

Platform Team Structures and Concerns



Adrian Cockcroft
Craft Conference
May 2025



View from the
Hilton Budapest



What This Talk is About



Real-world platform
design patterns that
scale



Lessons learned from
Netflix and Nubank's
platform journey



How to structure
platform teams for
success



What to avoid when
building internal
platforms

Topics



Platform Engineering



Evolving Your Stack



Making It Work

Why “Platform Engineering”



Kubernetes complexity demanded better tooling and orchestration



Vendors promoted 'platform engineering' to sell tools



Team Topologies book formalized platform team structure and roles

Principle 1: Platforms Are Plural

You don't build 'a platform' — you construct a *layered stack* of platforms

Each layer requires different knowledge, tooling, and ownership

Different teams handle different parts of the stack

Principle 2: Platform Layers Evolve

Platforms move 'up the stack'
over time

New functionality is added; older
capabilities are sedimented

Constant re-evaluation and
refactoring is essential

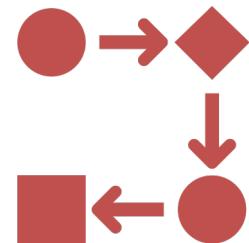
Principle 3: User-Driven Interfaces

Treat internal platforms
like products

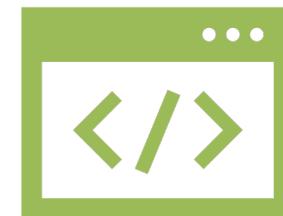
Designate a product
manager for each platform

Use roadmaps and request
prioritization systems

Principle 4: Treat Internal and External Platforms Differently



Internal - platforms SHOULD evolve quickly and will be messy



External - platforms must optimize for stability and backward compatibility

Platform Stack Layers



Deployment Platforms (Kubernetes, AWS Lambda)



Virtualization and Networking (VMware, AWS EC2)



Languages and Libraries (Java, Go, Python, Rust, etc.)



Operating Systems (Linux distros, Windows variants)



Hardware (CPU/GPU vendors, instance types)

Netflix Platform Strategy



Used AWS as a stable base; avoided intermediate vendor layers

Encouraged AWS to evolve to meet Netflix's needs

Built a *thin*, fast-changing internal layer on top of AWS

Open sourced some platform components

Why Avoid Intermediate Vendors

They evolve slower
than your business
needs

Introduce extra
layers of failure and
complexity

Misaligned
incentives: their
goal is to lock in
value, not move fast

Prefer Open Source Over Vendors



Open development model enables in-house contributions



Transparency in roadmap, bug fixing, and support



Netflix contributed to many Apache projects from the cloud platform team (Cassandra, Zookeeper...)

Multiple Platform Teams at Netflix



Teams for online services, personalization, video players, data science, encoding, security, IT, etc.



Each team built a custom platform layer suited to their domain



Different tech stacks: Java, Python, C++, Swift, Javascript...

Internal Developer Platforms



Developer Experience Platform Team: CI/CD, library support, vendor integration



Container Platform Team: Kubernetes, orchestration, infrastructure



Distinct roles, skills, and tooling needs

Key Takeaways for Platform Teams



Treat platforms as a **layered, evolving stack**



Design with product thinking — serve your internal customers



Use vendors to support open-source tooling



Separate infra-oriented and dev-experience teams



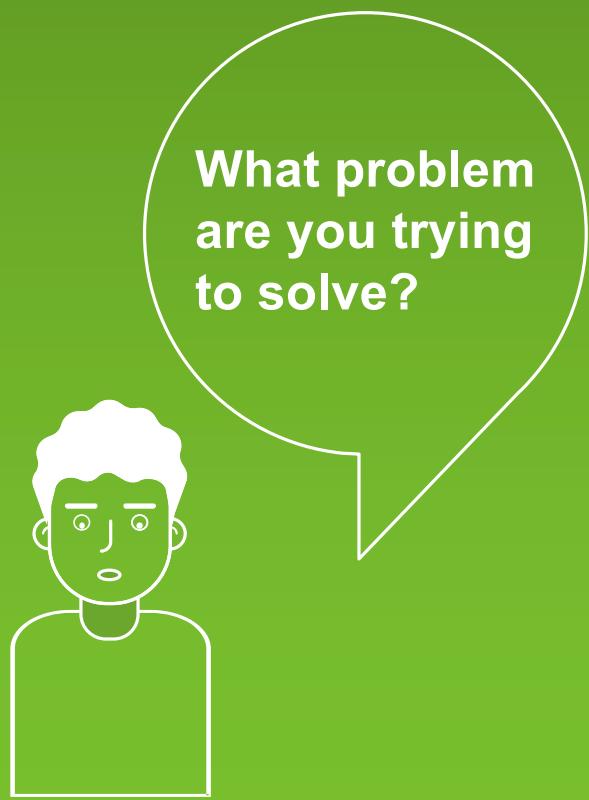
Optimize for change where you can, for stability where you must

Wardley Mapping: Platform Evolution



Technology evolves, and our platform technology stack choices need to be discussed, justified and decided.

Start with a few helpful questions...



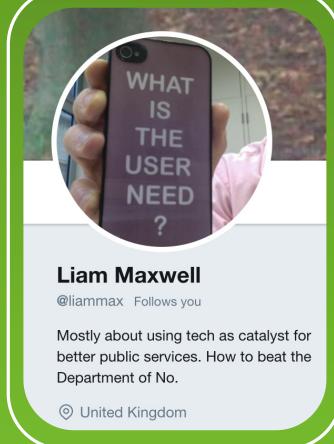
HINT—Building or installing a platform is the wrong answer...

Something involving new **user** or **business needs** is a good answer:

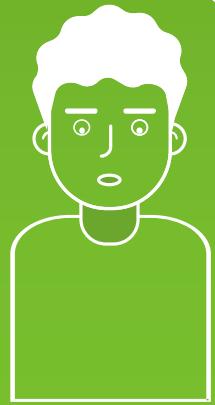
- Faster feature delivery?
- International launch?
- Lower cost?
- Improved Quality of Service?



What is the user need?



User need



**What does the
value chain
look like?**

Have you heard of
Simon Wardley?

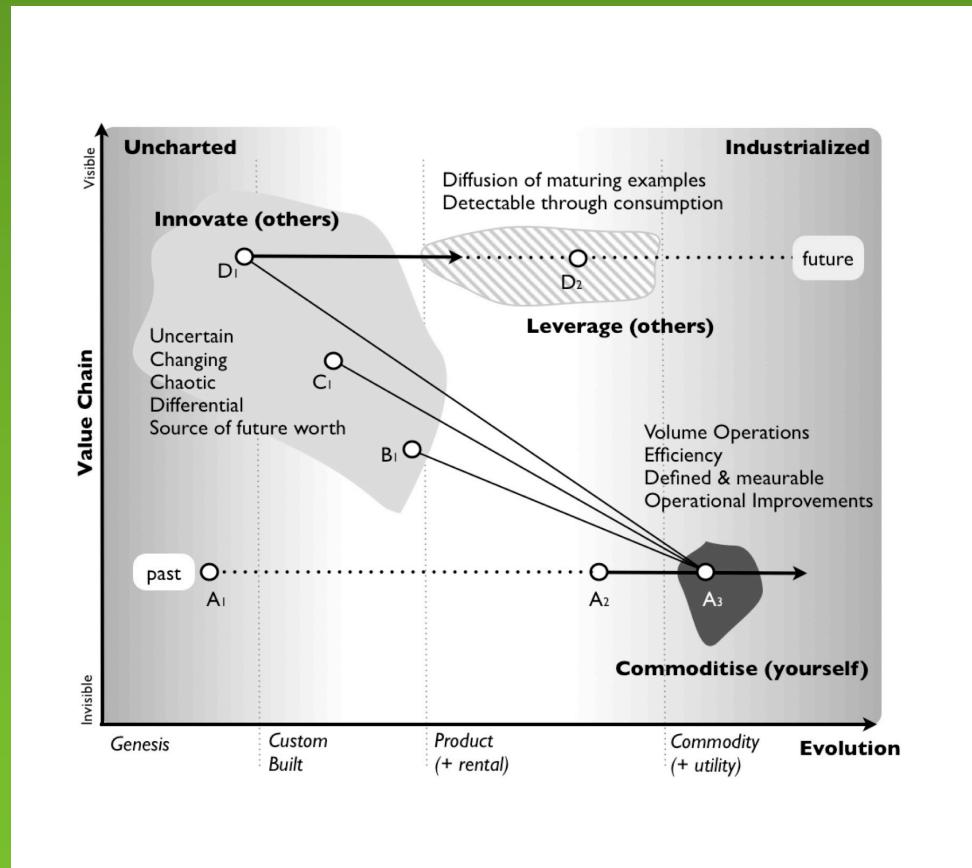
What's your
time-to-value?
How is the value
chain evolving?



A screenshot of a Medium profile for Simon Wardley. It shows a photo of him, his name "swardley", his bio ("I like ducks, they're fowl but not through choice. Corporate cartographer, chaotic evil, destroyer of undeserved value. RT ≠ +1"), his location ("Burmarsh, Kent, UK"), and a link to his profile ("medium.com/wardleymaps/on...").

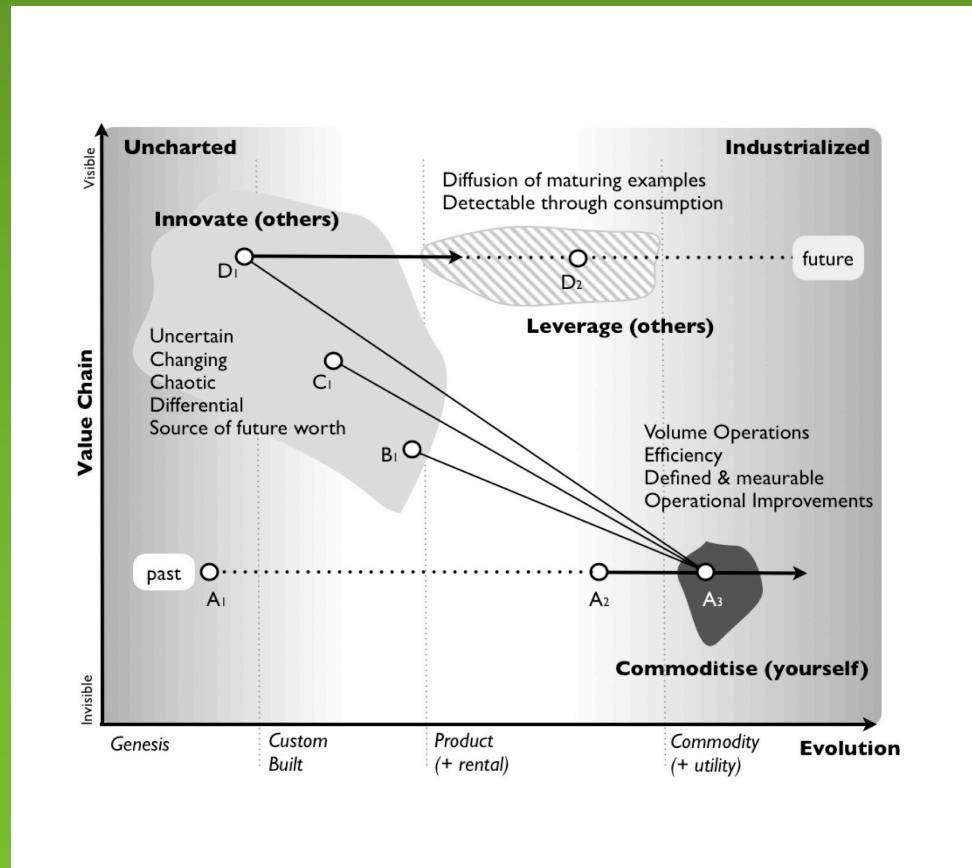
Leverage Wardley Maps

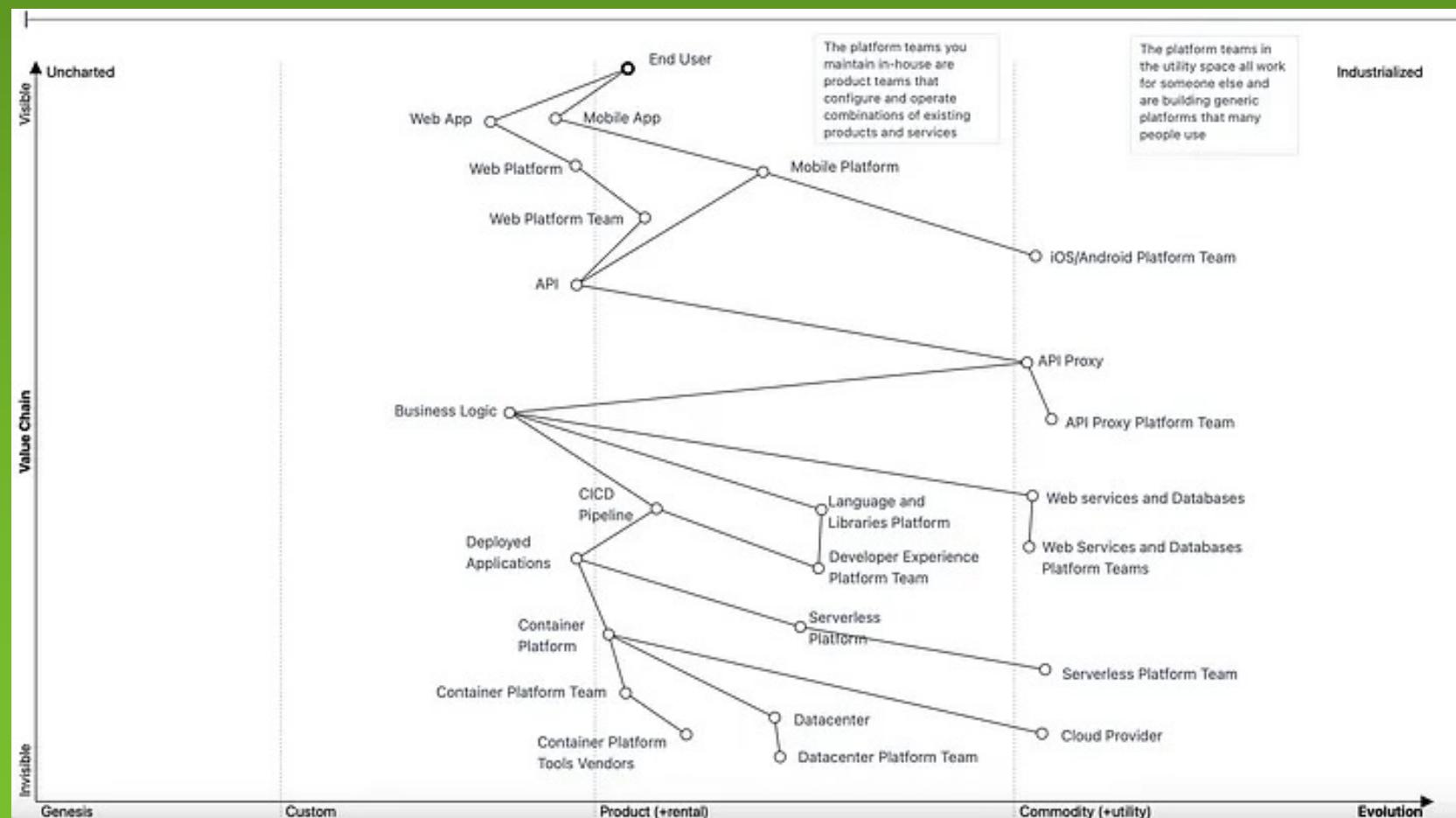
Visualize evolution
and innovate further
up the value chain



Let's look at your Wardley Map

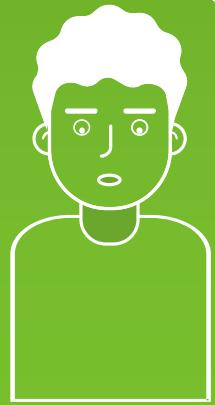
What's your position
and movement on
the map?





A Specific Example:

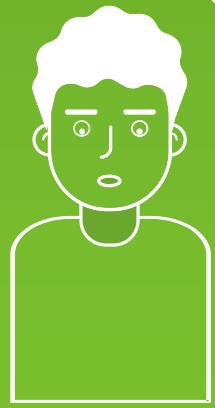
Travel Search



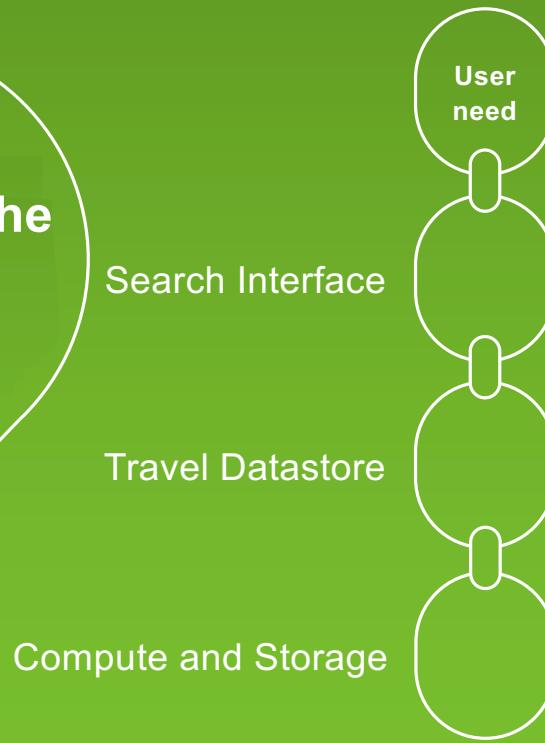
**What problem
are you trying
to solve?**

Users want:
Faster personalized search

Business wants:
Faster feature releases
Lower costs

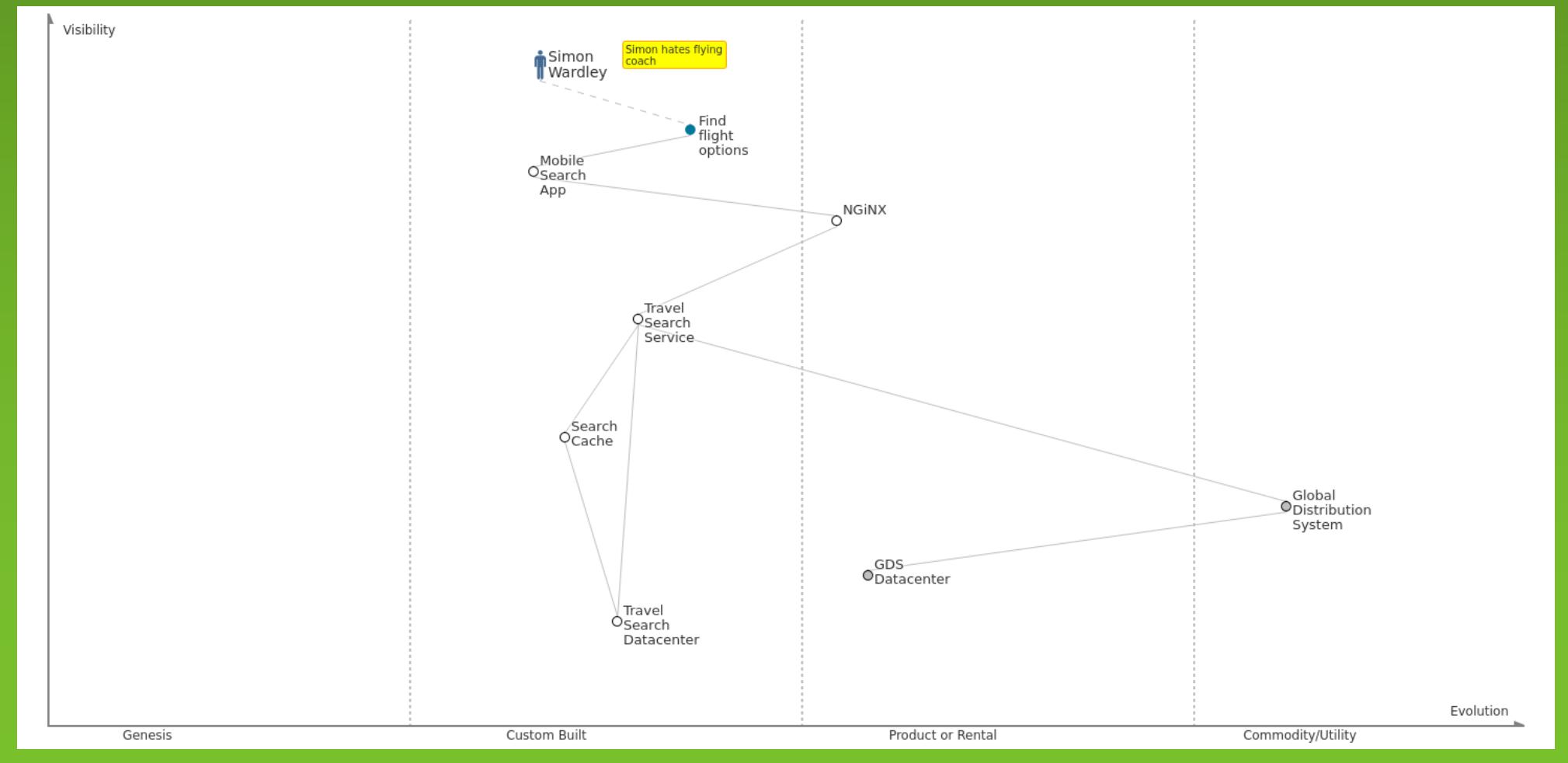


What does the
value chain
look like?



**Example created using
mapkeep.com**

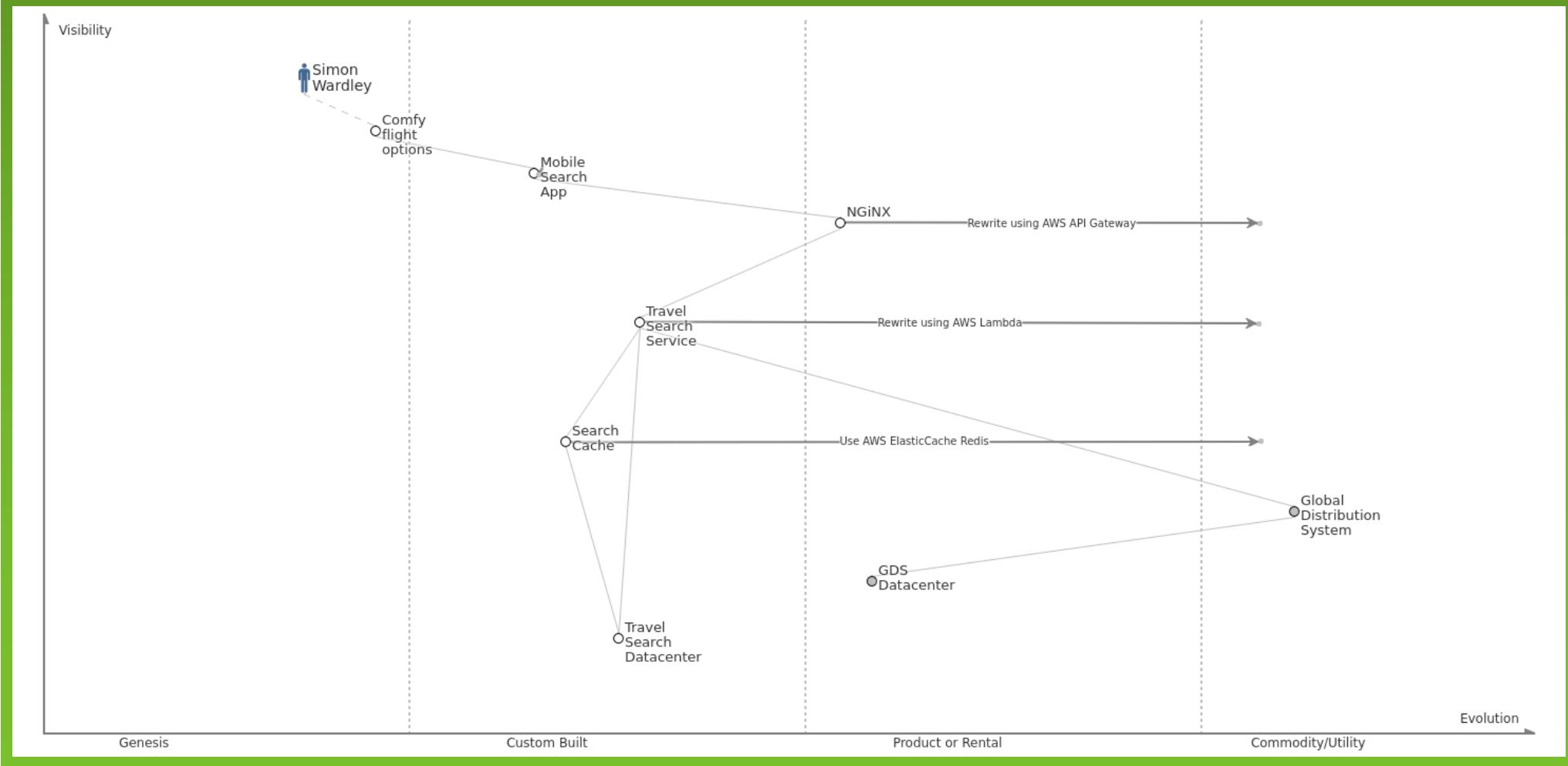
Travel search – starting point



Travel search – comfy seat search – what kind of plane is it?



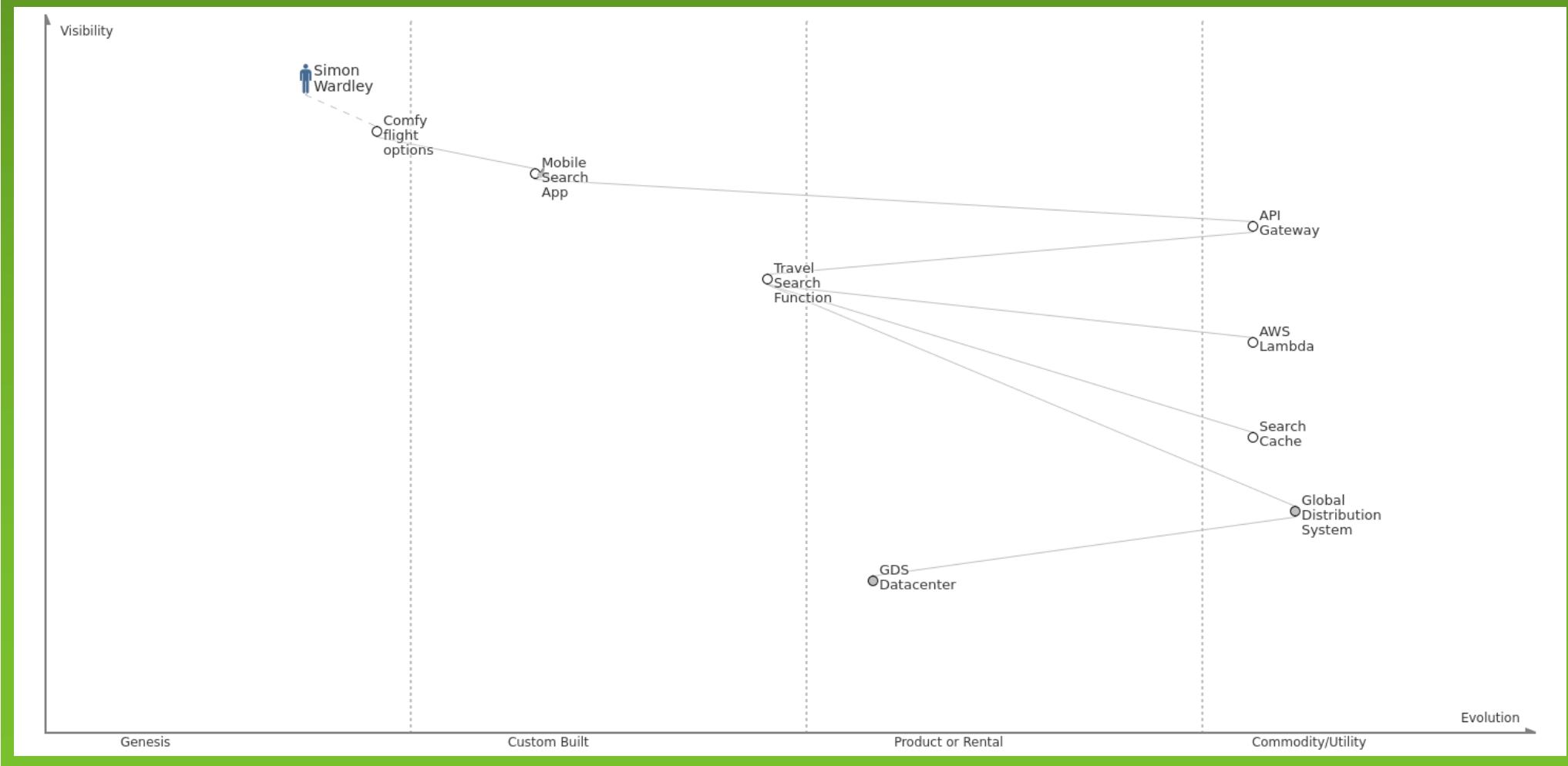
Travel search – reduce search cost



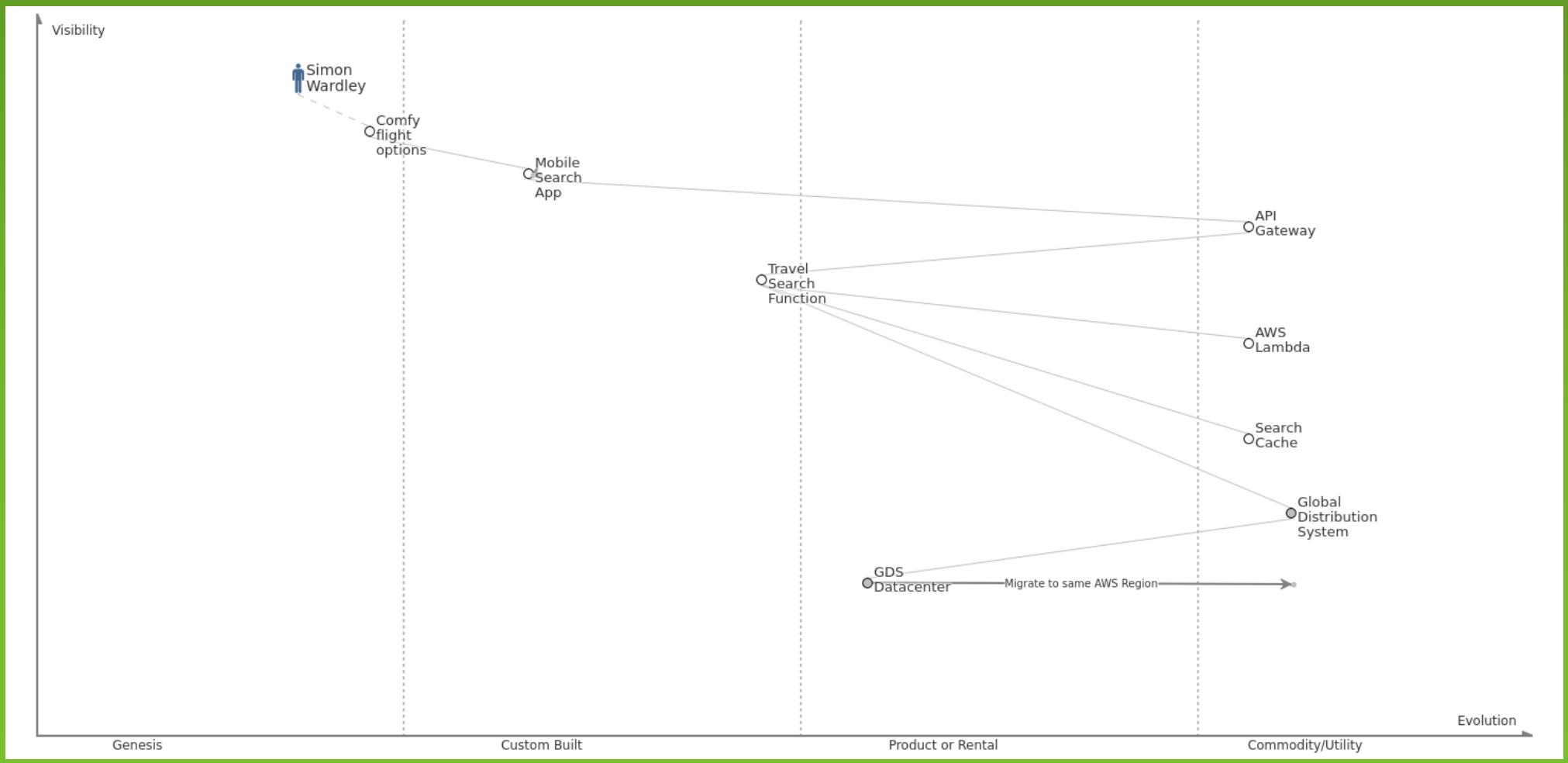
Travel search – exit datacenter



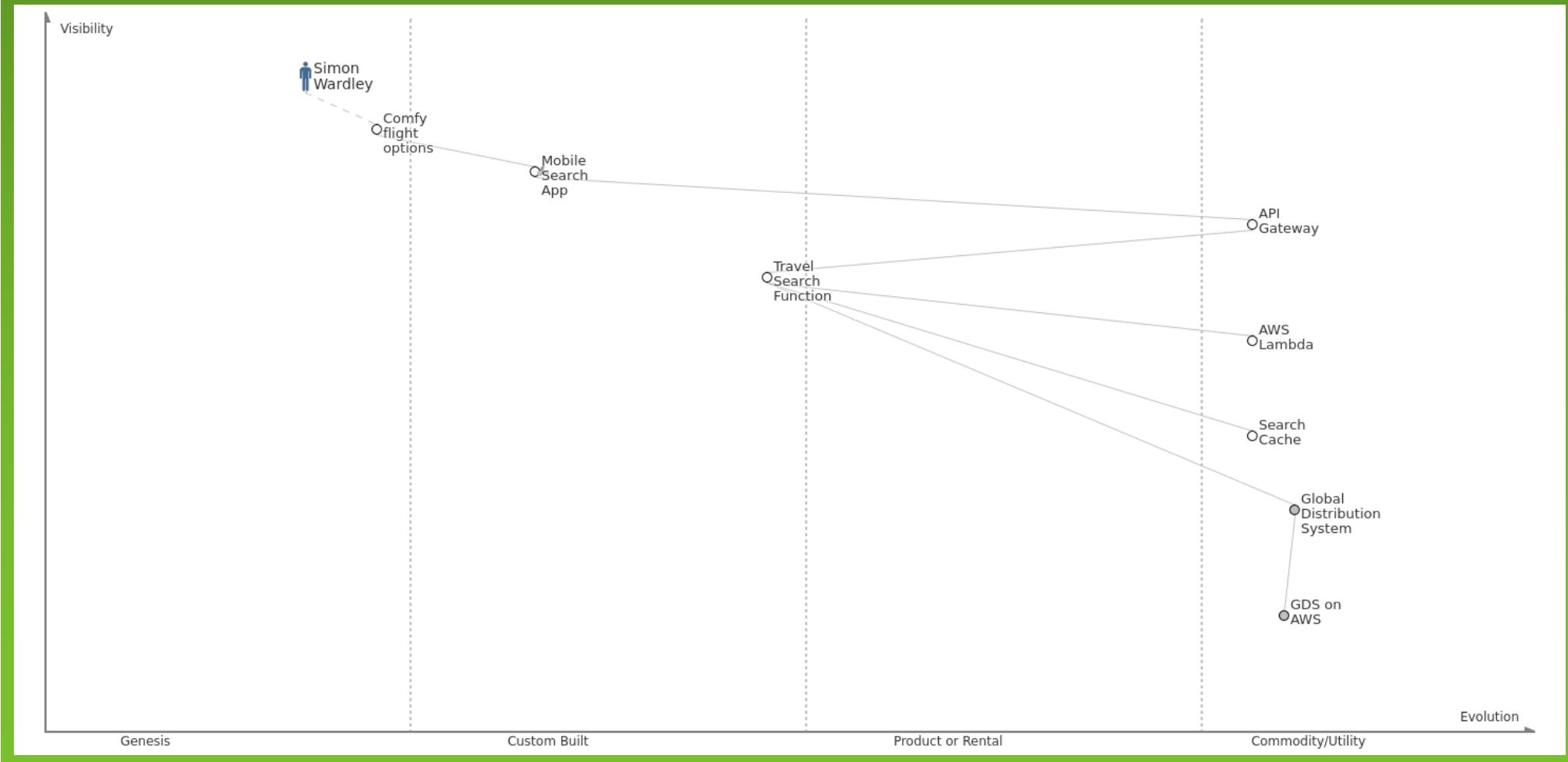
Travel search – serverless



Travel search – GDS optimized network



Travel search – happy Simon, lower cost to serve



Making it Work at
Nubank



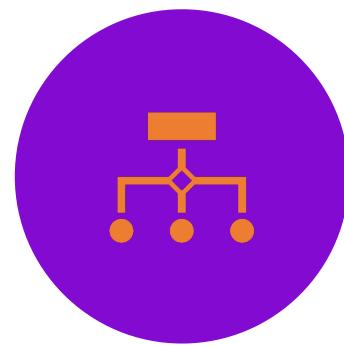
Questions



WHO ARE NUBANK?



WHY ARE THEY
INTERESTING?



WHAT ARE YOU DOING
WITH THEM?

Who are Nubank?

Nubank is one of the largest digital financial services platforms in the world, currently serving over 118 million customers across Brazil, Mexico and Columbia, with plans to expand internationally.

Founded in 2013 using a cloud-based technology platform, Nubank operates at massive scale on a modern software infrastructure platform

Nubank Platforms Stack

-  Mobile App (Banking, Insurance, Brokerage, Marketplace, Travel, Betting, Crypto...)
-  Clojure business logic (thousands of microservices, 24,504 deploys in April '25)
-  Datomic immutable storage abstractions backed by AWS DynamoDB
-  Python for data science and AI/ML
-  Istio Service Mesh, Kafka messaging
-  Kubernetes using AWS EKS (many large clusters) and Karpenter
-  EC2 Intel and Graviton (ARM) on demand and spot market

Nubank Technology Leadership

Vitor Olivier CTO, Edward Wible DE, founding technology team

Nubank acquired Cognitect and the Clojure/Datomic ecosystem in 2020, along with Rich Hickey DE (retired), Stu Halloway DE, Justin Gehtland VP Technology

Michael Nygard Chief Architect, well known as author of *Release IT*

Cat Swetel is General Manager of the Foundation Platform Business (200+ engineers)

Adrian Cockcroft was AWS exec sponsor for Nubank, part time advisor since 2022



Cat Swetel's Doomsday Clocks



📈 Managing a rapidly growing collection of platforms

🏠 Technical debt builds up, old versions, cruft

✓ Scalability limits constrain growth

👥 How can the team prioritize work?

⌚ Every component has a doomsday clock, when it will fail

⌚ Estimate time needed to reset the clock

⚡ Sort by most pressing work, to avert the next failure

What am I doing with Nubank?



Consulting with Observability Platform Team (In-house logging platform, Victoria Metrics, Grafana etc)



Consulting with Controllability Platform Team (Reliability, chaos testing with Litmus and AWS FIT, game days, incident management)



Building new tools for workload analysis and increasing utilization with System Performance team. Using Cursor to vibe-code Python...



Asking awkward questions, reviewing docs, making suggestions

Plus ça change, plus c'est la même chose

The more it changes, the more it stays the same...



Platforms need to be stable and reliable



Platforms need to add new features and fixes quickly

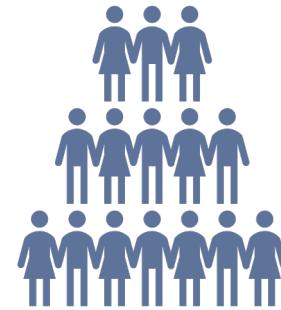


How did we solve for these conflicting goals at Netflix?

Team Appetite for Change



Some teams are innovating rapidly and need new platform features; they take the pain and debug for everyone else



Other teams are stability and maintenance focused; they want a platform that just works

*“There is no perfect way to handle versioning,
just some less bad options...”*

From 2016 Microservices Workshop

<https://github.com/adrianco/slides/blob/master/Microservices%20Workshop%202016.pdf>

Non-Destructive Production Updates



- “*Immutable Code*” Service Pattern
 - *Existing services are unchanged, old code remains in service*
 - *New code deploys as a new service group*
 - *No impact to production until traffic routing changes*
- *A|B Tests, Feature Flags and Version Routing control traffic*
 - *First users in the test cell are the developer and test engineers*
 - *A cohort of users is added looking for measurable improvement*



From 2016 Microservices Workshop

<https://github.com/adrianco/slides/blob/master/Microservices%20Workshop%202016.pdf>



Benefits of version aware routing



Immediately and safely introduce a new version

Canary test in production

Use DIY feature flags,  LaunchDarkly, A|B tests with Wasabi

Route clients to a version so they can't get disrupted

Change client or dependencies but not both at once

Eventually remove old versions

Incremental or infrequent “break the build” garbage collection

From 2016 Microservices Workshop

<https://github.com/adrianco/slides/blob/master/Microservices%20Workshop%202016.pdf>



Versioning, Routing



Version numbering: Interface.Feature.Bugfix

V1.2.3 to V1.2.4 - Canary test then remove old version

*V1.2.x to V1.3.x - Canary test then remove or keep both
Route V1.3.x clients to new version to get new feature
Remove V1.2.x only after V1.3.x is found to work for V1.2.x clients*

*V1.x.x to V2.x.x - Route clients to specific versions
Remove old server version when all old clients are gone*

From 2016 Microservices Workshop

<https://github.com/adrianco/slides/blob/master/Microservices%20Workshop%202016.pdf>



Interfaces



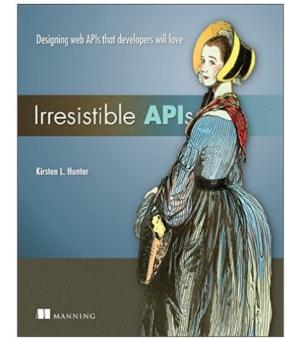
*When you build a service, build a “driver” client for it
Reference implementation error handling and serialization*

Release automation stress test using client

Validate that service interface is usable!

Minimize additional dependencies

*Swagger - OpenAPI Specification
Datawire Quark adds behaviors to API spec*



From 2016 Microservices Workshop

<https://github.com/adrianco/slides/blob/master/Microservices%20Workshop%202016.pdf>



Interface Version Pinning



*Change one thing at a time!
Pin the version of everything else*

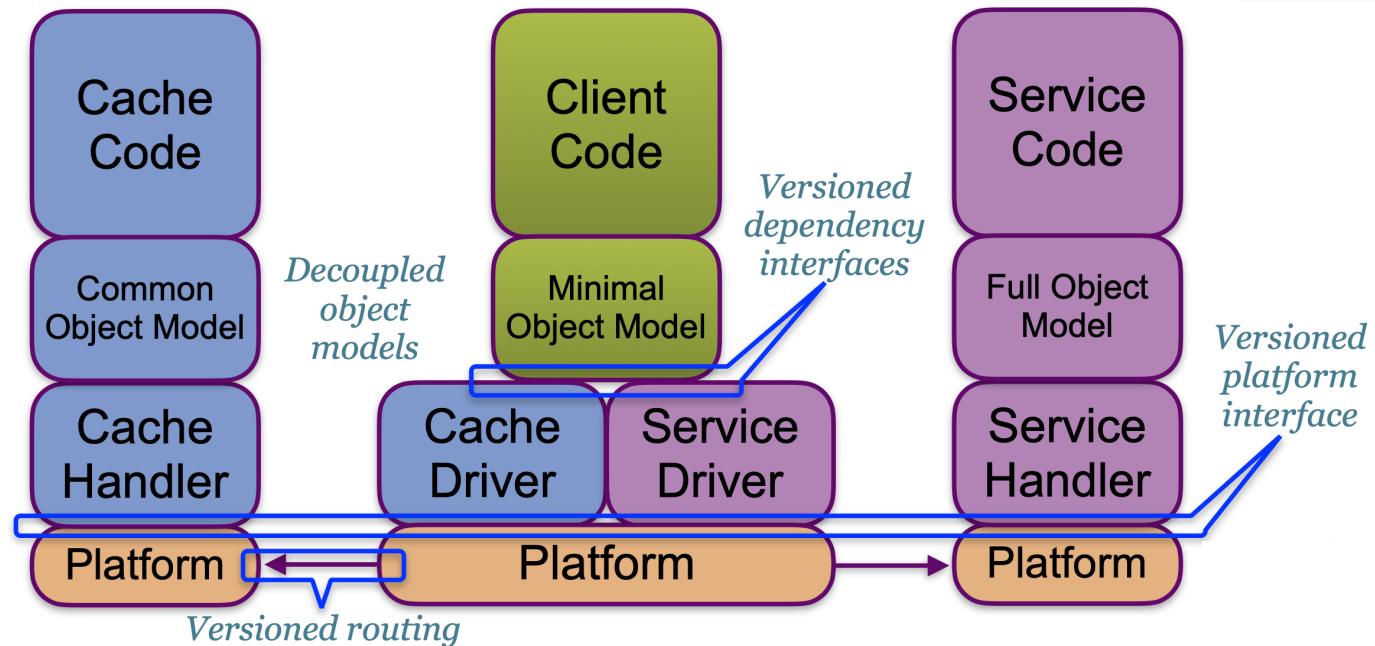
Incremental build/test/deploy pipeline

*Deploy existing app code with new platform
Deploy existing app code with new dependencies
Deploy new app code with pinned platform/dependencies*

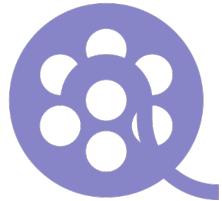
From 2016 Microservices Workshop

<https://github.com/adrianco/slides/blob/master/Microservices%20Workshop%202016.pdf>

Interfaces between teams



Lots of old advice



Answer every question
with "we did that at
Netflix 15 years ago"



Lots of published content,
too much to remember,
so...



I've developed an AI
Persona that can answer
detailed questions

Try it out! <https://app.soopra.ai/Cockcroft/chat>



Discover

Experts

Courses

Search

Account

More

Create AI Persona

Sign In



Adrian Cockcroft @Cockcroft

Adrian Cockcroft is a technologist and strategist with broad experience from the bits to the boardroom, in both enterprise and consumer-oriented businesses, from startups to some of the largest companies in the world, equally at home with hardware and software, development and operations. He's best known as the cloud architect for Netflix during their trailblazing migration to AWS and was a very early practitioner and advocate of DevOps, microservices, and chaos engineering, helping bring these concepts to the wider audience they have today.

I'm Adrian Cockcroft, a seasoned engineer and leader in the tech industry. My career has spanned various roles at Sun Microsystems, eBay, Netflix, venture capital and Amazon. You can ask me about my experiences in the tech industry, my insights on micro services architecture, chaos engineering, DevOps, resilience, capacity planning, performance engineering, or my thoughts on the future of AI.

20 May 2025

What's the best way to introduce chaos engineering into an organization?

Introducing chaos engineering into an organization is a strategic process that requires careful planning and communication. Given the organizational structure you've described, with managers overseeing developers who are responsible for multiple microservices, here are some steps to consider:

- Start with Education and Awareness:** Begin by educating the team about the principles and benefits of chaos engineering. This involves understanding that chaos engineering is about proactively identifying weaknesses in a system by intentionally injecting failures in a controlled manner.
- Gain Management Buy-In:** Since you have a clear managerial structure, it's crucial to get buy-in from the managers first. They can help champion the initiative and allocate resources and time for chaos experiments.
- Pilot Program:** Select a small, non-critical microservice to start with. This allows the team to experiment without risking major disruptions. The goal is to learn and iterate on the process.
- Build a Cross-Functional Team:** Form a small team that includes developers from different areas. This team will be responsible for designing and running chaos experiments. Their diverse perspectives will be valuable in identifying potential failure points.
- Use Tools and Frameworks:** Leverage existing chaos engineering tools like Chaos Monkey or Gremlin to conduct experiments. These tools can help automate the process and provide insights into system behavior under stress.
- Iterate and Scale:** After initial experiments, analyze the results, learn from any failures, and gradually scale the chaos engineering practices to other parts of the organization.
- Foster a Culture of Resilience:** Encourage a mindset where failures are seen as learning opportunities. This cultural shift is essential for chaos engineering to be effective.



Say something



Extensible data preparation for any author at github.com/adrianco/megpt

The screenshot shows a GitHub repository page for [adrianco/meGPT](https://github.com/adrianco/meGPT). The repository is public and has 25 forks and 252 stars. The main navigation bar includes links for Code, Issues (14), Pull requests, Discussions, Actions, Projects, Security, and Insights.

The left sidebar displays the project's directory structure:

- main (selected)
- authors
- code
- mcp_resources
- processors
 - .gitignore
 - LICENSE
- README.md (selected)
- build.py
- create_mcp.py
- process.py
- requirements.txt

The right pane shows the content of the README.md file:

meGPT - upload an author's content into an LLM

I have 20 years of public content I've produced and presented over my career, and I'd like to have an LLM that is trained to answer questions and generate summaries of my opinions, in my "voice". At this point, I've found a few companies that are building persona's and tried out soopra.ai. To encourage development and competition in this space I have organized my public content and references to sources in this repo.

My own content is stored or linked to in authors/virtual_adrianco and consists of:

- 4 published books (pdf of two provided), ~10 forewords to books, ~100 blog posts (text)
- Twitter archive 2008-2022 (conversation text)
- Mastodon.social - 2021-now <https://mastodon.social/@adrianco> (RSS at <https://mastodon.social/@adrianco.rss>)
- Github projects (code)
- Blog posts from <https://adrianco.medium.com> extracted as text to authors/virtual_adrianco/medium_posts (with extraction script)
- Blog posts from <https://perfcap.blogspot.com> extracted as text to authors/virtual_adrianco/blogger_perfcap_posts (with extraction script)
- ~100 presentation decks (images) greatest hits: <https://github.com/adrianco/slides/tree/master/Greatest%20Hits>
- ~20 podcasts (audio conversations, should be good Q&A training material)
- ~50 videos of talks and interviews (audio/video/YouTube playlists)

If another author wants to use this repo as a starting point, clone it and add your own directory of content under authors. If you want to

AI, Build Thyself



"Vibe coded" in Python using ChatGPT and Cursor/Claude



Create <author>/published_content.csv

Your blog posts on Medium and Blogger
Youtube videos and playlists
Podcast appearances
Published stories elsewhere
Books, papers and presentations



Download and process content: \$ python build.py <author>



Claude said this should be formatted as MCP resources, so I'm doing what it said...

Clone the repo, edit the sample author to be you (send me a pull request!)

Soopra.ai Setup

Copy all the files into a shared google drive and link it

The screenshot shows the Soopra.ai setup interface. On the left, there's a sidebar with various options: 'Publish my AI Persona' (with a toggle switch), 'Account' (Profile, Purchases, My Courses), 'AI Persona' (Price Card, Dashboard), and 'Training Material' (Documents, Journal, Weblinks, Social Media, Voice Sample, Introduction). The 'Documents' option is highlighted with a grey background. The main content area has a heading 'Upload a Document' with a sub-instruction 'Upload articles, blogs, scripts, or any other content you've created to give a wider lens into your creativity.' Below it is a large 'Upload Document' button. Further down is a section titled 'Google Drive Links' with the instruction 'Add a link to a Google Drive folder. Google Drive Folders must be public to be synced and used for training.' It includes a note: 'NOTE: After saving, you must reload this page to see the updated sync progress of your Google Drive Links.' There are two 'Add Google Drive Link' buttons, one in a white box and one in a grey box.

OrionX^{net}

Consulting projects: orionx.net

Persona: app.soopra.ai/Cockcroft/chat

Social: @adrianco@mastodon.social

Blog: adrianco.medium.com

Monthly Analyst Podcast:
orionx.net/tag/analyst-roundtable/

