4th Annual

MLOPS WORLD CONFERENCE & EXPO

OCTOBER 25-26

DAGWORKS

Getting Higher ROI on MLOps Initiatives: Five Lessons Learned While Building Out the MLOps Platform for 100+ Data Scientists

Stefan Krawczyk - DAGWorks Inc.

whoami

Stefan Krawczyk
Co-creator of **Hamilton** &&
CEO **DAGWorks** Inc.

12+ years in ML & Data platforms

STITCH FIX

iDiBON

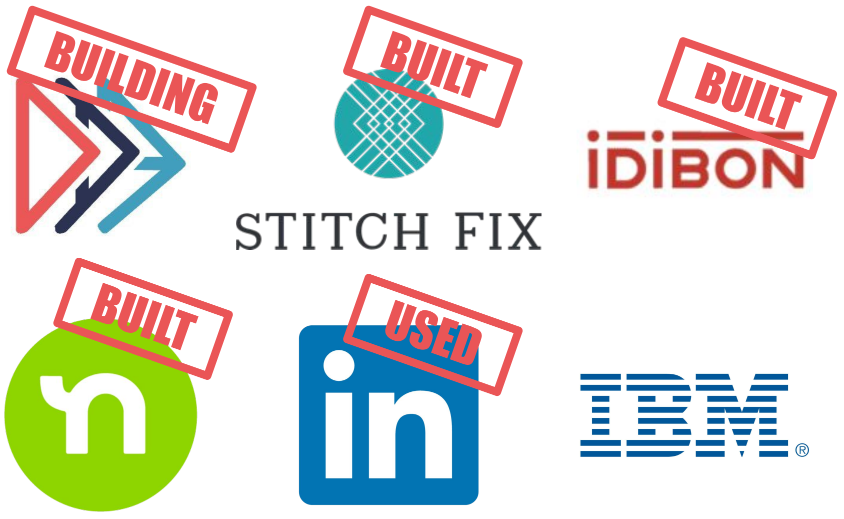
IBM®

VICTORIA UNIVERSITY OF
WELLINGTON
TE HERENGA WAKA

whoami

Stefan Krawczyk
Co-creator of **Hamilton** &&
CEO **DAGWorks** Inc.

12+ years in ML & Data platforms



whoami

Stefan Krawczyk
Co-creator of **Hamilton** &&
CEO **DAGWorks** Inc.

12+ years in ML & Data platforms

100+ DS

BUILDING

BUILT

STITCH FIX

BUILT

iDiBON

BUILT

USED

IBM

VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

For me:
*Ops Initiative == Platform initiative

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Some reasons:

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Some reasons:

1. Repetitive tasks

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Some reasons:

1. Repetitive tasks
2. No standardization

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Some reasons:

1. Repetitive tasks
2. No standardization
3. Duplication of work across teams

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Some reasons:

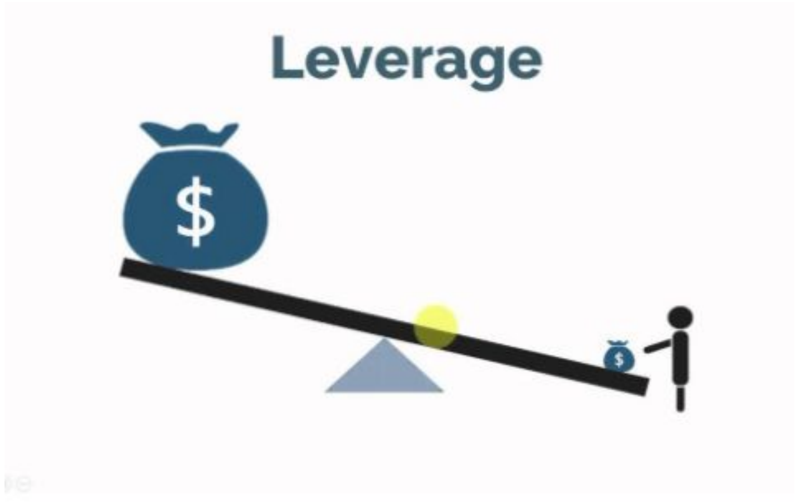
1. Repetitive tasks
2. No standardization
3. Duplication of work across teams
4. Fragmented workflow experience

Why *Ops/Platform?

Delivering sustained value over time is hard without good abstractions

Some reasons:

1. Repetitive tasks
2. No standardization
3. Duplication of work across teams
4. Fragmented workflow experience



Let's increase your ROI*

5 lessons

Lesson breakdown:

1. Users:
   a. Adoption
   b. Sophisticated Users
2. What to build:
   a. Product Management
3. Technical approaches:
   a. Vendor APIs
   b. API Layers

Users

Lesson 1 & Lesson 2

Lesson 1.
Focus on Adoption, Not Completeness

VS

Lesson 1. Focus on Adoption, Not Completeness

My tactics for adoption:

1. Adopt existing user tooling
2. Partner closely with a team and a specific use case

VS

Tactic 1: Adopt existing user tooling

e.g. someone’s internal abstraction/script, etc.

Tactic 1: Adopt existing user tooling

e.g. someone's internal abstraction/script, etc.

Perfect case is Team B asking Team A for that script/tool/abstraction.

Tactic 1: Adopt existing user tooling

e.g. someone's internal abstraction/script, etc.

Perfect case is Team B asking Team A for that script/tool/abstraction.

Why is this a good idea?

- Derisked product; you have a defacto users.
- Value to business should be proven.

Tactic 1: Adopt existing user tooling

e.g. someone's internal abstraction/script, etc.

Perfect case is Team B asking Team A for that script/tool/abstraction.

Why is this a good idea?

- Derisked product; you have a defacto users.
- Value to business should be proven.

Caveats:

- Must see bigger picture.
- Some people don't like giving things up.

Tactic 2:
Partner closely with a team for a specific use case

Tactic 2:
Partner closely with a team for a specific use case

Ideals:

- Narrow use case.
- That team needs it; has a deadline.
- Can incrementally deliver to bring them along.

Tactic 2:
Partner closely with a team for a specific use case

Ideals:

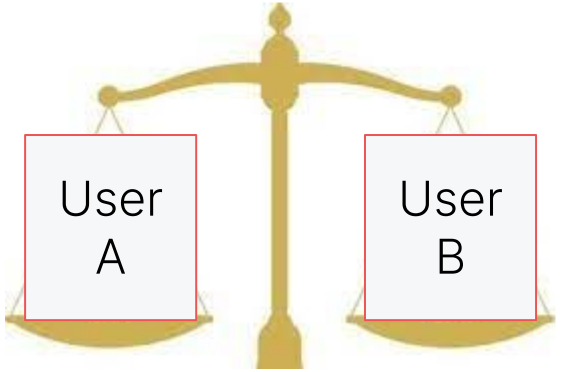
- Narrow use case.
- That team needs it; has a deadline.
- Can incrementally deliver to bring them along.

Goal:

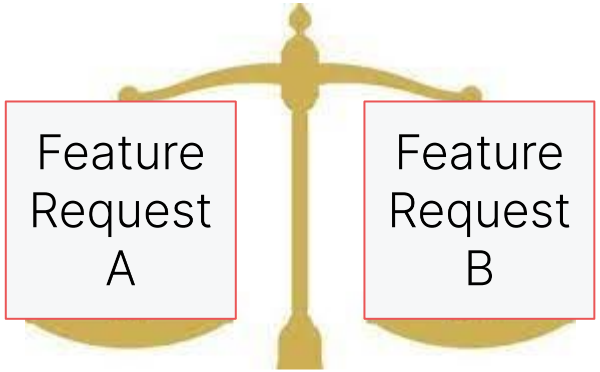
- You have users
- Users see business value

Lesson 2.
Your Users are Not All Equal

It’s tempting to think like this:



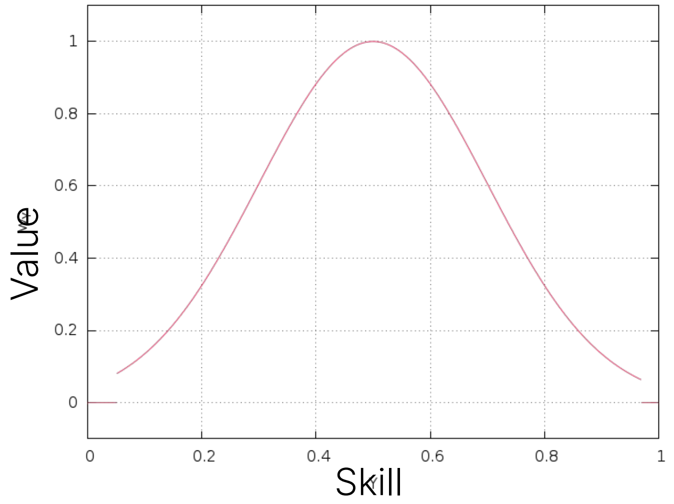
Value to you

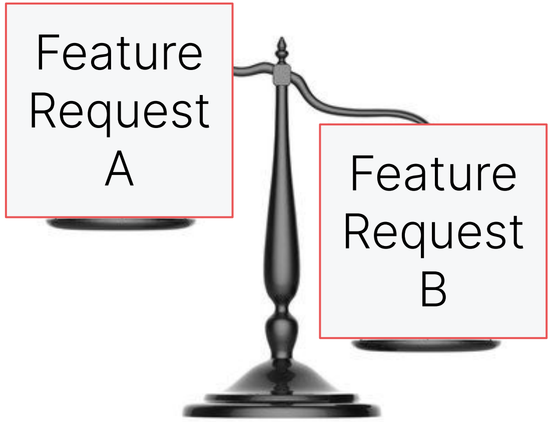


Burden on Platform

Two facts

1. Users fall on a spectrum.
2. Requests aren't equal in development or maintenance costs.





→ Don’t be egalitarian

Don’t be egalitarian

No is probably a good answer when:

1. It’s speculative work and on the periphery of the business.
2. The user is sophisticated and they’re asking for something complex.

Don’t be egalitarian

<u>No</u> is probably a good answer when:

1. It’s speculative work and on the periphery of the business.
2. The user is sophisticated and they’re asking for something complex.

If “the ask” leads to failure:

- No investment spent by you.

Don’t be egalitarian

No is probably a good answer when:

1. It’s speculative work and on the periphery of the business.
2. The user is sophisticated and they’re asking for something complex.

If “the ask” leads to failure:

- No investment spent by you.

If “the ask” leads to success:

- Can then plan to adopt it.

What to build

Lesson 3

Lesson 3.
Live Your Users' Life Cycle

Directionally:

Q: How do you know where to invest?

Directionally:

Q: How do you know where to invest?

Q: Do you know how your work impacts users?

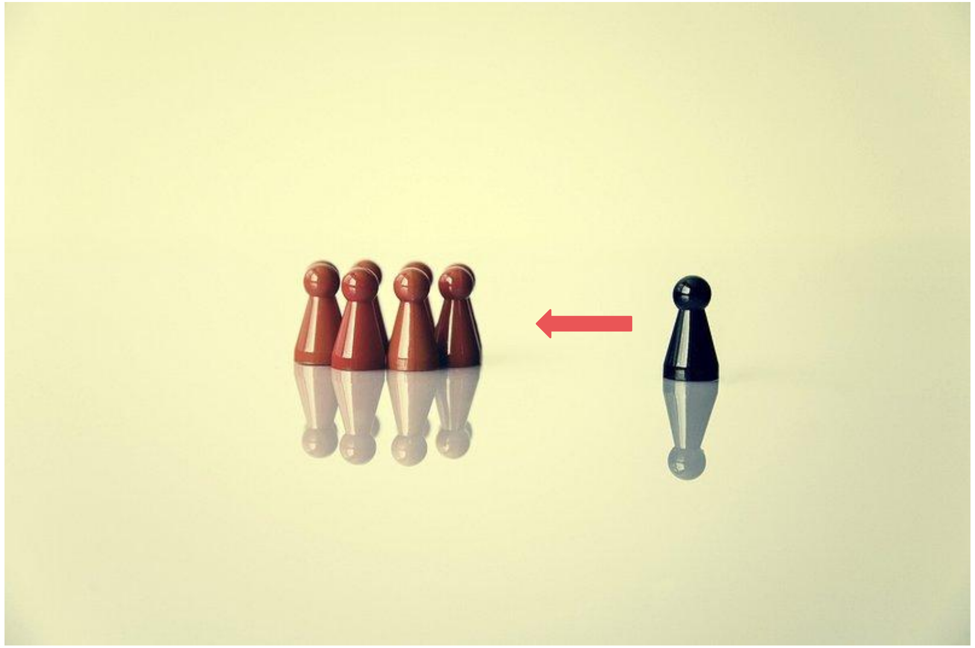
→ Build Empathy

Build Empathy

1. Drink your own champagne / eat your own dog food.

Build Empathy

1. Drink your own champagne / eat your own dog food.
2. Bring in an end user.



Build Empathy

1. Drink your own champagne / eat your own dog food.
2. Bring in an end user.
3. Build relationships.

Technical Approaches

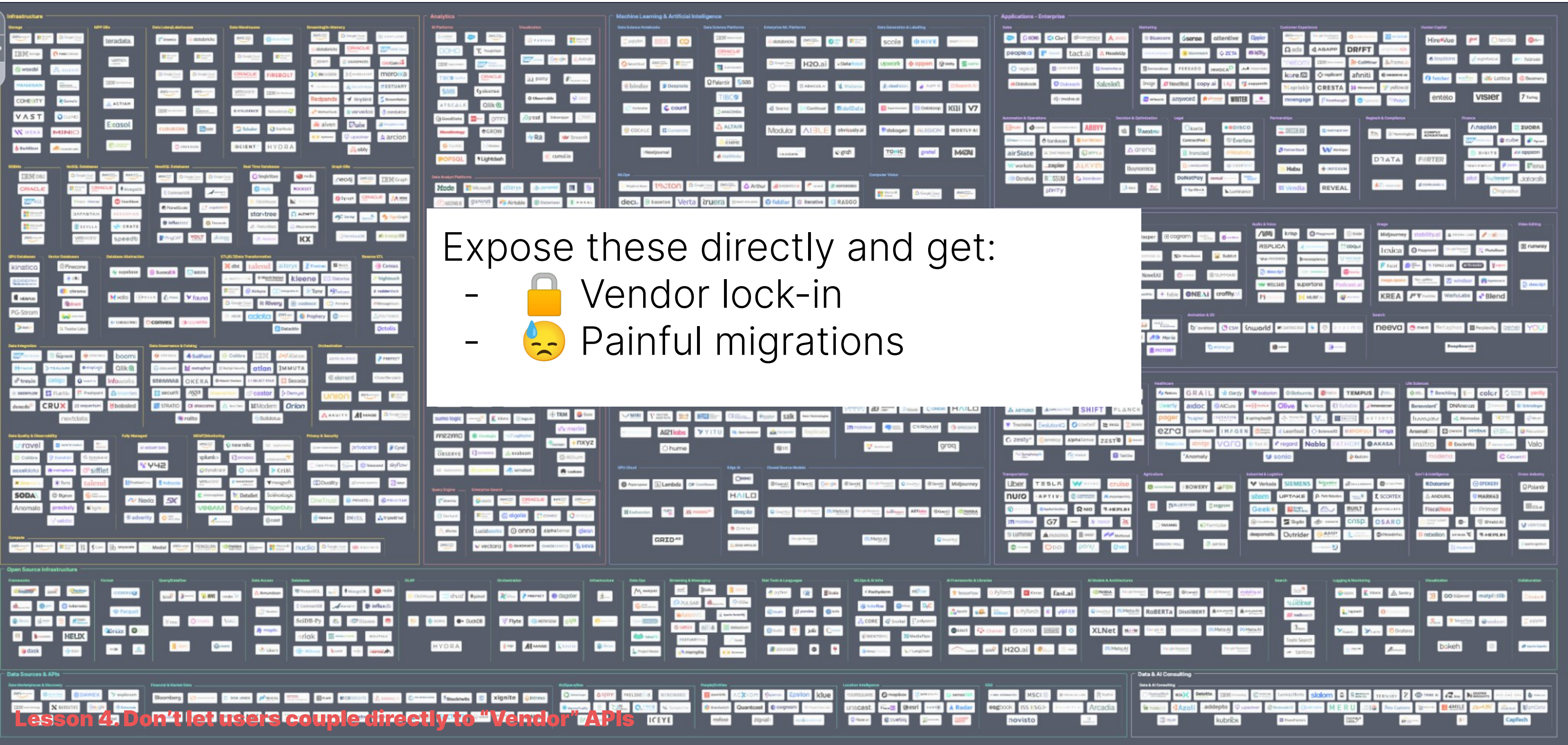
Lesson 4 & Lesson 5

Lesson 4.
Don’t let users couple directly to “Vendor” APIs

Things you could integrate with:

Lesson 4: Don't let users couple directly to "Vendor" APIs

Things you could integrate with:

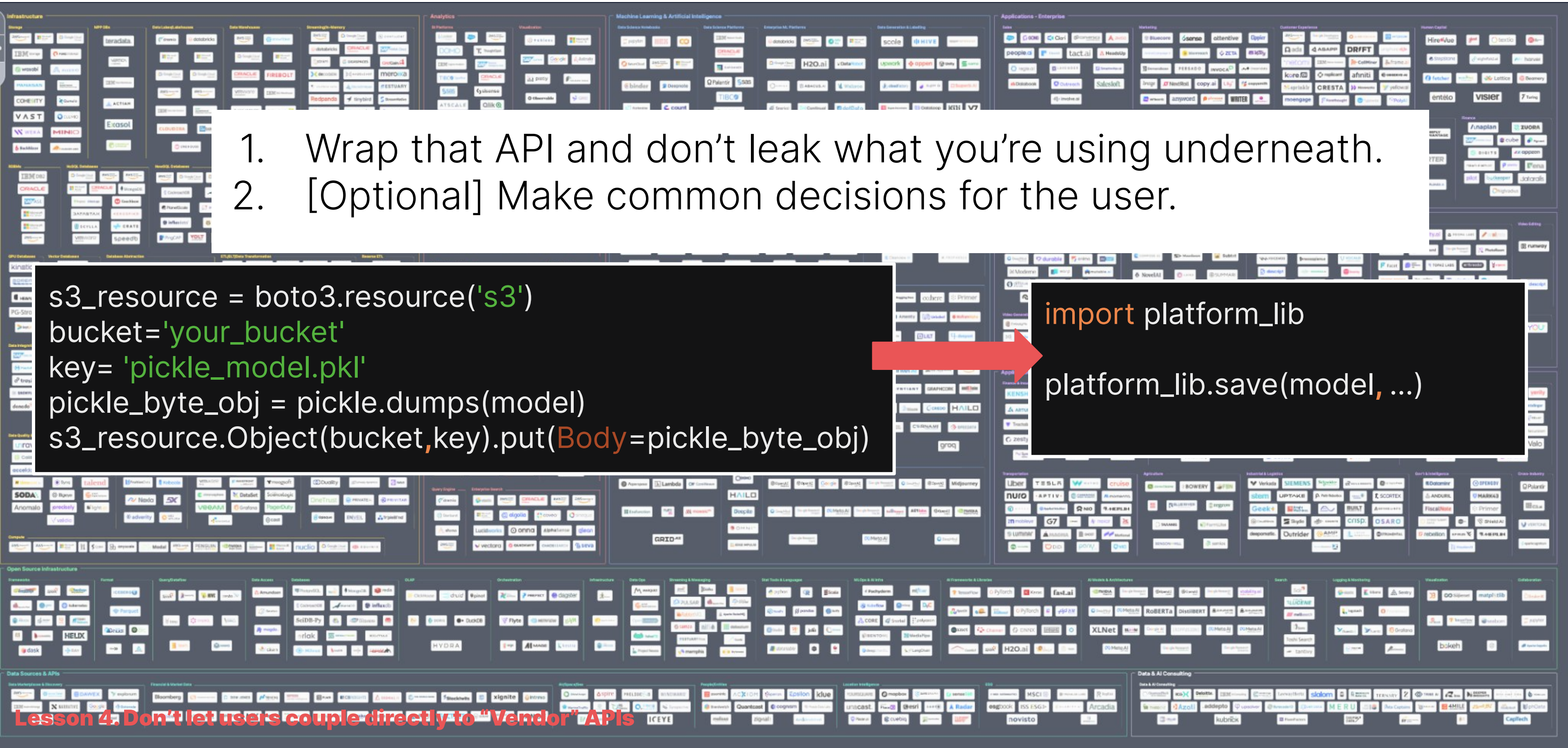


Expose these directly and get:

- 🔒 Vendor lock-in
- 😓 Painful migrations

Lesson 4: Don't let users couple directly to "Vendor" APIs

What you should do instead



1. Wrap that API and don’t leak what you’re using underneath.
2. [Optional] Make common decisions for the user.

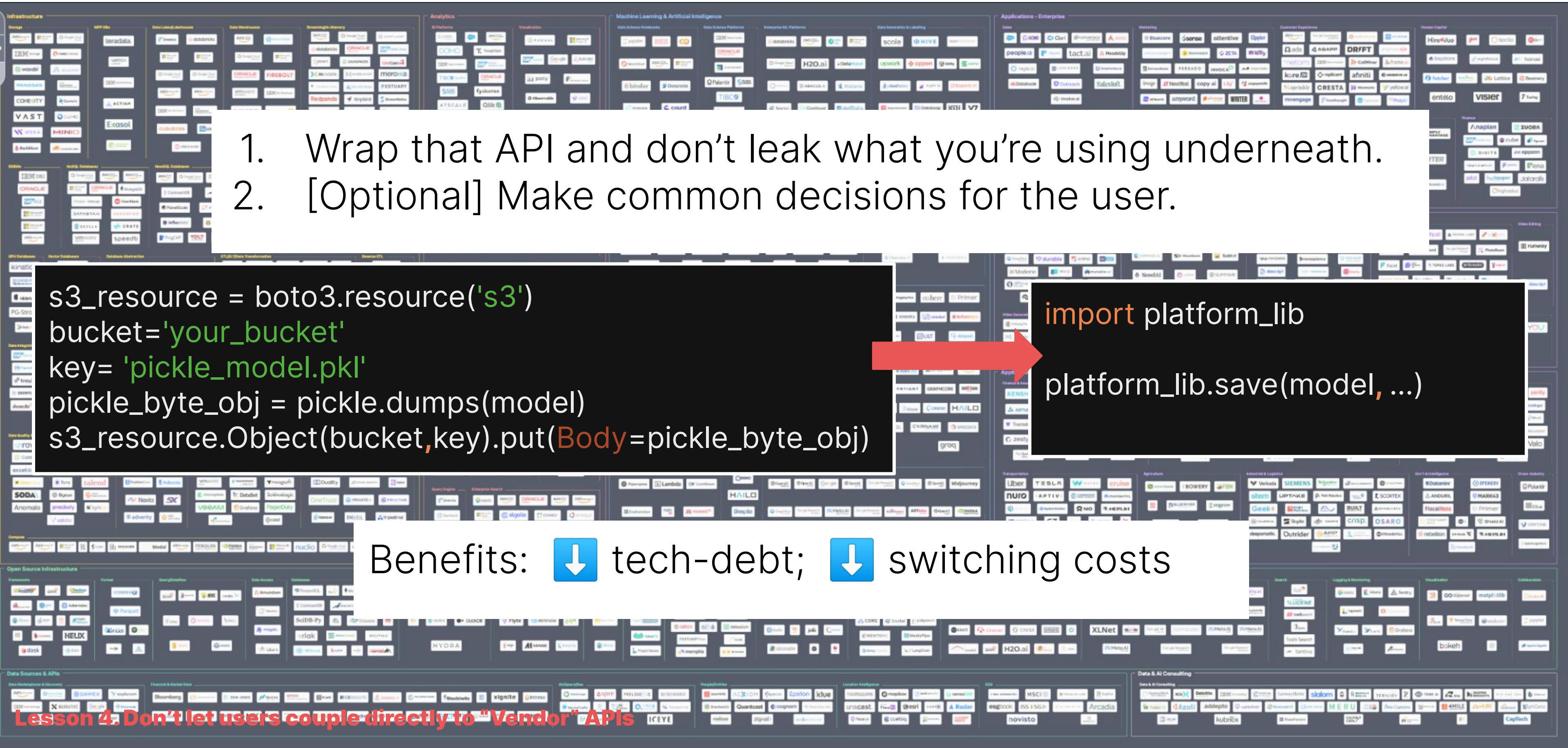
```python
s3_resource = boto3.resource('s3')
bucket='your_bucket'
key= 'pickle_model.pkl'
pickle_byte_obj = pickle.dumps(model)
s3_resource.Object(bucket,key).put(Body=pickle_byte_obj)
```

```python
import platform_lib

platform_lib.save(model, ...)
```

Lesson 4: Don’t let users couple directly to “Vendor” APIs

What you should do instead



1. Wrap that API and don’t leak what you’re using underneath.
2. [Optional] Make common decisions for the user.

```python
s3_resource = boto3.resource('s3')
bucket='your_bucket'
key= 'pickle_model.pkl'
pickle_byte_obj = pickle.dumps(model)
s3_resource.Object(bucket,key).put(Body=pickle_byte_obj)
```

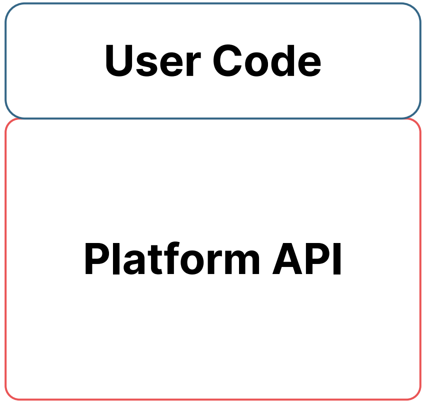
```python
import platform_lib

platform_lib.save(model, ...)
```

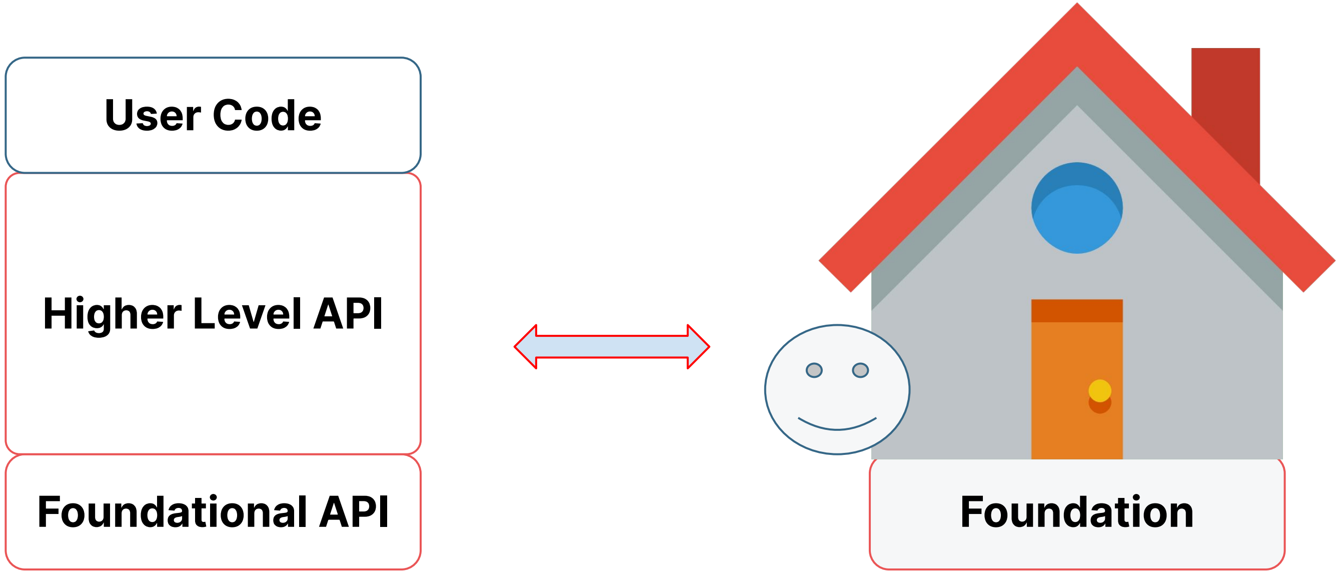
Benefits: ⬇️ tech-debt; ⬇️ switching costs

Lesson 5.
The Two Layer API Trick

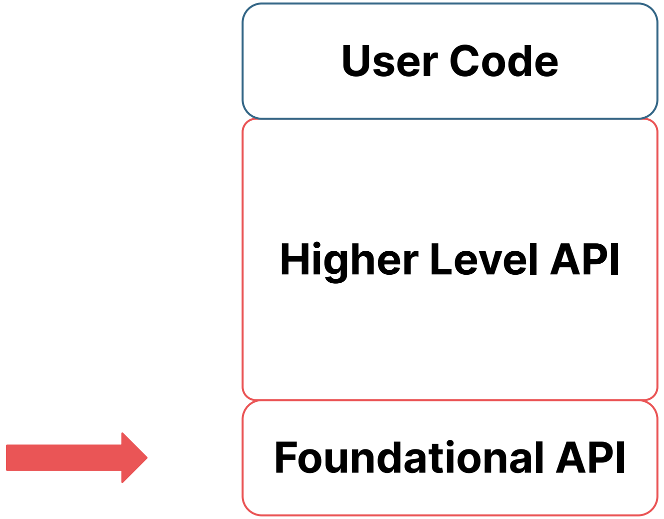
Common Approach:



Two Layer API Trick



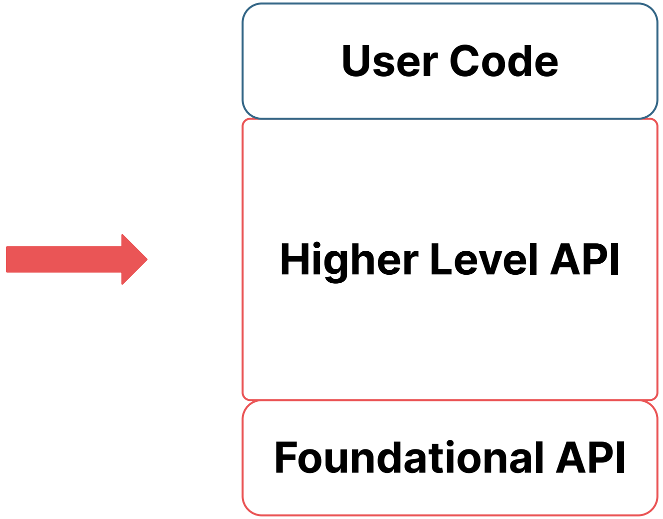
Bottom API Layer



E.g. what you want to expose on top of “Vendors”.

- Allows anyone to build anything, but in a bounded way.
- Primary user is your team.

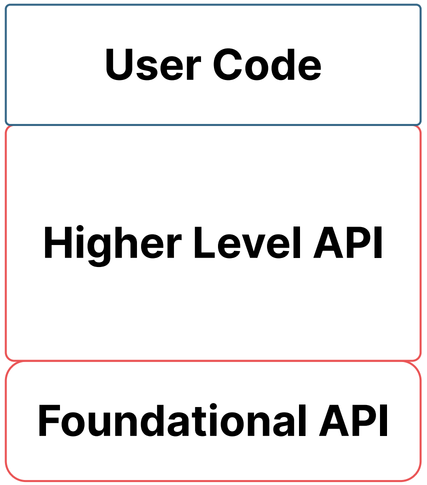
Top API Layer



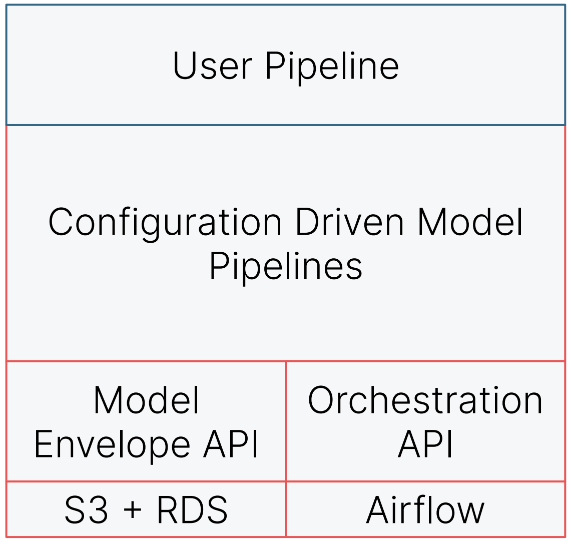
- Main API for users.
- Goal is to simplify the experience.
- Built solely off of Foundational API.

E.g. one line to save and deploy a model,
one line to save a prompt, etc.

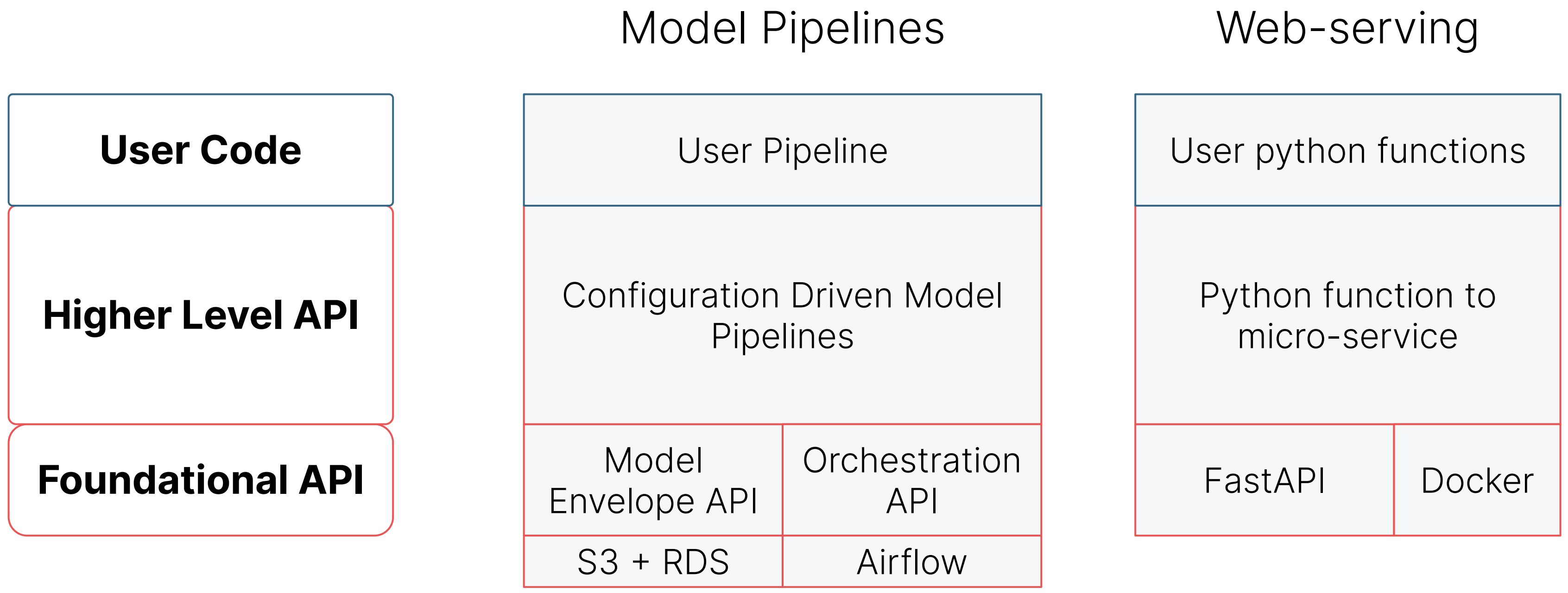
Two Layer API Examples



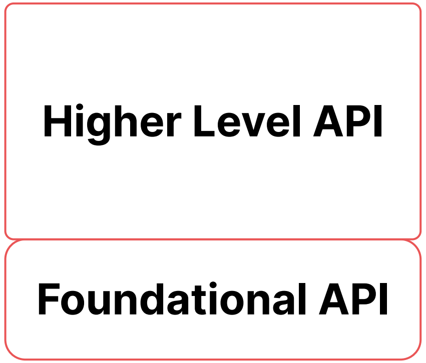
Model Pipelines



Two Layer API Examples



Two Layer API Benefits



- 🥷 You can be more nimble.
- ⬇️ Coupling & ⬇️ tech-debt maintenance.
- 🤠 Provide escape-hatch for sophisticated users.
- 😌 Simpler APIs reduce time to value.

Summary:
Getting more ROI on your MLOPs (& LLMOps) initiatives

Summary: Getting more ROI on your initiatives

1. Build for immediate adoption → *show value sooner.*

Summary: Getting more ROI on your initiatives

1. Build for immediate adoption → *show value sooner.*
2. Don't build for every user equally → *use time more effectively.*

Summary: Getting more ROI on your initiatives

1.	Build for immediate adoption	→ *show value sooner.*
2.	Don’t build for every user equally	→ *use time more effectively.*
3.	Build empathy	→ *know what is impactful.*

Summary: Getting more ROI on your initiatives

1. Build for immediate adoption → *show value sooner.*
2. Don’t build for every user equally → *use time more effectively.*
3. Build empathy → *know what is impactful.*
4. Wrap vendor/cloud APIs → ⬇️ *technical debt;* ⬇️ *switching costs*

Summary: Getting more ROI on your initiatives

1. Build for immediate adoption → *show value sooner.*
2. Don't build for every user equally → *use time more effectively.*
3. Build empathy → *know what is impactful.*
4. Wrap vendor/cloud APIs → ⬇️ *technical debt;* ⬇️ *switching costs*
5. Provide two layers of APIs → ⬇️ *technical debt;* ⬆ *iteration speed;* ⬇️ *time to value for a user*
    a. foundational layer.
    b. opinionated higher level layer.

Want to see 👀 some of this in action?

https://github.com/DAGWorks-Inc/hamilton

www.dagworks.io

Thanks for listening!

Questions?

Connect with me:

https://twitter.com/stefkrawczyk

https://www.linkedin.com/in/skrawczyk/

https://blog.dagworks.io/