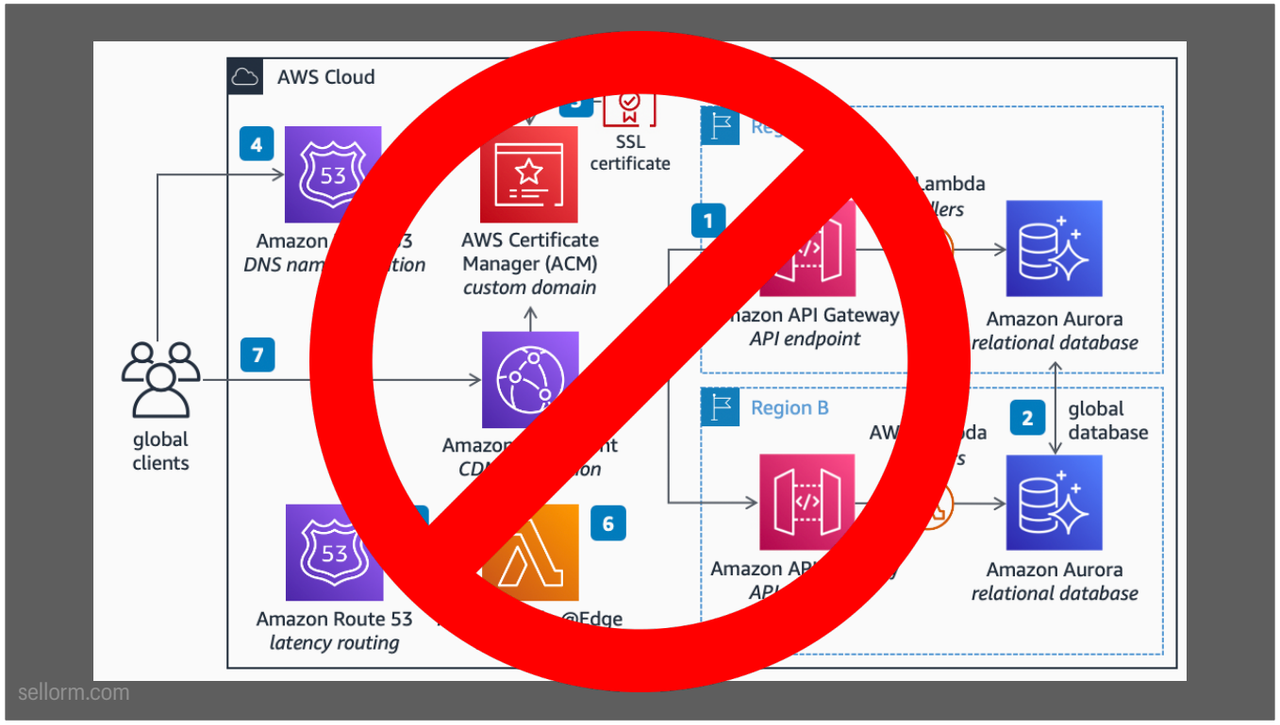
Here’s a reference architecture:

But we’re not going to use this.

This is the sort of thing that IT departments and cloud computing companies love, but it’s not the best place to start.

Borrowed from: https://docs.aws.amazon.com/architecture-diagrams/latest/multi-region-api-gateway-with-cloudfront/multi-region-api-gateway-with-cloudfront.html

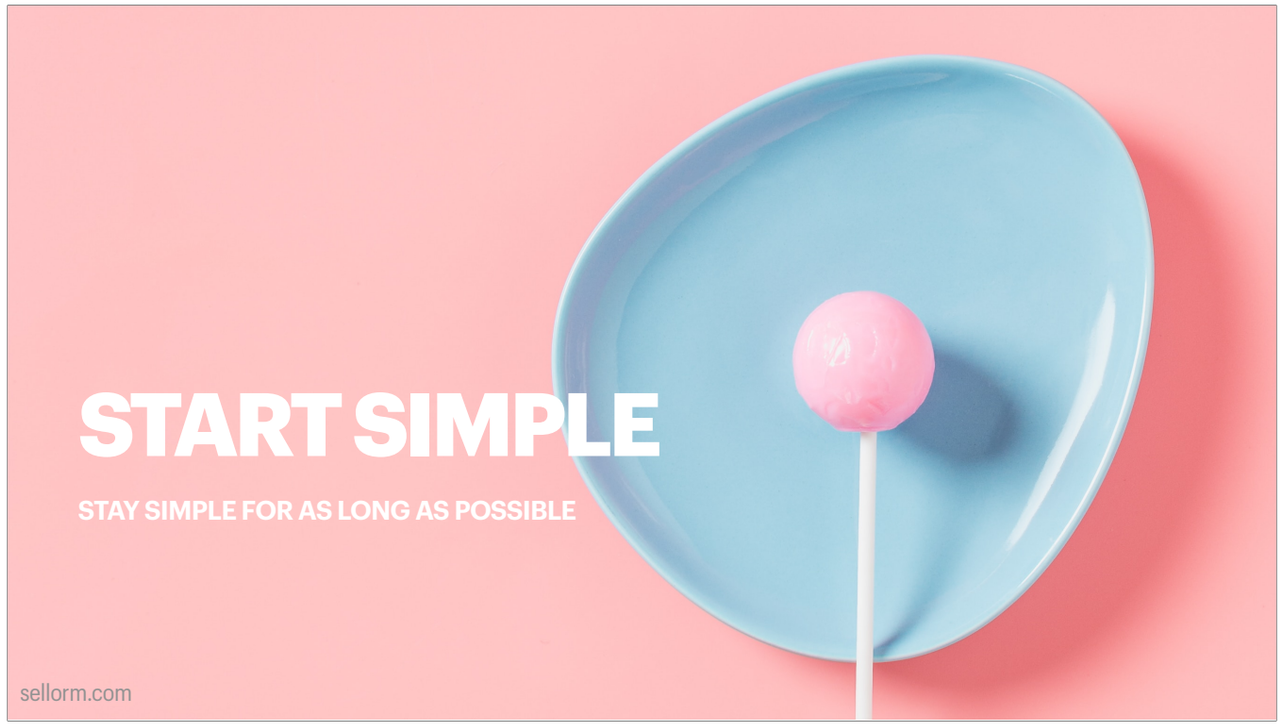
I want to be really clear here - this sort of architecture is great, it’s very powerful, flexible etc - but if you’re just starting out, it’s also complete overkill. A project like this will take too long to deliver and almost certainly not recoup the costs quickly enough to show real business benefit in the short to medium term.



So, we’re not going to use it.

Nothing wrong with this sort of architecture, but it’s not the best place to start

Initially, when you’re creating and publishing APIs your goal is to learn, but in order to do that you actually have to get something out in the world.



I want you to start simple…

Adhere to the KISS principle (https://en.wikipedia.org/wiki/KISS_principle)

Keep the API simple, something that a single person can manage,
Deploy it in the simplest way you have available to you.

Your first APIs should be internal-only - external/public-facing APIs can come later.

Think carefully about the design of your API. Make it easy for end users (yourself or other programmers) to use.
Remember that even in machine-to-machine comms, the first users of any API are always other humans, who need to understand how to work with it in order to write the programs that will interact with it.

Stick to simpler-to-implement stuff in the REST area.
GraphQL is the new hotness, but it's not a good fit for writing your first API.

Work directly with early consumers to ensure the API meets their needs.

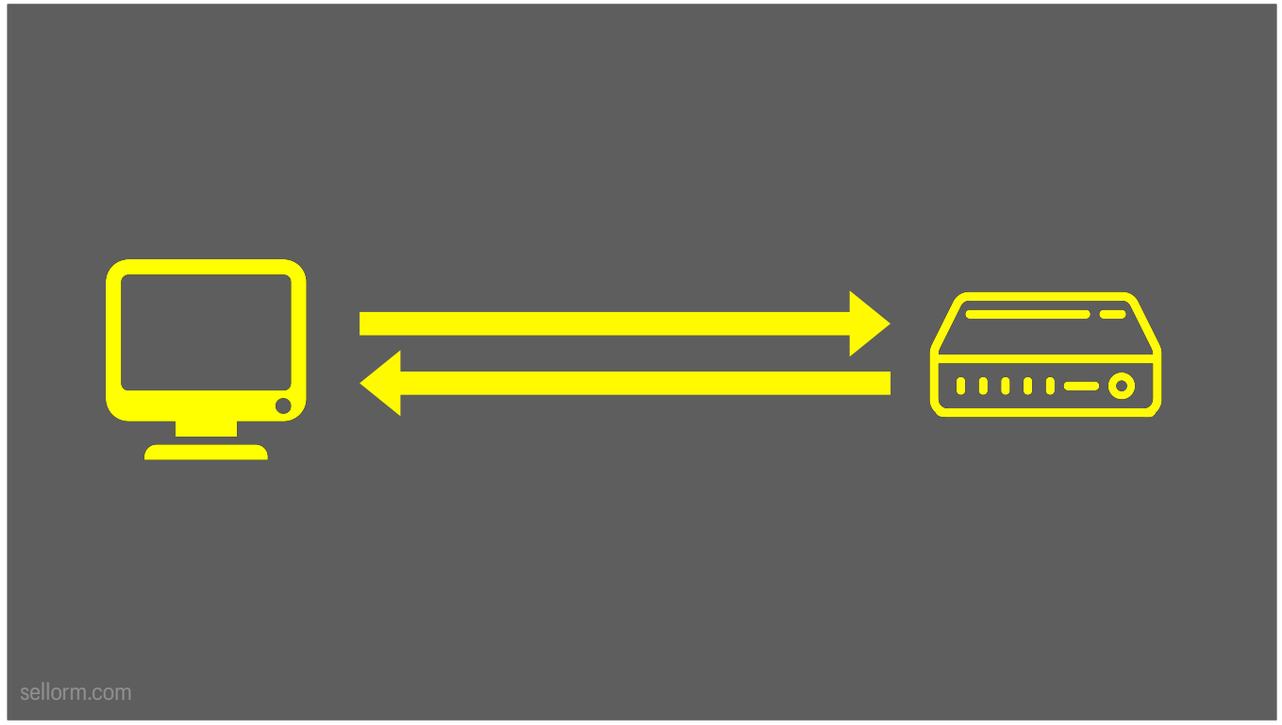
The important thing is to start. So, here's you first reference architecture…

Photo by Mae Mu https://unsplash.com/@picoftasty

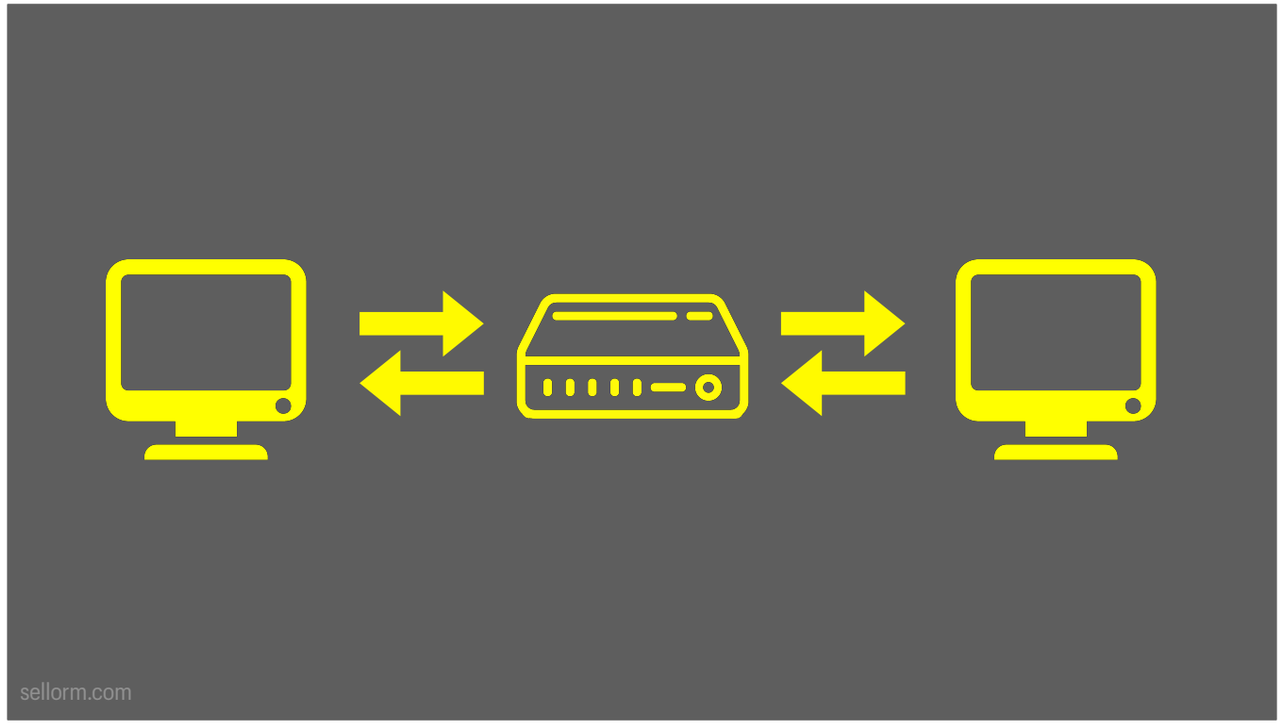
…there you go, that’s it.

So this is your actual first-pass reference architecture
Think smaller.
When you’re starting out, your needs are far smaller.
This could be accommodated with a single server.
You can host direct on the server, or use Docker if you’re into that (but you don’t need to be)
Life’s even easier if you use a platform like Domino Data Lab or Posit Connect.
Using a platform like this means you can publish your API immediately and people can start using it immediately.
Some of you have these platforms available to you today. You could go into the office next week and have an API published in a day or two.

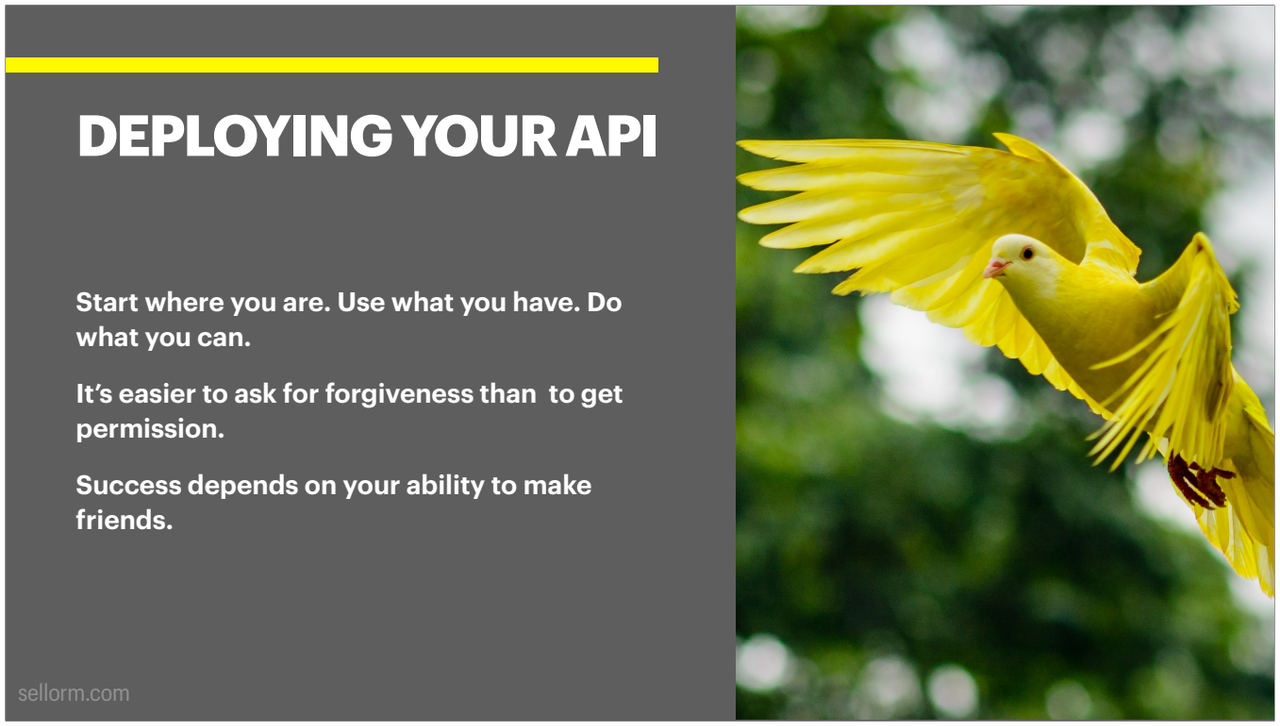
Now, sometimes when you show this to people, they have a tendency to freak out because it’s not complicated enough.



So, here it is with a consumer on it if you want to complicate it a little.



And here it is with another consumer on it if you want to complicate it a little more.



Some nice quotes which aren't directly about deploying APIs, but I feel have some relevance to the situation.

- Start where you are…
Usually attributed to US tennis player, Arthur Ashe. Sometimes also attributed Theodore Roosevelt, but actually comes from elsewhere (https://pacific.edu.ni/start-where-you-are-use-what-you-have-do-what-you-can-arthur-ashe/)

In this context, this is about not waiting for some future state before starting work in understanding APIs, but rather to get going today by working with the tools you already have and either accepting their limitations, or working around them.

- Easier to ask forgiveness…
This quote is often attributed to pioneering computer scientist Grace Hopper, but again it's much older than that.

If you already have a way to do this stuff without bringing in extra people from around the business, use it! Domino Data Lab and Posit Connect both have the means to publish APIs immediately. You don't need to ask permission to use things that are already available to you. If you go to your IT team and ask them if you can publish an API, you're going to walk straight into a months long API deployment, and I'd love for you to avoid that when you're starting out. Naturally, if your business has expressly forbidden such an activity I'm not suggesting for one minute that you should violate company policy.

- "Success in [*show business]* depends on your ability to make [and keep] friends."
Singer, comedian and actress, Sophie Tucker

Sometimes, your life will be easier if you can find a sympathetic bridge builder on the other side of the chasm you're trying to cross. If there's someone on the IT team who's shown a particular interest in your work, bring them along for the ride. They may very well be an excellent ally in deploying your APIs

Photo by Pixabay: https://www.pexels.com/photo/flying-yellow-bird-459198/

OTHER CONSIDERATIONS

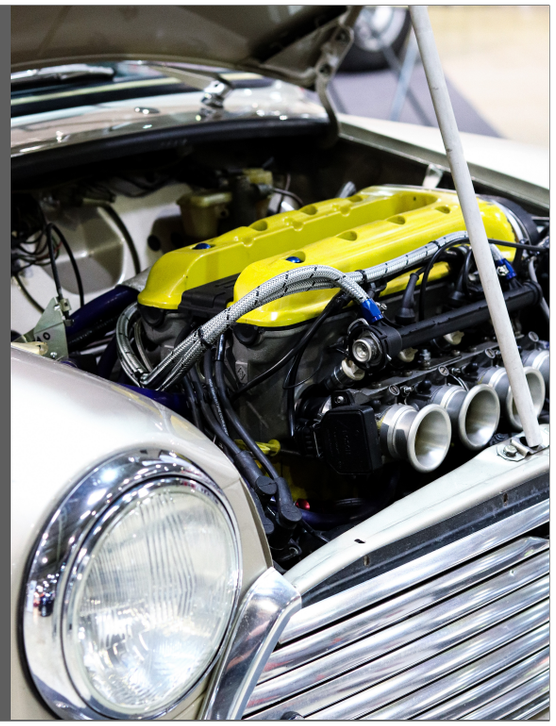
sellorm.com

There are lots of other consideration that pop-up the deeper you go into the API landscape.

I’m going to briefly talk about some of those just to get you thinking.

MAINTENANCE AND SUPPORT

sellorm.com



Who’s going to do it? You?

To start with, yes, but as your businesses API usage grows, you probably don’t want to be that person.

Do you want that 4am call because the API is down and you’re the API person?

Do what you can to make the API easy to support - good error handling and messages, logging etc.
APIs have a habit of becoming all consuming, especially when you have a couple of business-critical ones under your belt. Once you’re somewhat confident in what you’re doing, bring other people along for the ride.

Photo by Tommy https://unsplash.com/@tommyluvi

Logging is an important component of any API, especially if you’re handing off support — or sharing support — with another group.

It’s really useful to share information from the API in the form or a log so that others can more quickly see what the API is doing - or perhaps what it’s not doing!

—

There’s more information about logging here - https://blog.sellorm.com/2021/06/16/getting-started-with-logging-in-r/
It’s specifically about R, but much of the information is broadly applicable to any language.

Image: Photo by Etienne Girardet https://unsplash.com/@etiennegirardet

API HOSTING

sellorm.com

Hosting is a whole topic of it’s own and it’s an extremely complex one. The important thing to remember is that it’s always easier to work within existing deployment framework within your organisation. Introducing a new API at the same time as pushing for new API hosting technology can be an extremely long road.

Try to work with what you have as best as you can, for as long as you can.

Photo by Taylor Vick https://unsplash.com/ko/@tvick

You should not worry about this when you're starting out. Most early stage APIs are very low traffic (and that's OK!), so even the smallest server hosting them would be fine. As the complexity of your APIs increases and the number of client interactions goes up, you can start to think about it.

Remember, even though you might need a lot of compute to train a model, the resulting model objects are generally very small, so wrapping that object in a little function to handle predictions can leave you with an API that requires very few resources to run.

Photo by Naina Vij https://unsplash.com/@_nainz

Once an API is in use, you (the API developer) can’t just change it, as you now have down-stream dependencies that might include things you really don’t want to break, like your corporate website.

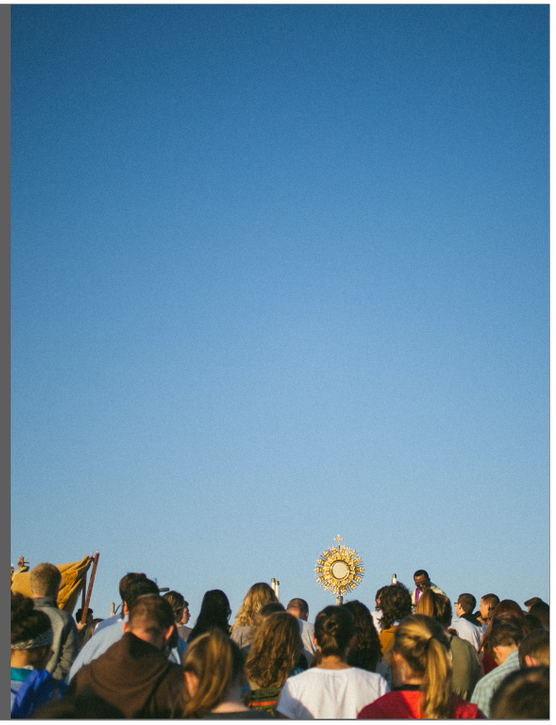
Approach updates cautiously and if you introduce breaking changes talk to your consumers in advance, give people plenty of notice and consider whether you need to implement some sort of namespacing so you can provide the new functionality without breaking existing use cases. This is why you sometimes see API URL paths like "https://example.com/api/v2/predict" the “v2” in there is for version 2 and means there was probably a version 1 API at some point that is now deprecated.

Having a documented depreciation strategy can be really useful here too.

Photo by Denny Müller https://unsplash.com/@redaquamedia

PUBLIC APIS

sellorm.com



Creating public facing APIs can be a business defining move, but it’s a process that requires much planning and support from numerous groups across the business. Approach with caution as all sorts of technical, security and legal challenges are ahead.

Photo by Rachel Moore https://unsplash.com/@racheljomoore

SECURING YOUR API

sellorm.com

It’s a huge topic and this isn’t a security conference, so we’re not going to go too in depth here
Just don’t publish anything on the open web without giving it extensive investigation and ensure you understand what you’re doing. Again, it will likely be necessary to engage with other groups across the business, especially if you want to secure a public facing API.

Image: Photo by John Salvino https://unsplash.com/@jsalvino

MONETISING APIS

sellorm.com

An extension of the idea of creating public APIs. Many of your organisations are sitting on data or insights that could be broadly useful to the rest of the industry. You could sell access to the data or insight via a public facing API.

Again, not to be entered into lightly.

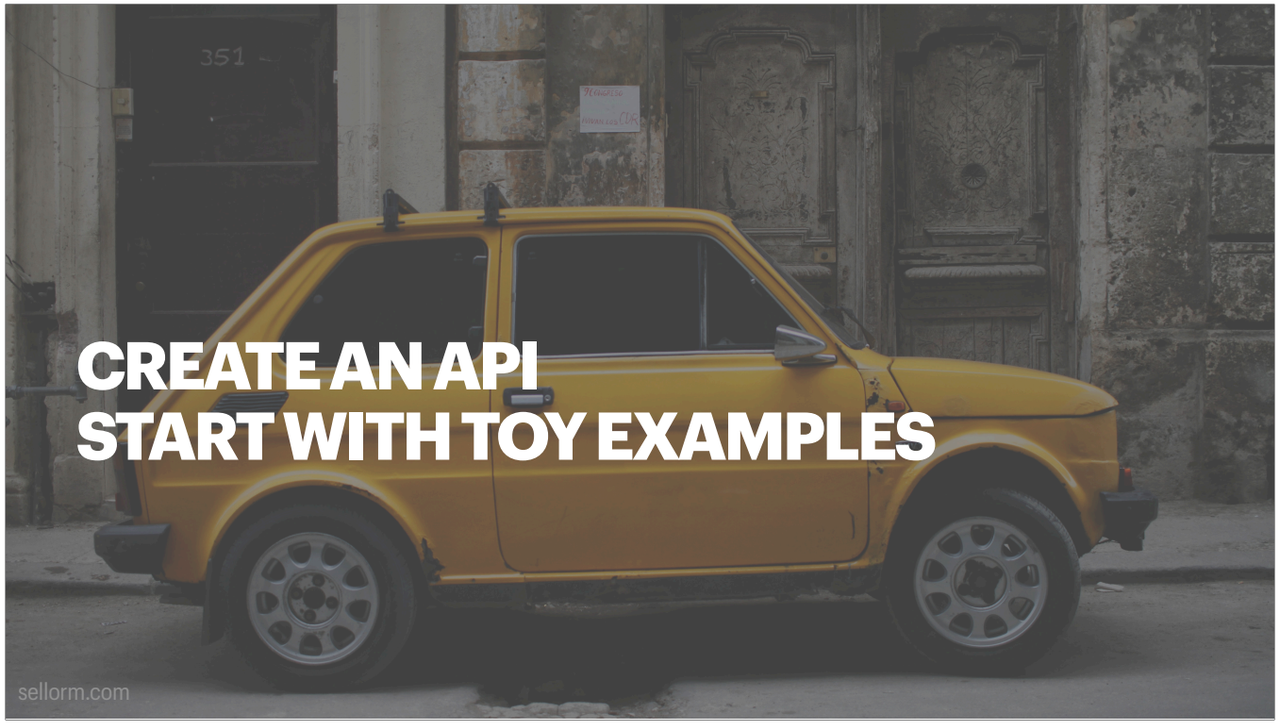
Photo by micheile henderson https://unsplash.com/@micheile

OK, so we’re pretty much at the end now.

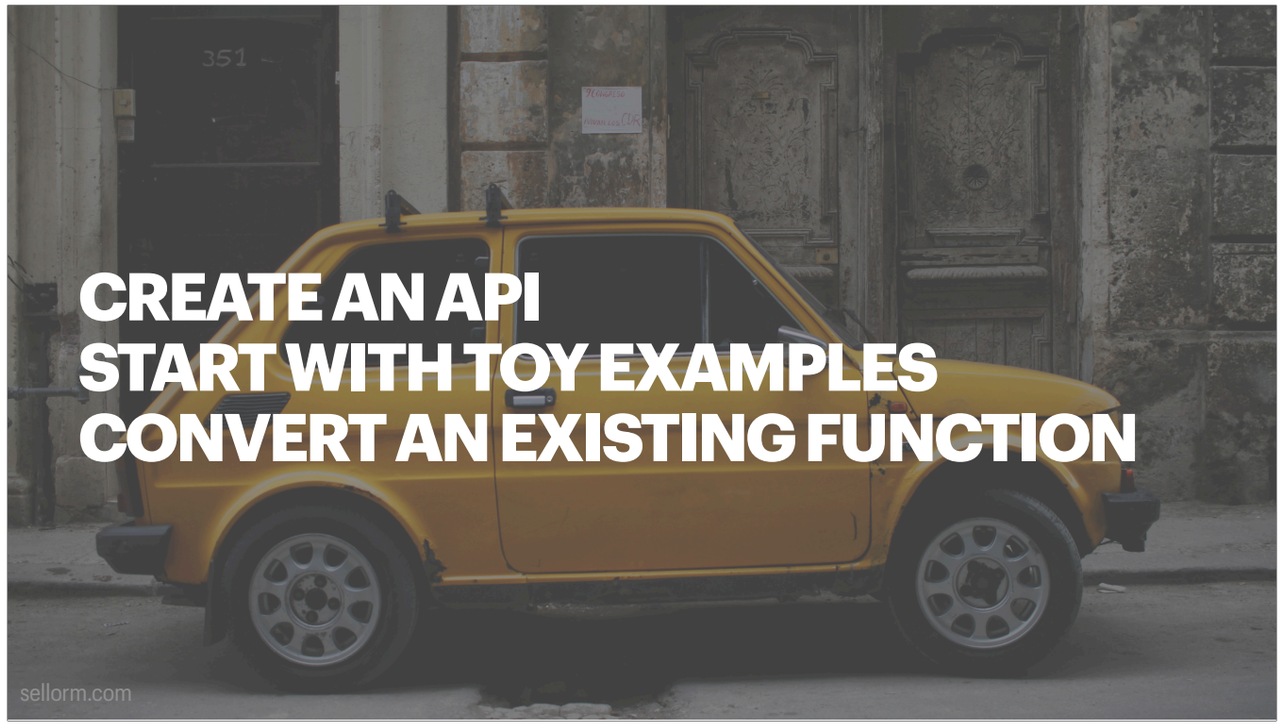
This little Fiat 126 is a great example of keeping things small but functional.

Photo by Jessica Knowlden https://unsplash.com/@mybibimbaplife

I would love for you to go away from this talk inspired to learn more about APIs.



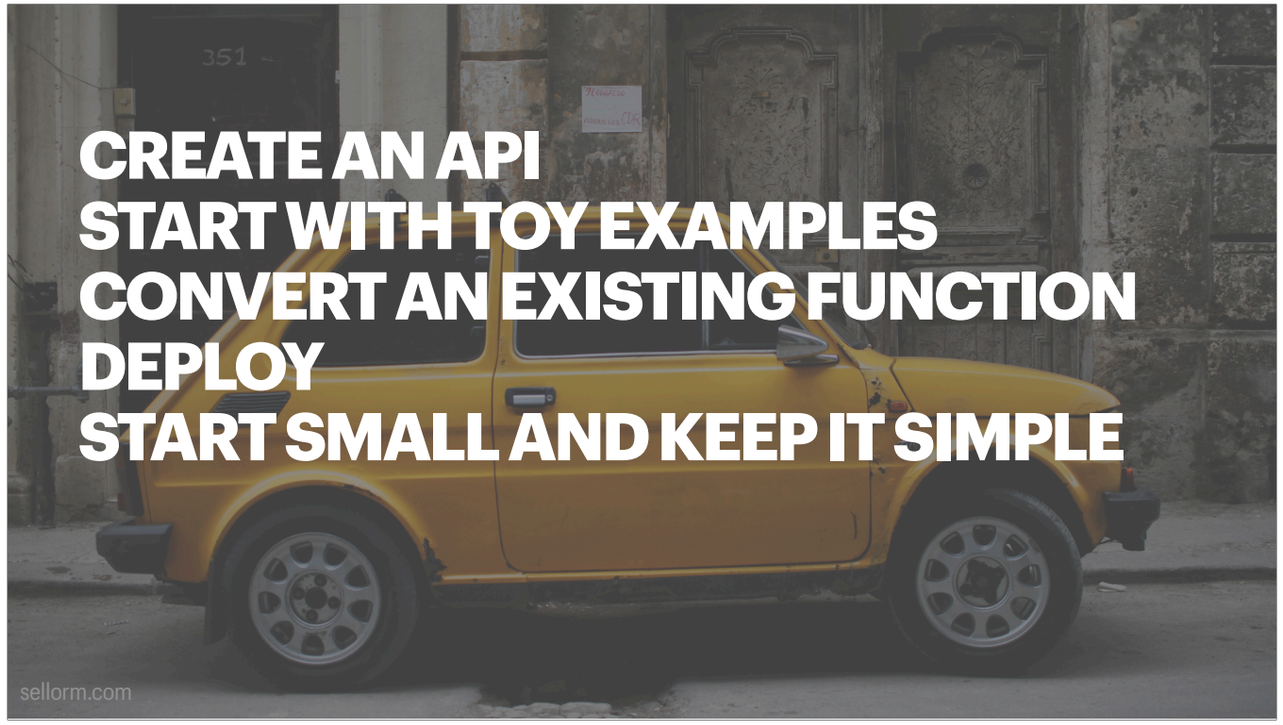
Remember to start with the toy examples that are likely in the documentation for your framework of choice.



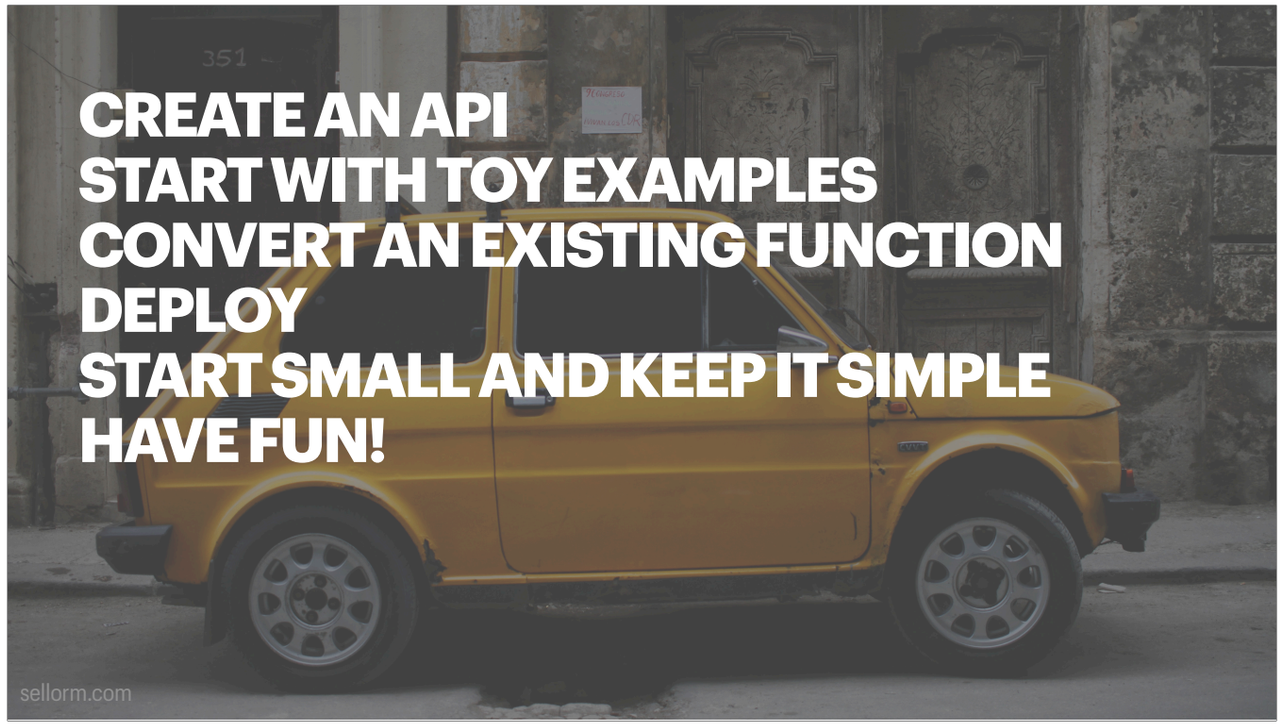
Once you’re comfortable with how a toy API works, convert an existing function of yours to an API.

Get a feel for how it behaves. How does it compare to other APIs you might have consumed yourself?

Deploy your API somewhere using whatever deployment infrastructure you already have available to you if at all possible.



Remember to keep things small and simple initially and to use what you already have available to you.



But most of all, have fun.

Learning about APIs is a huge topic and you won’t get it all straight away, so play around. Learn what works and what doesn’t and build up from there.

THANKS

hello@sellorm.com