ProjectX

MANAGED DATABASES · DATASHEET

One resource for every engine you run.

PostgreSQL, MySQL, MariaDB, Redis, Valkey, MongoDB, ClickHouse and Kafka, each a value of one managed database resource — so backup, upgrade and failover reach every engine without a per-engine surface. Point an application at an endpoint that names no upstream.

Why teams choose it

01
Every engine, one mental model
Eight engines behind one resource model — learn backup, upgrade and failover once and it applies to Postgres, Redis, ClickHouse and the rest alike.

02
Failover that does not break the app
A failover leaves the connection string unchanged, so a primary going away is an event, not an incident with a deploy attached.

03
A wrong timestamp costs a throwaway
Point-in-time recovery restores into a new database, so a mistaken restore never destroys the original you were trying to protect.

04
Upgrades with a way back
Version upgrades are orchestrated with rollback, and query, connection and replication-lag figures are read live while you watch.

In numbers

1
connection string that survives a failover unchanged

2
endpoints, so reads can go to replicas and writes to the primary

Any second
in the retention window you can restore to

What it does

The engines you run

Postgres, MySQL and MariaDB, Redis and Valkey, and the Mongo / ClickHouse / Kafka trio, each with its own validated version menu and default port on one resource.

Eight engines behind one resource model, so backup, upgrade and failover are learned once and apply to all of them. Each engine carries its own validated version menu — no “latest or nothing” surprises.

HA topology and failover

Standalone, high-availability or cluster chosen rather than implied, with read replicas on a distinct read endpoint and a failover verb that leaves the connection string unchanged.

You choose the topology explicitly. Read replicas answer on their own endpoint for reporting and read-heavy work, and when a primary fails over, the application’s connection string does not change.

Point-in-time recovery

Continuous backup to any moment in the retention window, restored into a new database so a wrong timestamp costs a throwaway rather than the original.

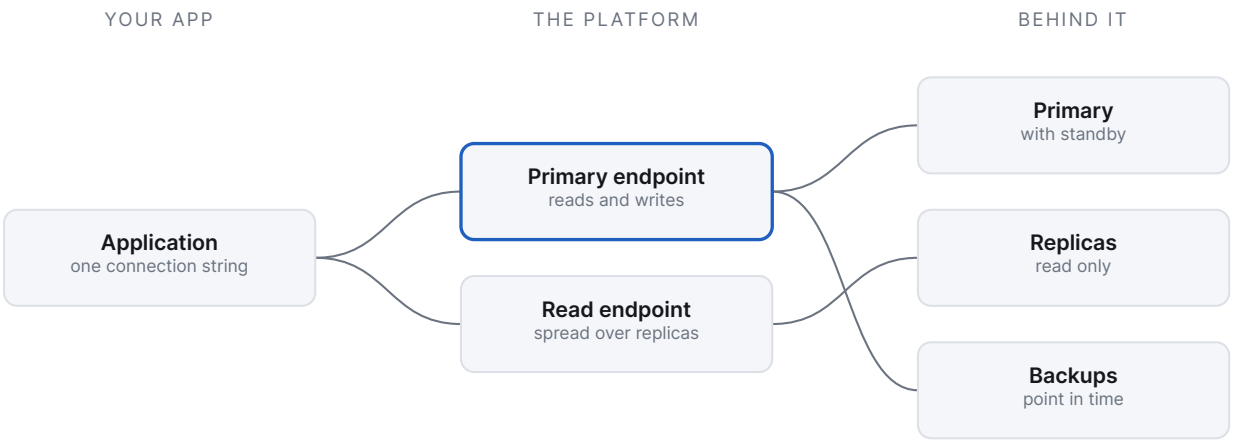
Pick any moment in the retention window and the restore lands in a new database. Compare it, promote it, or throw it away — the original you were protecting is never at risk from the restore itself.

Upgrades, pooling and monitoring

Orchestrated version upgrades with rollback, an optional connection pooler on its own endpoint, and query, connection and replication-lag figures read live.

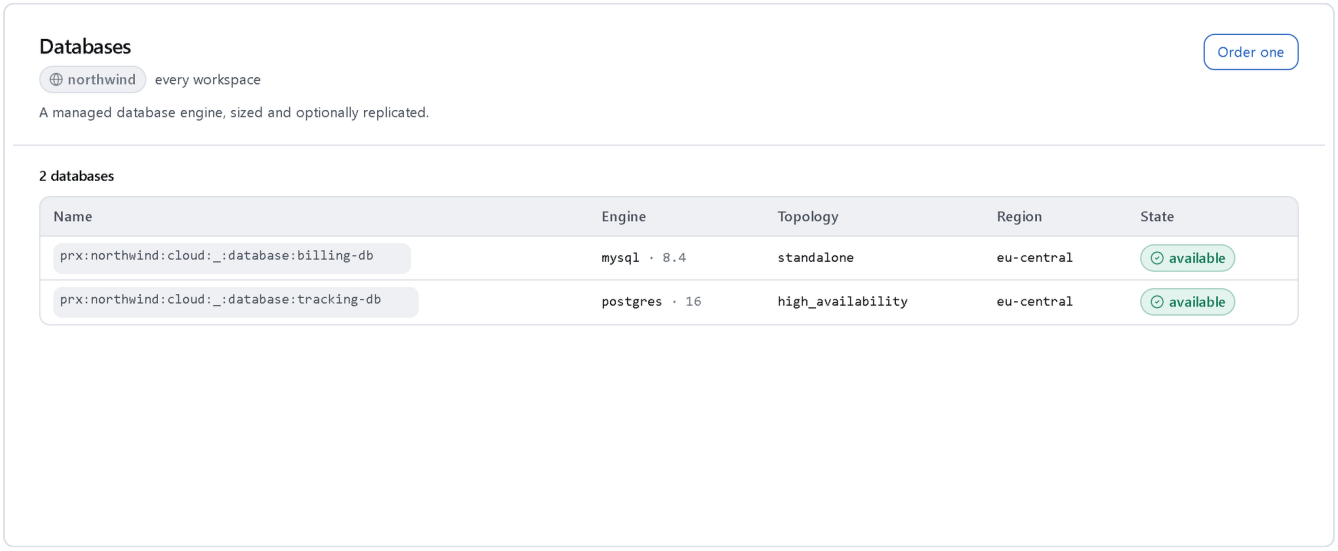
Upgrades are orchestrated with a recorded way back. The optional pooler absorbs chatty applications on its own endpoint, and the figures you need in an incident — queries, connections, replication lag — are read live.

How it fits together

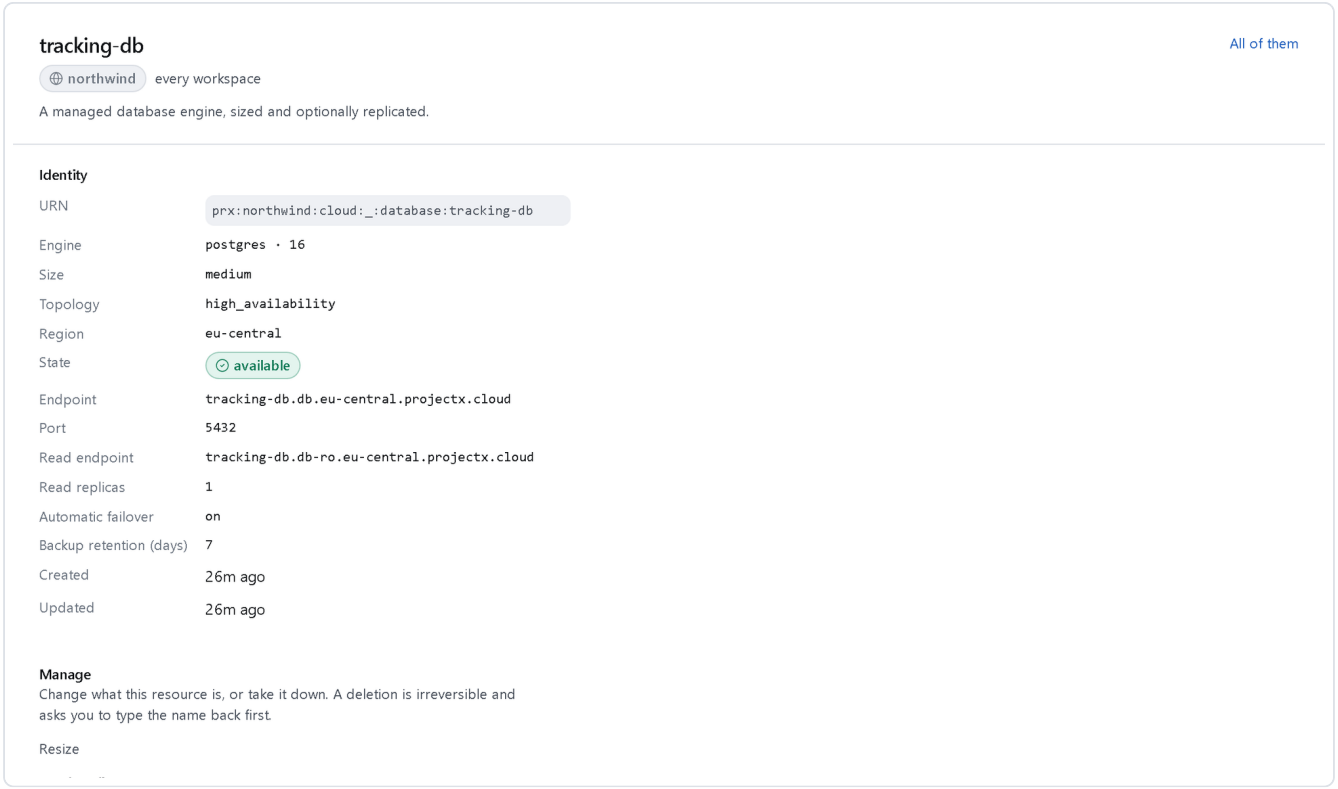


Your application connects to one endpoint for writes and another for reads. Behind them the platform runs the primary and its replicas, fails over without changing the address, and takes the continuous backups a point-in-time restore draws on.

In the console



Every engine in one list, each with its topology and version. Screens shown with demo data.



One database: primary and read endpoints, replicas, failover and backups. Screens shown with demo data.

Key ideas

Topology
Standalone, high availability with a standby, or a native multi-node cluster — chosen when you create the database and changed later as it grows.

Read endpoint
A second address that spreads queries across the read replicas, so heavy reporting does not slow down writes.

Point-in-time recovery
Restore to any moment inside the retention window, as a new database — the original keeps running while you check the restore.

The engines
PostgreSQL, MySQL, MariaDB, Redis, Valkey, MongoDB, ClickHouse and Kafka, each provisioned the same way.

In depth

Availability
High availability keeps a standby ready; failover promotes it and the endpoint follows, so applications reconnect to the same address. A failover drill proves it works before you need it.

Upgrades and tuning
Version upgrades are orchestrated with a way back, parameter groups hold engine settings, and a connection pooler answers on its own endpoint so a busy app does not exhaust the engine.

Watching it
Query performance, connections and replication lag are monitored for every database, on the same alerting as the rest of the platform.

How it works

01 **Pick an engine** · Console
PostgreSQL, MySQL, MariaDB, Redis, Valkey, MongoDB, ClickHouse and Kafka — one resource model for all of them.

02 **Choose the topology** · Console
Standalone, high-availability or cluster, with read replicas on their own endpoint.

03 **Point your app at one endpoint** · Connection string
The endpoint names no upstream, and a failover never changes the connection string.

04 **Sleep at night** · API
Continuous backup with point-in-time recovery, orchestrated upgrades and live monitoring figures.

Who it is for

Product backends

A relational database that survives a node failure.

PostgreSQL or MySQL in an HA topology, with read replicas on their own endpoint.

Caching and queues

Fast in-memory stores without babysitting.

Redis and Valkey as managed resources with the same backup and upgrade story.

Analytics and streams

Columnar analytics and event streams next to your operational databases.

ClickHouse and Kafka on the same resource model and the same invoice.

Works with

Console · API · Connection pooler · Read-replica endpoint

Questions buyers ask

Which engines are available?
PostgreSQL, MySQL, MariaDB, Redis, Valkey, MongoDB, ClickHouse and Kafka — each with a validated version menu on the same resource model.

What does high availability include?
You choose standalone, HA or cluster explicitly. HA brings automatic failover that leaves the connection string unchanged, plus read replicas on a distinct read endpoint.

How do restores work?
Continuous backup lets you pick any moment in the retention window. The restore lands in a new database, so the original is never at risk from the restore itself.

How is it billed?
By the instance-hour for the engine, size and topology at published rates, on the one monthly invoice. The rate card is on the pricing page.

Pricing

Current published rates, in your currency: https://adaydev.com/pricing/cloud