Simplicity Scales:

Rewriting to a Django Monolith and Monorepo

Rayan Daod Nathoo
ML Engineer @ Atinary Technologies

PyCon Italia 2026
Thursday May 28th, 15:20 - 16:05

ATINARY

My Intro - Rayan Daod Nathoo

- Machine Learning Engineer
- Previous experience in ML Research and Software Engineering
- MSc. Computer Science @ EPFL
- rayan.daodnathoo.com

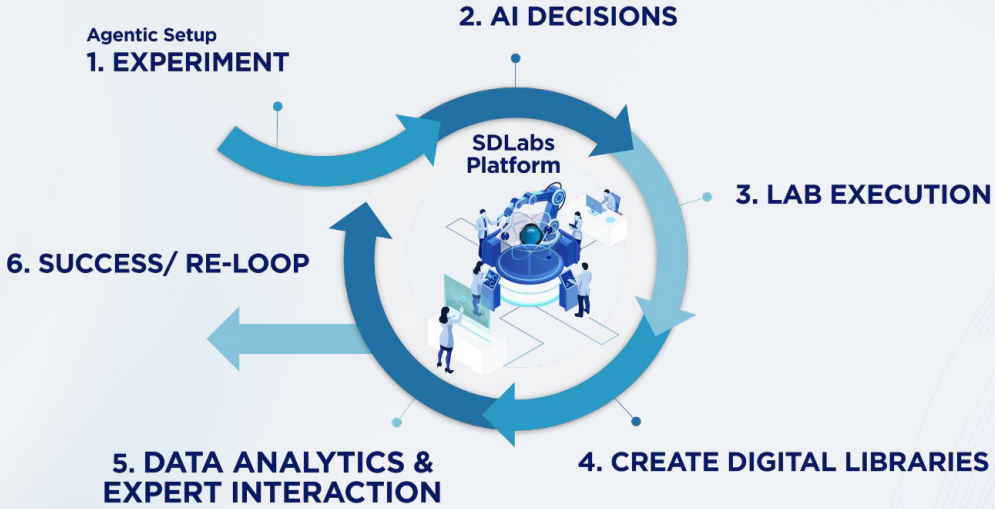
Our Company - Atinary Technologies

- Swiss-American startup founded in 2019
- Augmenting scientists with Physical AI and Robotics to accelerate R&D
- Multi-disciplinary team - ML, Chemistry, Software, …

Team based in Switzerland and US

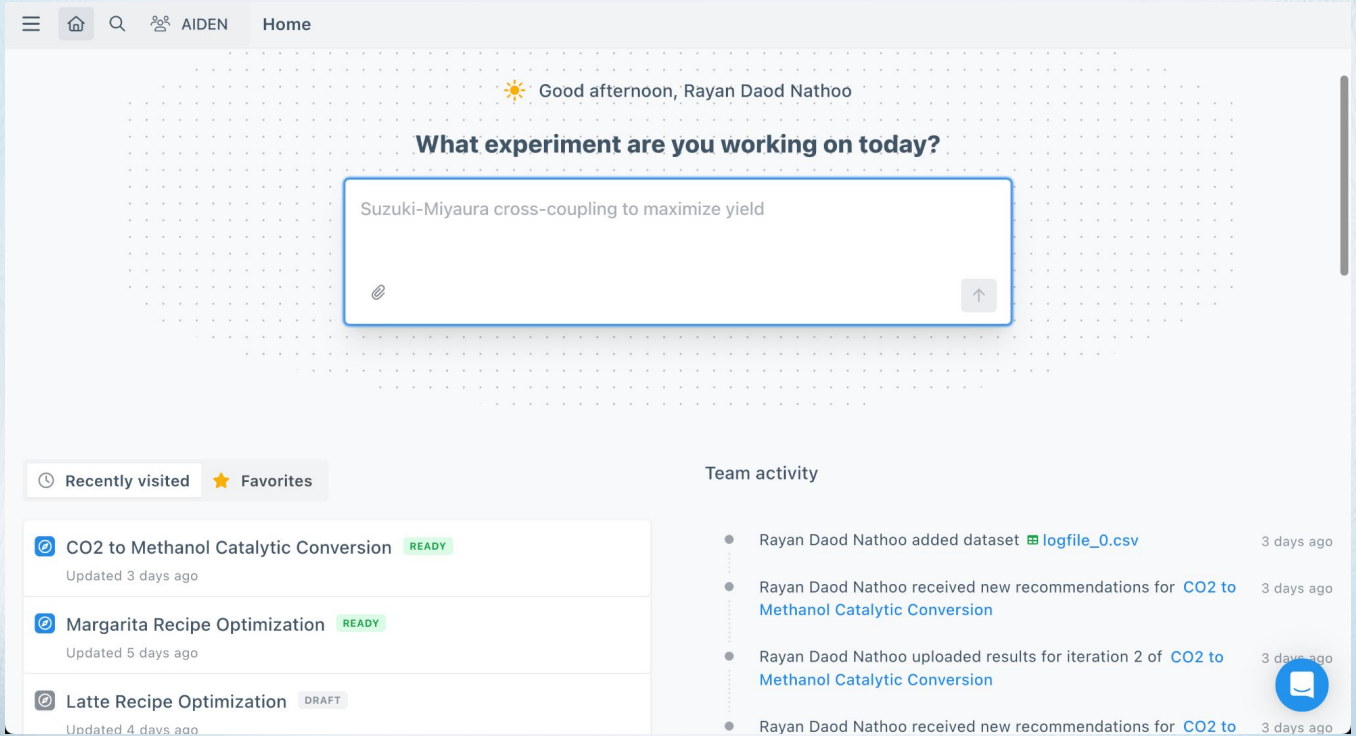
Laboratory in Boston

Our Software – SDLabs



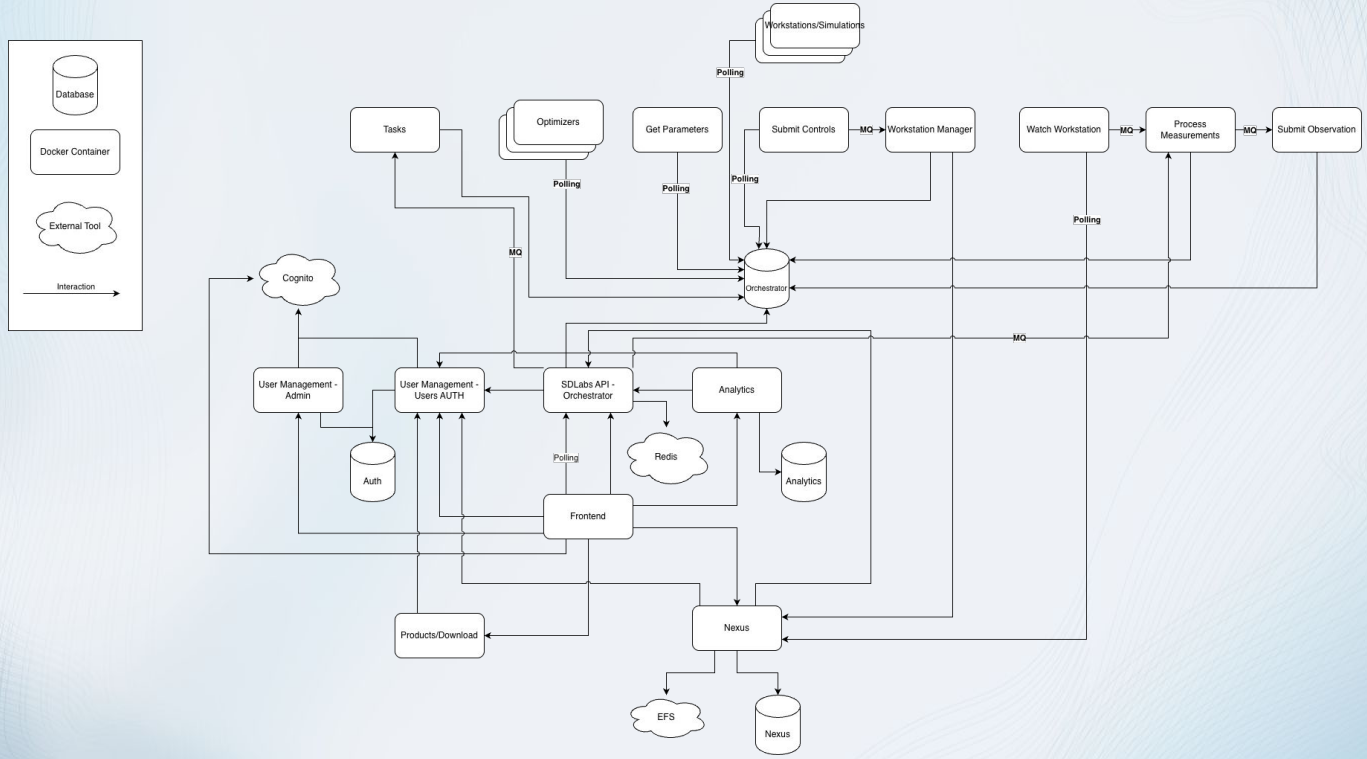
*Design, Make, Test, Analyze, Learn

Our Software – SDLabs

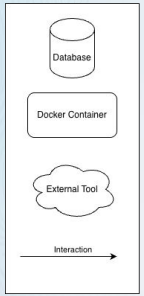


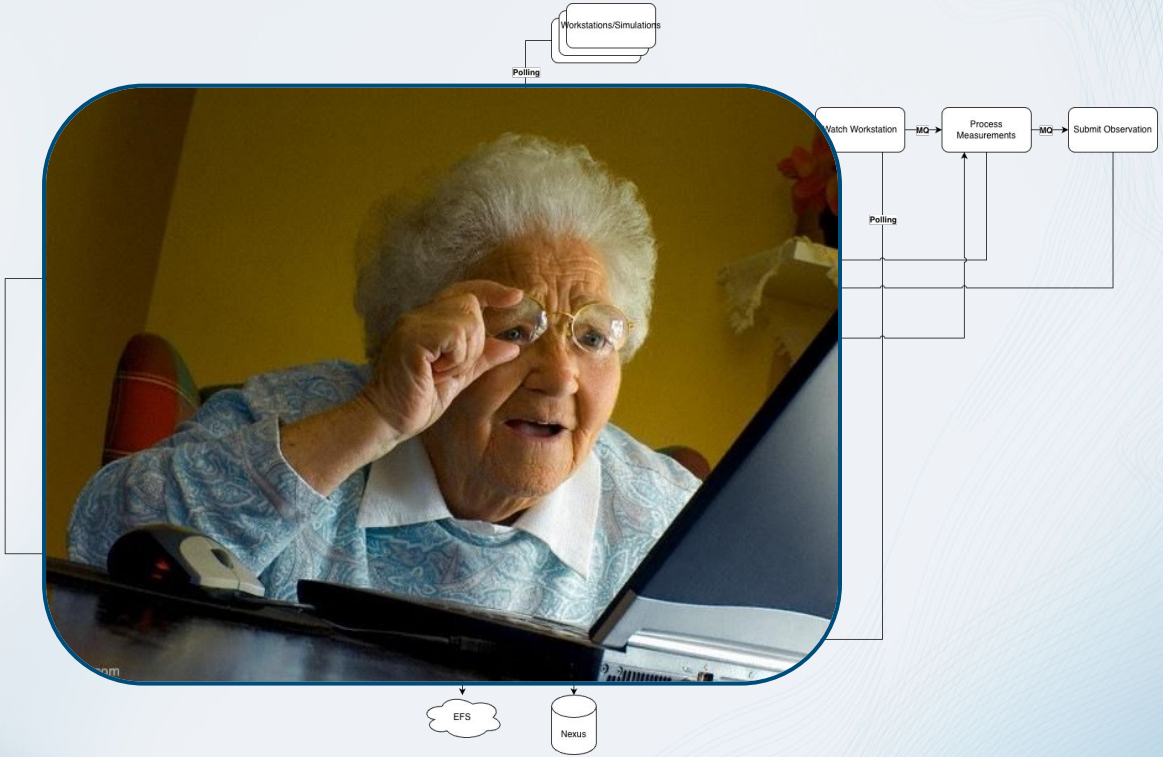
Flashback 1 year ago...

Old Architecture



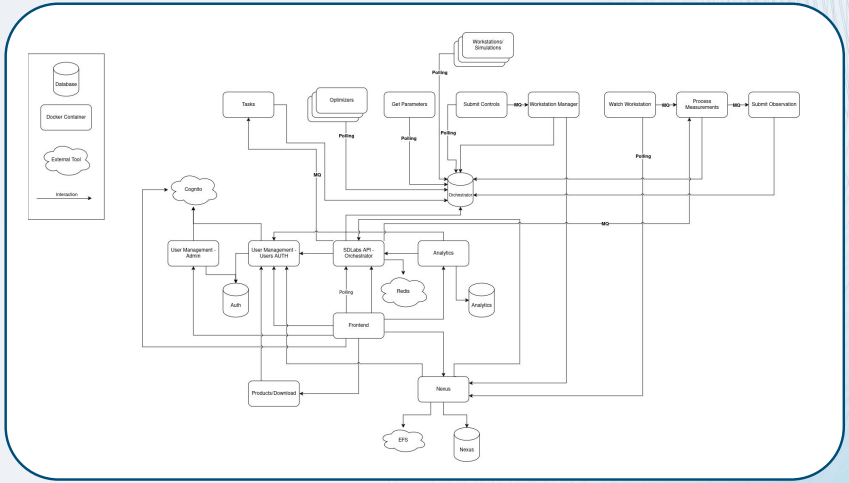
Old Architecture





Old Architecture Issues

- 👎 Complex system
- 👎 Unclear ownership
- 👎 Coupled codebases
- 👎 Hard bug fixing process
- 👎 Hard release process
- 👎 Hard onboarding



Old architecture

We need to do something!

Spoiler Alert!

✅ Simpler system design

✅ Clearer ownership

✅ Easier bug hunts

✅ Faster release process

Infrastructure Cost	- 44%
Feature PRs	+ 55 %
Bug fix PRs	- 31 %

Agenda

1. The Questions We Asked
2. What We Built
    a. The Monolith
    b. The Monorepo
    c. The Local Environment
3. Results & Takeaways

1. The Questions We Asked

Refactor or Rewrite?

Refactor or Rewrite?

Incremental Refactoring

Pros

- Easier testing/rollback
- Preserves embedded knowledge
- Keeps product momentum

Complete Rewrite

Refactor or Rewrite?

Incremental Refactoring

Pros

- Easier testing/rollback
- Preserves embedded knowledge
- Keeps product momentum

Cons

- Decomposition complexity & cognitive cost
- Hard to solve core architecture issue
- Slow results

Complete Rewrite

Refactor or Rewrite?

Incremental Refactoring

Pros

- Easier testing/rollback
- Preserves embedded knowledge
- Keeps product momentum

Cons

- Decomposition complexity & cognitive cost
- Hard to solve core architecture issue
- Slow results

Complete Rewrite

Pros

- Allows architecture changes
- Room for new tech stack
- Allows re-scoping the product

Refactor or Rewrite?

Incremental Refactoring

Pros

- Easier testing/rollback
- Preserves embedded knowledge
- Keeps product momentum

Cons

- Decomposition complexity & cognitive cost
- Hard to solve core architecture issue
- Slow results

Complete Rewrite

Pros

- Allows architecture changes
- Room for new tech stack
- Allows re-scoping the product

Cons

- Maintaining two systems in parallel
- Long time-to-production
- Data migration

Refactor or Rewrite?

Incremental Refactoring

Pros

- Easier testing/rollback
- Preserves embedded knowledge
- Keeps product momentum

Cons

- Decomposition complexity & cognitive cost
- Hard to solve core architecture issue
- Slow results

Complete Rewrite

Pros

- Allows architecture changes
- Room for new tech stack
- Allows re-scoping the product

Cons

- Maintaining two systems in parallel
- Long time-to-production
- Data migration

Refactor or Rewrite?

Incremental Refactoring

Complete Rewrite

Pros

- Easier testing/rollback
- Preserves embedded knowl
- Keeps product momentum

hitecture changes
ew tech stack
scoping the product

Cons

- Decomposition complexity &
- Hard to solve core architect
- Slow results

ng two systems in parallel
-to-production
ation

Rewrite Playbook

Rewrite Playbook

I - Simplify and don’t reinvent the wheel

Rewrite Playbook

I - Simplify and don’t reinvent the wheel

II - Apply clear project management

Rewrite Playbook

I - Simplify and don’t reinvent the wheel

II - Apply clear project management

III - Improve product along the way

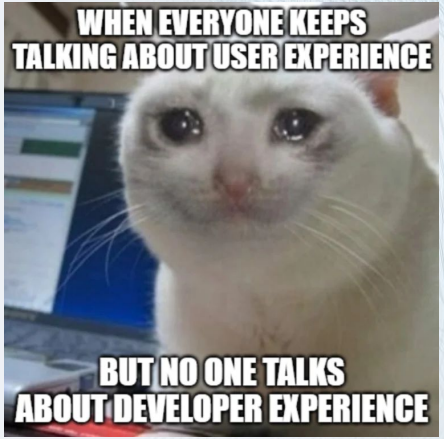
Rewrite Playbook

I - Simplify and don’t reinvent the wheel

II - Apply clear project management

III - Improve product along the way

IV - Prioritize developer experience



Product Requirements

Product Requirements

1. Serving up to 1k customers
    - B2B SaaS, high availability

Product Requirements

1. Serving up to 1k customers
    - B2B SaaS, high availability
2. Flexible integration with internal/external systems
    - ML systems, robots, data sources

Product Requirements

1. Serving up to 1k customers
    - B2B SaaS, high availability
2. Flexible integration with internal/external systems
    - ML systems, robots, data sources
3. Shorter release cycles
    - ~ Every 2 weeks

Do we even keep Python?

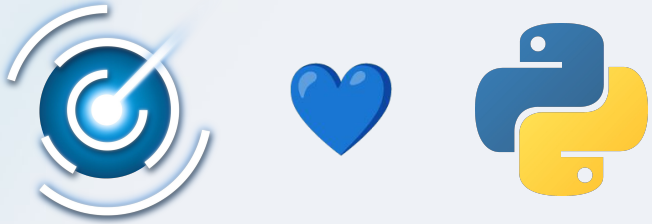
Do we even keep Python?

Yes.

Do we even keep Python?

Yes.

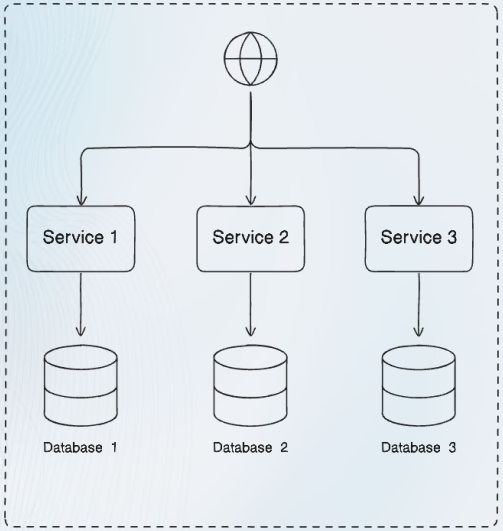
- Existing team already had experience in it
- Plan is to blend in even more ML and DS features in the future



2. What We Built

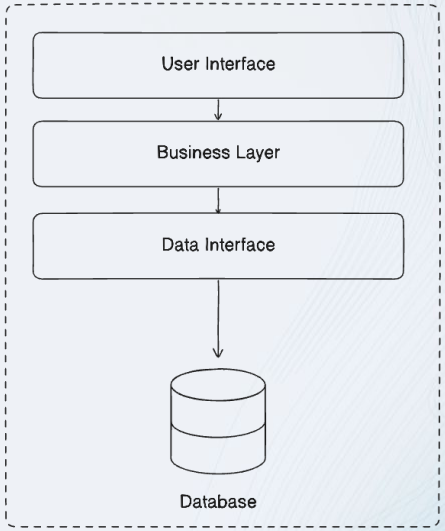
a. The Monolith

Microservices vs. Monolith



Microservices

vs.



Monolith

Microservices vs. Monolith

Microservices

Monolith

Microservices vs. Monolith

Microservices

Pros

- Easy to scale (in theory)
- Flexible in technology stack

Monolith

Microservices vs. Monolith

Microservices

Pros

- Easy to scale (in theory)
- Flexible in technology stack

Cons

- Complexity over time
- Network communication overhead
- Multiple deployments
- Hard to debug

Monolith

Microservices vs. Monolith

Microservices

Pros

- Easy to scale (in theory)
- Flexible in technology stack

Cons

- Complexity over time
- Network communication overhead
- Multiple deployments
- Hard to debug

Monolith

Pros

- Simpler to reason about
- No communication overhead
- Faster deployment process
- Easier to debug

Microservices vs. Monolith

Microservices

Pros

- Easy to scale (in theory)
- Flexible in technology stack

Cons

- Complexity over time
- Network communication overhead
- Multiple deployments
- Hard to debug

Monolith

Pros

- Simpler to reason about
- No communication overhead
- Faster deployment process
- Easier to debug

Cons

- Mistakes can take entire system down
- Cannot scale individual pieces
- Cross-team coordination becomes complex over time

Microservices vs. Monolith

Microservices

Pros

- Easy to scale (in theory)
- Flexible in technology stack

Cons

- Complexity over time
- Network communication overhead
- Multiple deployments
- Hard to debug

Monolith

Pros

- Simpler to reason about
- No communication overhead
- Faster deployment process
- Easier to debug

Cons

- Mistakes can take entire system down
- Cannot scale individual pieces
- Cross-team coordination becomes complex over time

Summary: Architectural Decisions

Summary: Architectural Decisions

Stay in the Python Ecosystem

Summary: Architectural Decisions

Stay in the Python Ecosystem

Move to a Monolith

Summary: Architectural Decisions

Stay in the Python Ecosystem

Move to a Monolith

Choose stable technologies and frameworks

Summary: Architectural Decisions

Stay in the Python Ecosystem

Move to a Monolith

Choose stable technologies and frameworks

Summary: Architectural Decisions

Stay in the Python Ecosystem

Move to a Monolith

Choose stable technologies and frameworks

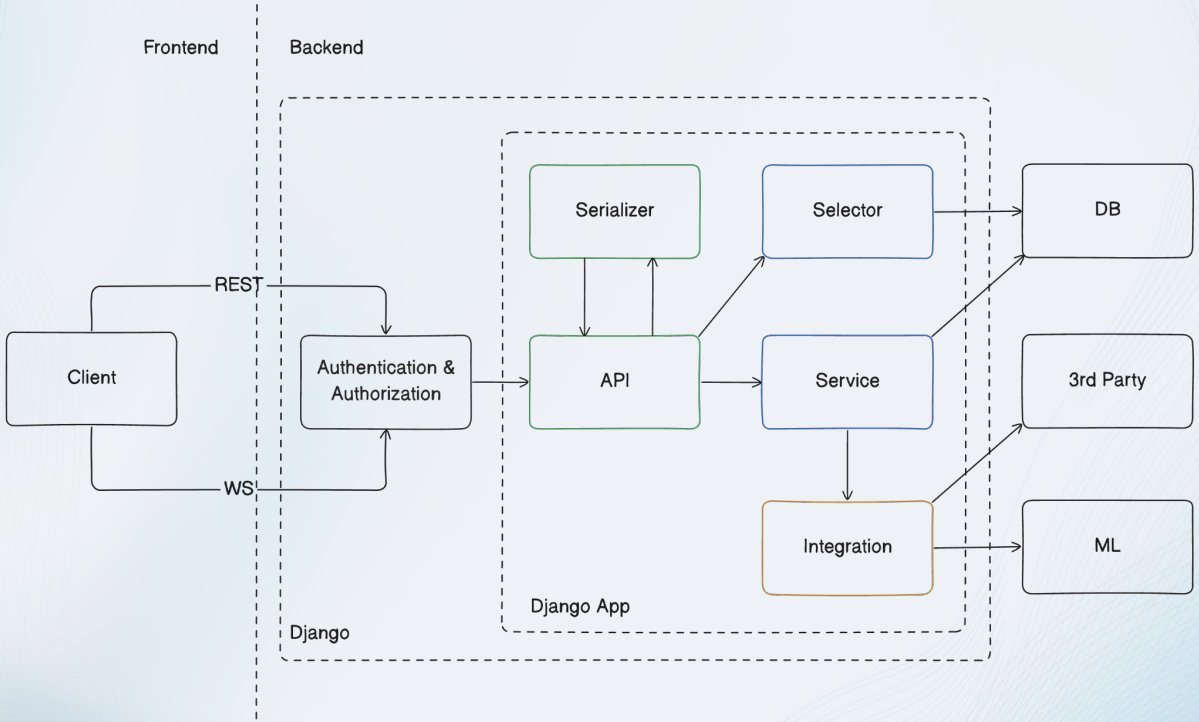
Summary: Architectural Decisions

Stay in the Python Ecosystem

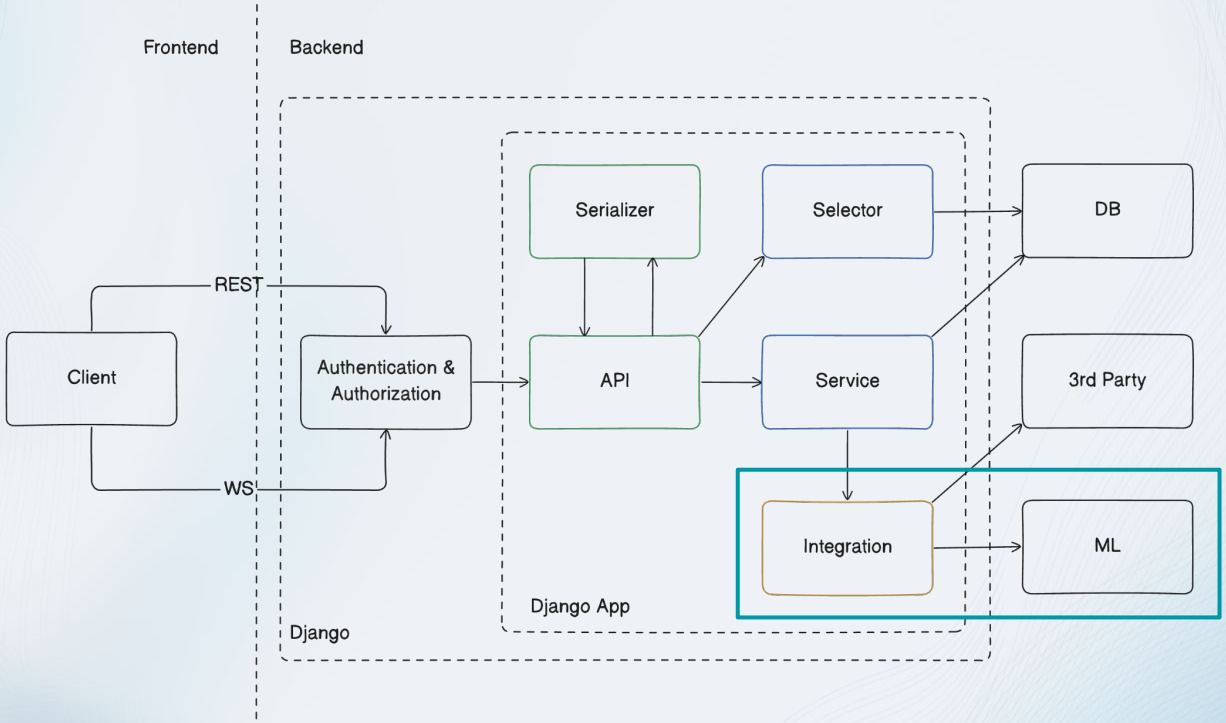
Move to a Monolith

Choose stable technologies and frameworks

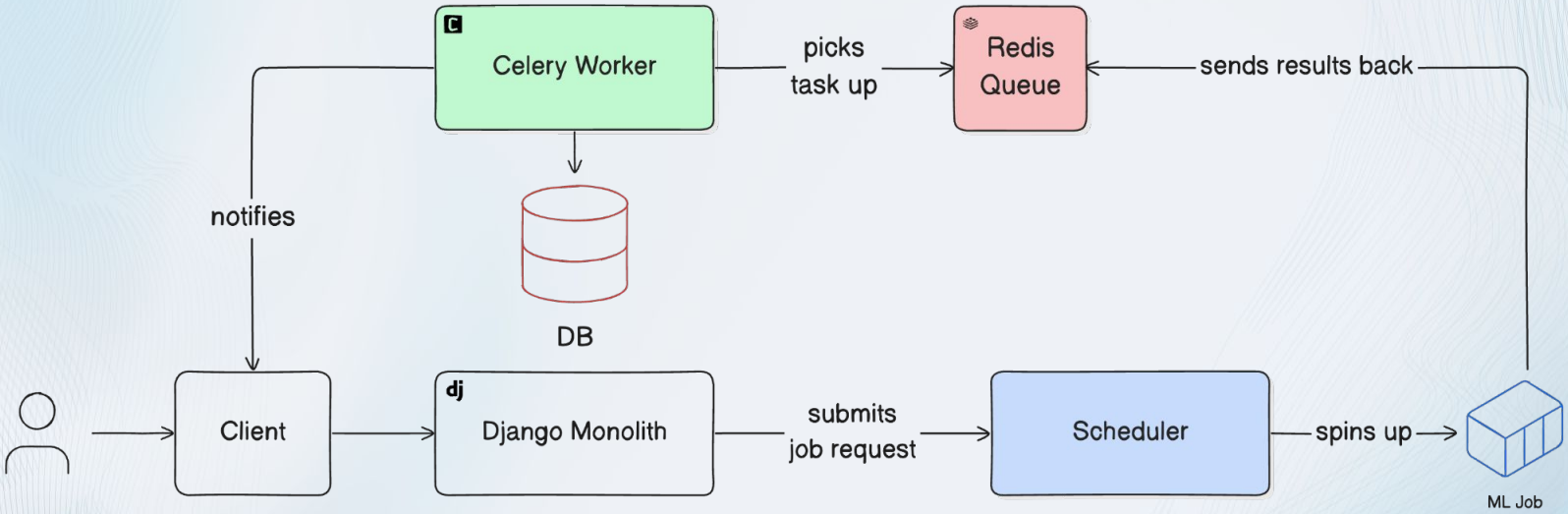
Django Monolith Architecture



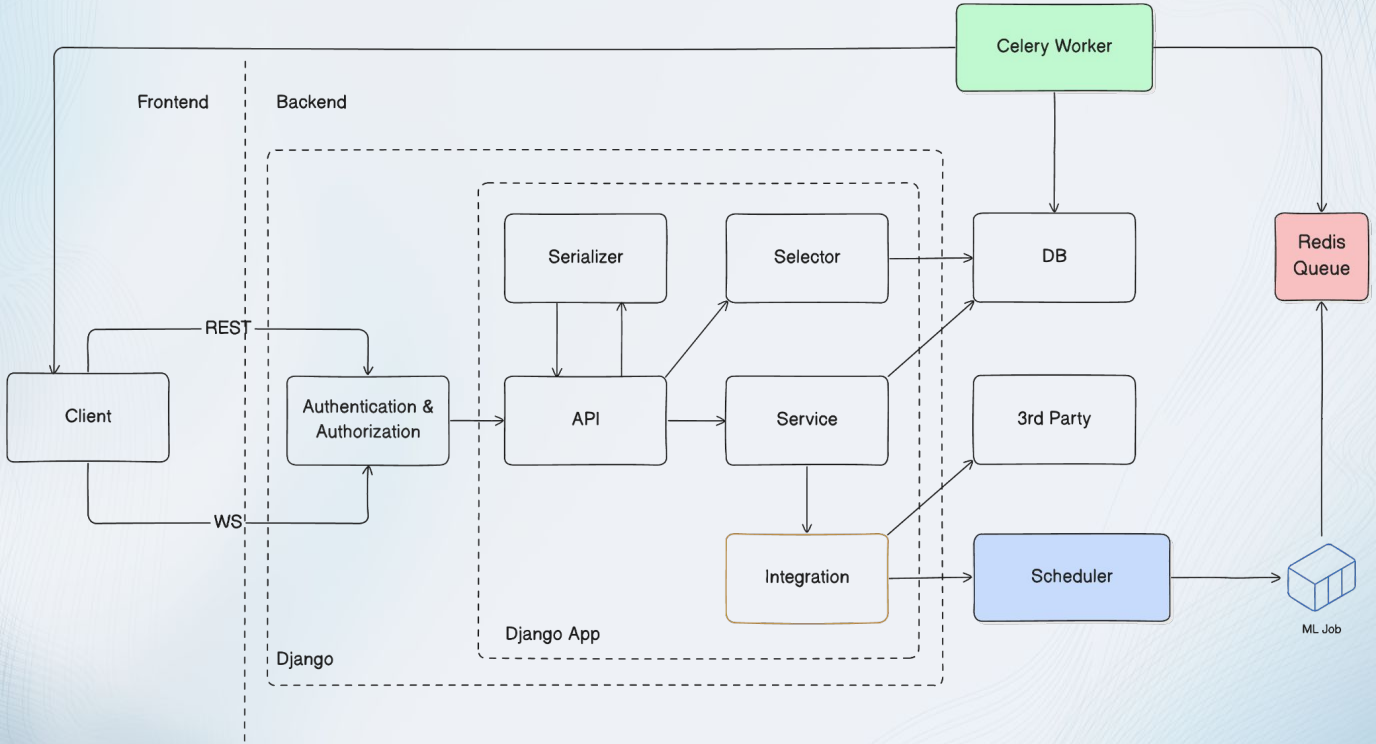
Django Monolith Architecture



Asynchronous ML Jobs

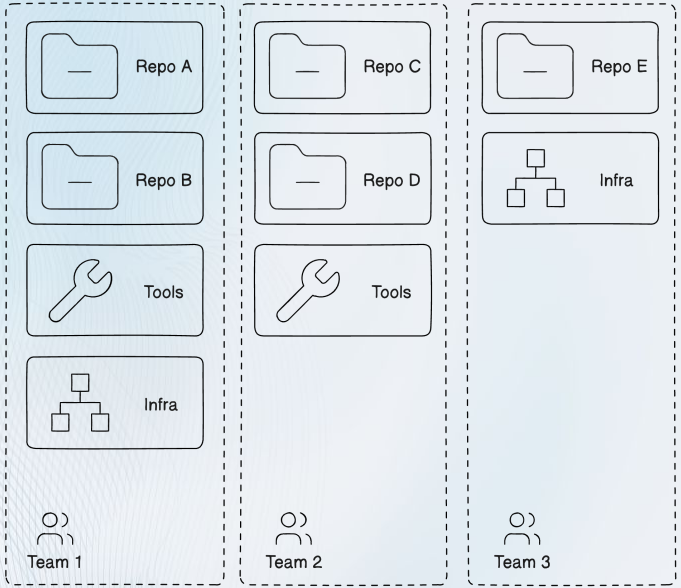


Django Monolith + Asynchronous ML Jobs

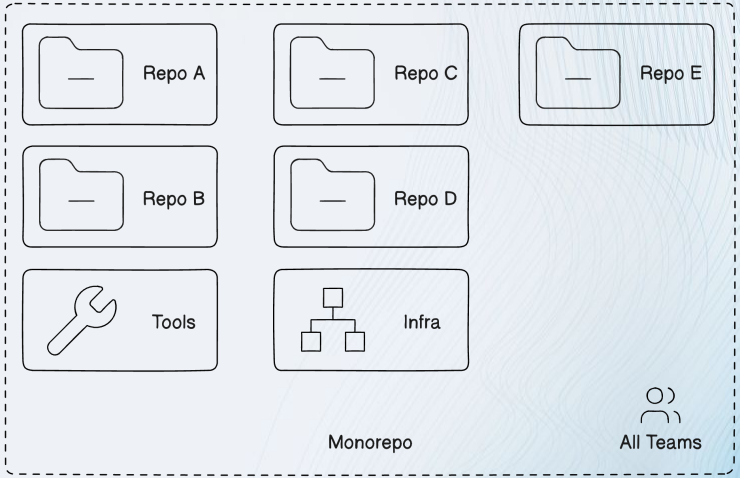


b. The Monorepo

b. The Monorepo



vs.



Multiple repositories

One Monorepo

Multi-repo vs. Monorepo

Multi-repo

Monorepo

Multi-repo vs. Monorepo

Multi-repo

Pros

- Clear ownership boundaries (in theory)
- Independent versioning
- Simple access control

Monorepo

Multi-repo vs. Monorepo

Multi-repo

Pros

- Clear ownership boundaries (in theory)
- Independent versioning
- Simple access control

Cons

- Painful cross-repo changes
- Dependency drift
- Code duplication

Monorepo

Multi-repo vs. Monorepo

Multi-repo

Pros

- Clear ownership boundaries (in theory)
- Independent versioning
- Simple access control

Cons

- Painful cross-repo changes
- Dependency drift
- Code duplication

Monorepo

Pros

- Cross-project commits
- Shared tooling and CI
- Easy code sharing

Multi-repo vs. Monorepo

Multi-repo

Pros

- Clear ownership boundaries (in theory)
- Independent versioning
- Simple access control

Cons

- Painful cross-repo changes
- Dependency drift
- Code duplication

Monorepo

Pros

- Cross-project commits
- Shared tooling and CI
- Easy code sharing

Cons

- Tooling complexity at scale
- Slower repo operations
- Risk of tight coupling

Multi-repo vs. Monorepo

Multi-repo

Pros

- Clear ownership boundaries (in theory)
- Independent versioning
- Simple access control

Cons

- Painful cross-repo changes
- Dependency drift
- Code duplication

Monorepo

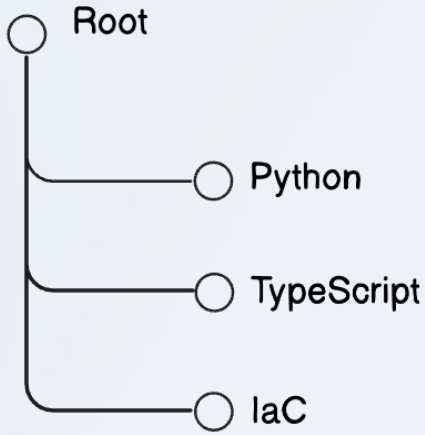
Pros

- Cross-project commits
- Shared tooling and CI
- Easy code sharing

Cons

- Tooling complexity at scale
- Slower repo operations
- Risk of tight coupling

Monorepo Implementation

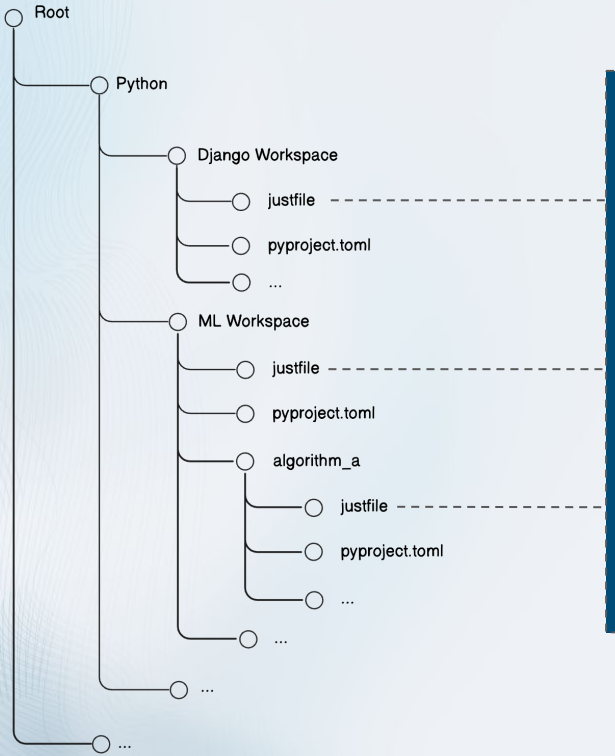


One folder per programming language

Monorepo Implementation

One uv workspace per deployable unit

Monorepo Implementation




```
# justfile

lint:
    just .. lint [django|ml|...]


type:
    just .. type [django|ml|...]


test *pytest_flags:
    just .. test [django|ml|...] {{pytest_flags}}

...
```

Monorepo Impact on ML Team

Monorepo Impact on ML Team

Faster adoption of best practices

Monorepo Impact on ML Team

 Faster adoption of best practices

 Shared tooling and housekeeping

Monorepo Impact on ML Team

 Faster adoption of best practices

 Shared tooling and housekeeping

 Product understanding

Monorepo Impact on ML Team

 Faster adoption of best practices

 Shared tooling and housekeeping

 Product understanding

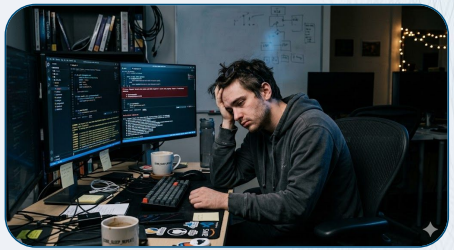
 Cross-team learning

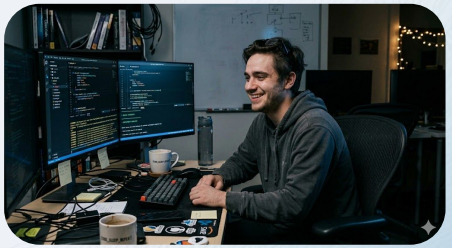
Monorepo Impact on ML Team

- Faster adoption of best practices
- Shared tooling and housekeeping
- Product understanding
- Cross-team learning

c. The Local Environment

c. The Local Environment

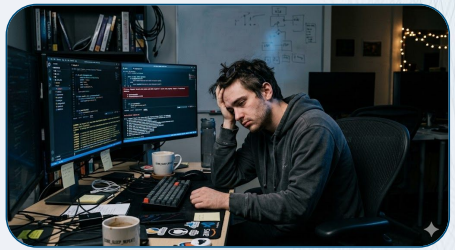


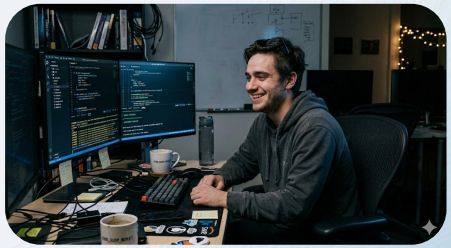


c. The Local Environment

Goals:

- Run everything locally
- Speed up feature development
- Facilitate bug fixing
- Increase feeling of product ownership

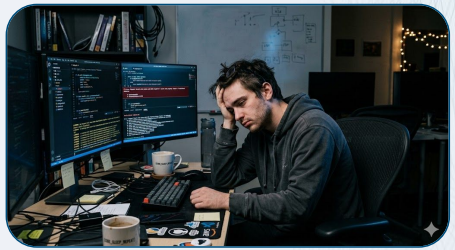


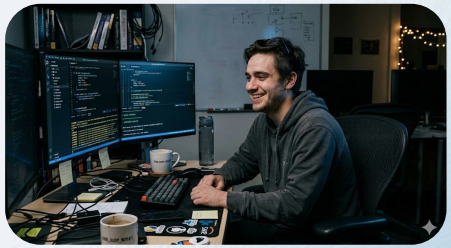


c. The Local Environment

Goals:

- Run everything locally
- Speed up feature development
- Facilitate bug fixing
- Increase feeling of product ownership





How?

- Mocking parts of the system
- Single start and stop commands
- But: keep writing tests!

3. Results & Takeaways

Results

✅ Simpler system design

✅ Clearer ownership

✅ Easier bug hunts

✅ Faster release process

Infrastructure Cost	- 44%
Feature PRs	+ 55 %
Bug fix PRs	- 31 %

Positive Surprises 🎁

Positive Surprises 🎁

⚡ Agentic coding setup

Positive Surprises 🎁

⚡ Agentic coding setup

🎓 Cross-team learning

Positive Surprises 🎁

⚡ Agentic coding setup

🎓 Cross-team learning

🧬 Staying up to date with product state

Positive Surprises 🎁

⚡ Agentic coding setup

🎓 Cross-team learning

🧬 Staying up to date with product state

🔬 Integration tests easier to add and maintain

Takeaways

Takeaways

1. There is no one-size fits all

Takeaways

1. There is no one-size fits all

2. Simplify if possible

Takeaways

1. There is no one-size fits all
2. Simplify if possible
3. Prioritize standard frameworks

Takeaways

1. There is no one-size fits all
2. Simplify if possible
3. Prioritize standard frameworks
4. Re-evaluate your situation from time to time

Thank you for listening!

Feel free to use our **monorepo template** for your own projects!

Thank you for listening!

Explore AI-driven discovery with Atinary SDLabs® for individual researchers!