Beyond CI/CD: Simplifying the Developer Journey

How platform abstractions unlock developer productivity at scale

Gaurav Nanda
Senior Engineering Manager, Databricks

Let me tell you a story...

...about a developer trying to connect two services at Databricks.

Building a service was easy.

Building a service was easy.

Connecting it wasn’t.

Building a service was easy.

Connecting it wasn’t.

4 WEEKS

in some cases, just to connect two services

HOW THE STORY UNFOLDS

01

THE PAIN

Why did connecting two services take 4 weeks?

02

THE PLATFORM RESPONSE

How do we simplify the developer journey?

03

THE BROADER LESSON

What are the broader lessons?

DATABRICKS AT SCALE

3

CLOUDS

70+

REGIONS

1,500+

CLUSTERS

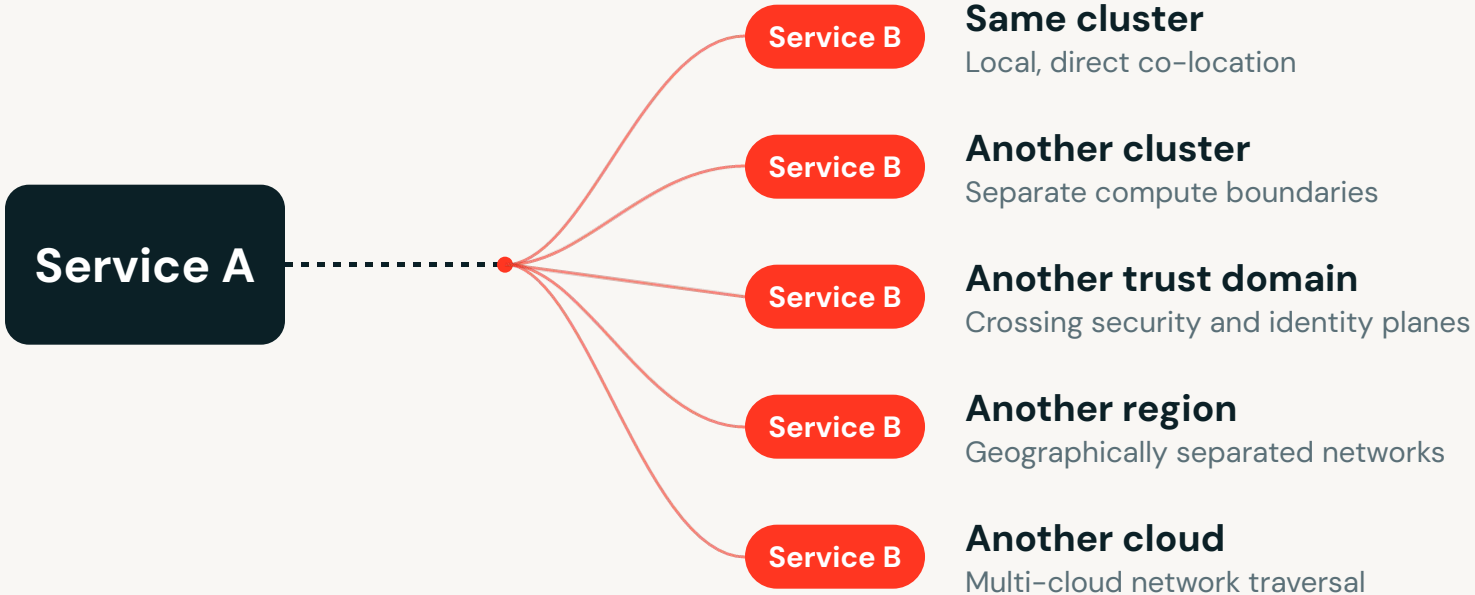
MILLIONS

OF VMs / DAY

Services operate across clusters, regions, planes, and clouds.

BACK TO OUR DEVELOPER

But where is Service B?



DATABRICKS NETWORKING CAPABILITIES

Strong building blocks.

Naming & Resolution

DNS · service names

Service Discovery

Discover endpoints

Network Connectivity

Cross-cluster · cross-region

Routing & Proxies

Load balancing · Envoy

Security & Policy

mTLS · AuthZ

DATABRICKS NETWORKING CAPABILITIES

Strong building blocks. Fragmented experience.

Naming & Resolution

DNS · service names

Service Discovery

Discover endpoints

Network Connectivity

Cross-cluster · cross-region

Routing & Proxies

Load balancing · Envoy

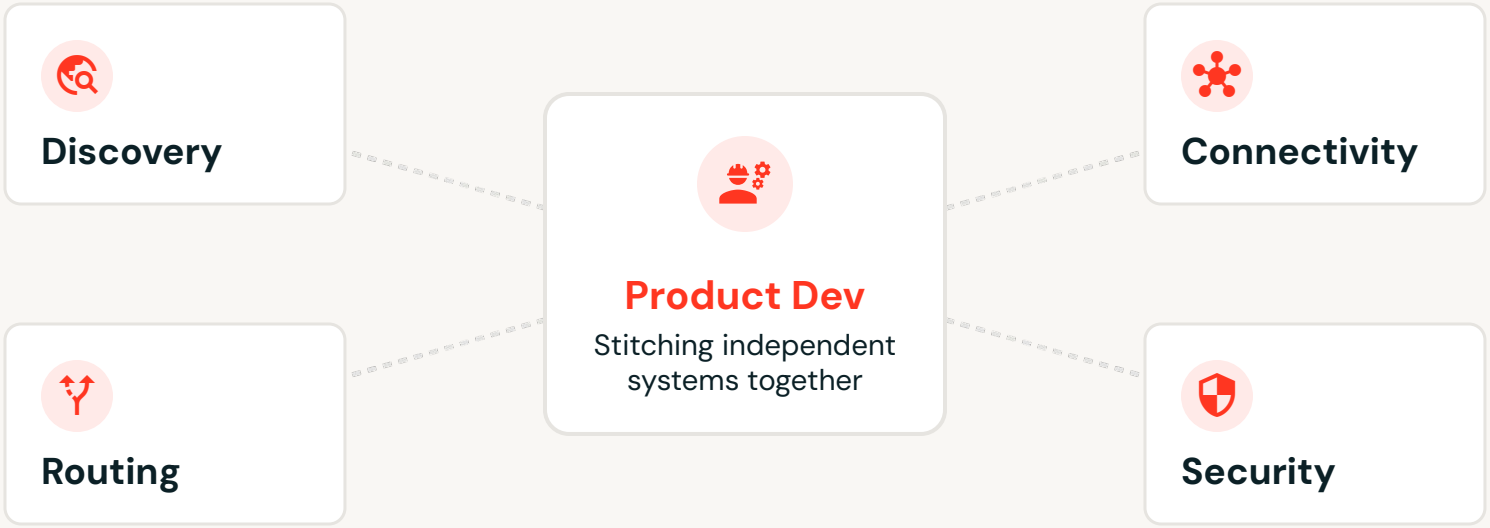
Security & Policy

mTLS · AuthZ

Built and owned by different infrastructure teams.

DEVELOPER JOURNEY

The developer became the integrator.



Find the right teams. Agree on a connectivity design.

DEVELOPER JOURNEY

The design was only the beginning.

Repeated across multiple infrastructure systems

STEP 01

Follow onboarding SOP

STEP 02

PR + Review

STEP 03

Release / Rollout

Multiple teams. Different docs, processes, reviewers, and approvals.

DEVELOPER JOURNEY

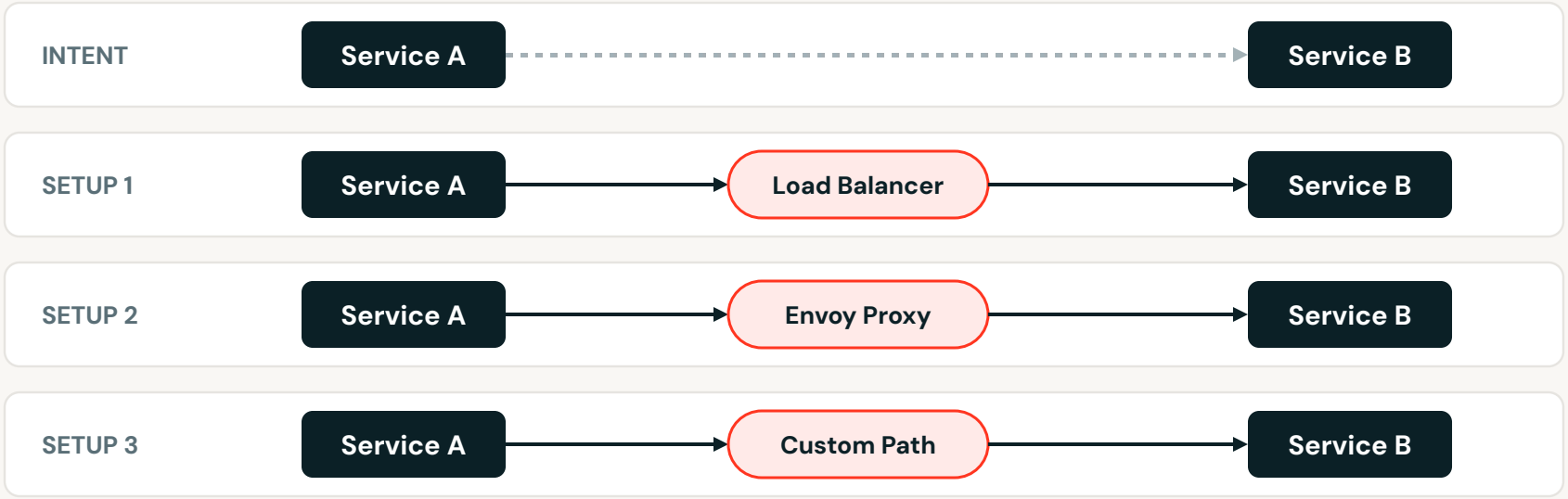
4 WEEKS

Just to connect two services.

Repeated again and again across product teams.

THE RESULT OF INDEPENDENT DECISIONS

Same intent. Different setups.



No single standard connectivity pattern.

Variation had a cost.

DEVELOPERS

- Longer setup
- More infra knowledge
- More handoffs

INFRASTRUCTURE TEAMS

- Harder to support
- Harder to standardize
- Harder to evolve

We kept paying for the complexity.

HOW THE STORY UNFOLDS

01

THE PAIN

Why did connecting two services take 4 weeks?

02

THE PLATFORM RESPONSE

How do we simplify the developer journey?

03

THE BROADER LESSON

What are the broader lessons?

Where platform teams often focus

BUILD → TEST → DEPLOY

And rightfully so. CI/CD is foundational.

Zooming out on developer productivity

CREATE → **CONNECT** → **OPERATE**

Build · Test · Deploy

Discover · Access · Communicate

Observe · Debug · Optimize

Zooming out on developer productivity

CREATE → **CONNECT** → **OPERATE**

Build · Test · Deploy

Discover · Access · Communicate

Observe · Debug · Optimize

What should connect journey look like?

DESIGN PRINCIPLE

Developers should not make infrastructure decisions. They should express intent.

DESIGN PRINCIPLE

Developers should not make infrastructure decisions. They should express intent.

Who should make the infrastructure decisions?

PLATFORM ABSTRACTION

One service name. One connectivity model.

Databricks Naming Service (DBNS)

BEHIND DBNS

DBNS handles the connectivity setup.

"Service B" DBNS configuration

Identifier

service-b

Target port

8080

Reachability

CLUSTER / REGION / ...

DBNS CONTROL PLANE

Configures the underlying connectivity.

THE IMPACT

Same intent. Far less developer work.

BEFORE

4 WEEKS

Connectivity setup

→

WITH DBNS

5 MINUTES

Connectivity setup

THE PAVED PATH

One supported, consistent way to connect services.

The platform handles the complexity underneath.

THE CHALLENGE

Building the paved path wasn’t enough.

NEW SERVICES

DBNS by default

EXISTING SERVICES

Still on older patterns

THE GOAL

We needed one consistent, supportable setup.

Reducing infrastructure complexity requires migrating existing services too.

MIGRATION AT SCALE

AI made it cheaper to move services onto the paved path.

BEFORE

Each service team drives its own migration

WITH AI

Platform team drives migrations centrally

AI-generated changes · Team approval · Monitored rollout

HOW THE STORY UNFOLDS

01

THE PAIN

Why did connecting two services take 4 weeks?

02

THE PLATFORM RESPONSE

How do we simplify the developer journey?

03

THE BROADER LESSON

What are the broader lessons?

Does everyone need DBNS?

Does everyone need DBNS?

It depends.

Intent-based service connectivity
can be valuable as you grow

THE BROADER LESSON

Your highest-leverage area may be different.

CREATE

Build · Test · Deploy

→

CONNECT

Discover · Access · Communicate

→

OPERATE

Observe · Debug · Optimize

PLATFORM AS A PRODUCT

Follow the developer journey.

Your developers are your customers.

PLATFORM AS A PRODUCT

Follow the developer journey.

Your developers are your customers.

QUESTION 01

What are developers trying to accomplish?

QUESTION 02

Where are they spending repeated effort?

QUESTION 03

Where can the platform create leverage?

PLATFORM AS A PRODUCT

Follow the developer journey.

Your developers are your customers.

QUESTION 01

What are developers trying to accomplish?

QUESTION 02

Where are they spending repeated effort?

QUESTION 03

Where can the platform create leverage?

Invest where your developers need the **most leverage.**

KEY TAKEAWAYS

01

DEVELOPER PRODUCTIVITY IS END-TO-END

Create · Connect · Operate

02

ABSORB COMPLEXITY BEHIND STABLE ABSTRACTIONS

Keep the developer model simple as infrastructure evolves.

03

REDUCE VARIATION. CONVERGE ON THE PAVED PATH.

Uniformity makes the platform easier to support and evolve

Use AI to accelerate migration.

Platform teams exist to remove
repeated complexity and toil
from product development

Connect with me on LinkedIn