



Building Infrastructure on Postgres forks

Tatiana Krupenya, DBeaver
October 2, 2026



Not every PostgreSQL-compatible database is a fork

TAXONOMY

Fork / derivative

PostgreSQL server source is modified into a separate database.

Examples: Cloudberry, openGauss, IvorySQL.

SOURCE CODE

Enterprise downstream

A maintained PostgreSQL downstream with enterprise, security or Oracle-compatibility features.

Examples: EDB PAS, Fujitsu Enterprise Postgres.

VENDOR PATCHES

Extension

Functionality is installed into upstream PostgreSQL rather than replacing the server.

Examples: TimescaleDB, Citus.

RUNS IN PG

PostgreSQL query layer

PostgreSQL SQL-processing code sits over a different distributed transaction and storage engine.

Example: YugabyteDB YSQL.

PG FRONTEND

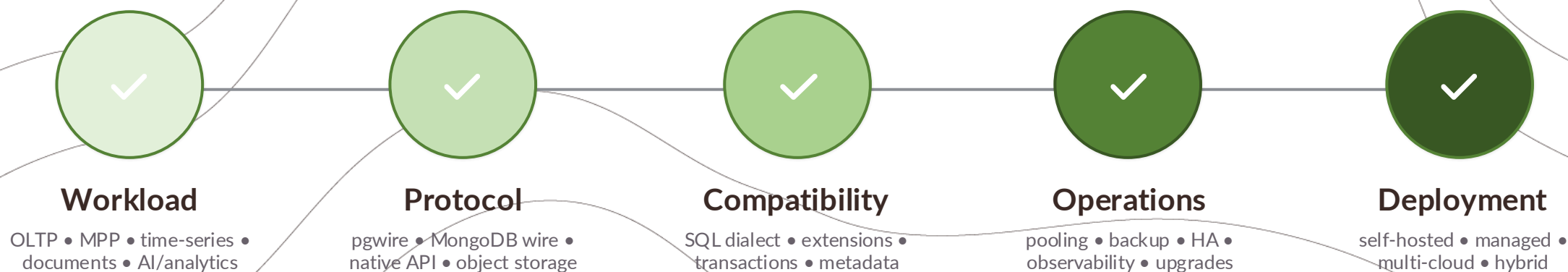
Compatible system / peer

Implements pgwire or serves a similar workload without being a PostgreSQL fork.

Examples: CockroachDB and cloud services with PostgreSQL interfaces.

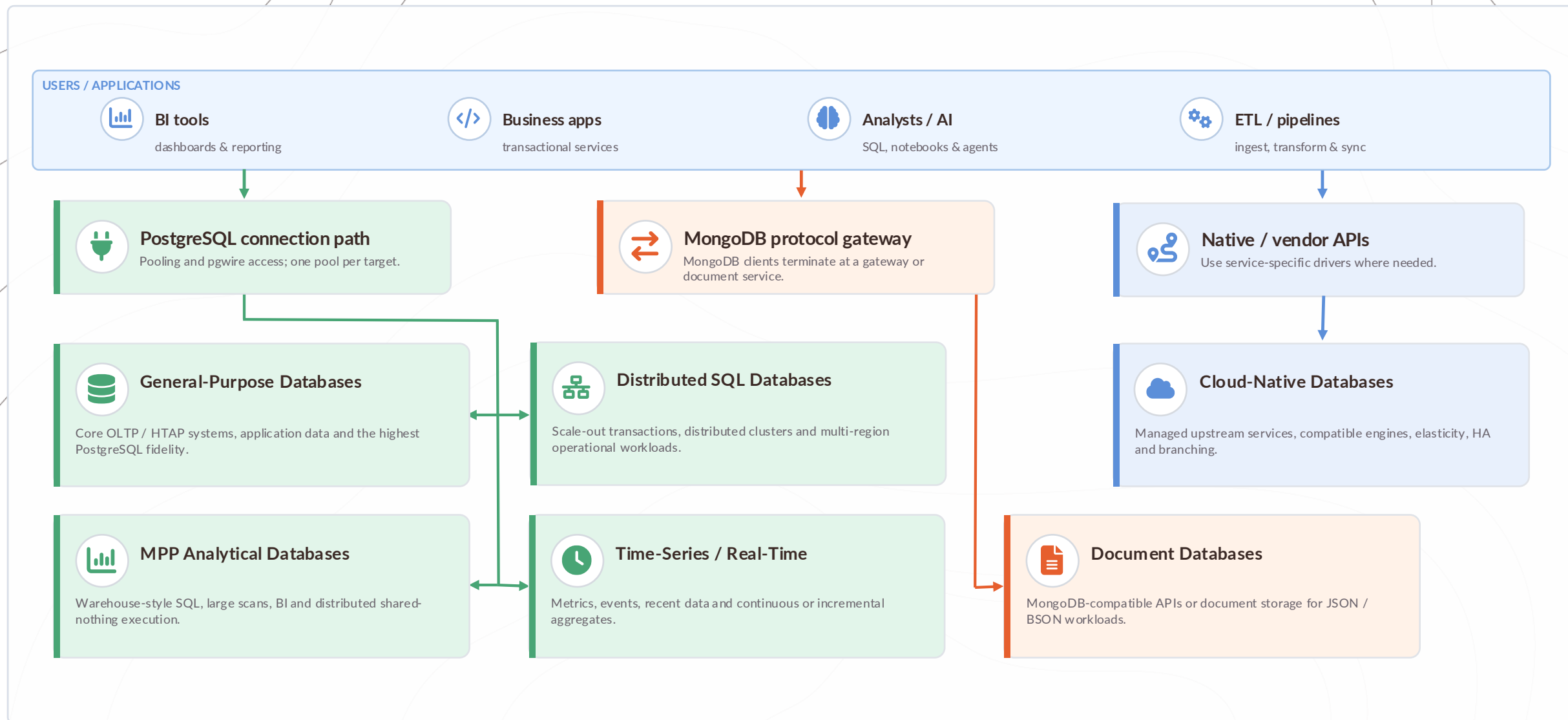
NO PG CODE

Compatibility is an interface property. Fork ancestry is a source-code relationship.



Do not select by logo or protocol alone

A PostgreSQL-compatible interface does not guarantee identical extensions, optimizer behavior, transactions, backup tooling or operational semantics.



Operational PostgreSQL

Upstream, enterprise and cloud-native services for application and mixed transactional workloads.





Operational PostgreSQL distributions by licensing and service model

OPERATIONAL

Community open-source databases

Open-source database code; community / foundation-led

PostgreSQL

Upstream relational database maintained by the PostgreSQL Global Development Group.

POSTGRESQL LICENSE

IvorySQL

Apache 2.0 PostgreSQL fork with built-in Oracle compatibility and active community releases.

APACHE 2.0

openGauss

Enterprise-grade PostgreSQL-derived database maintained by the openGauss community.

MULAN PSL v2

Open-source + commercial services

Open-source code with first-party support, subscription or managed service

Neon

Apache 2.0 cloud-native Postgres core plus a commercial managed serverless service.

OSS CORE + CLOUD

Percona Distribution for PostgreSQL

Fully open-source PostgreSQL distribution with tested HA, backup, security and monitoring components; paid support available.

OSS + SUPPORT

Crunchy Certified PostgreSQL

100% open-source PostgreSQL distribution packaged for enterprise deployment with commercial subscription and support.

OSS + SUPPORT

Proprietary enterprise distributions

Commercially licensed PostgreSQL-derived database distributions

EDB Postgres Advanced Server

Commercial PostgreSQL-derived enterprise edition with Oracle compatibility, security and administration features.

COMMERCIAL

Fujitsu Enterprise Postgres

Commercial enterprise PostgreSQL distribution focused on security, resilience, performance and 24/7 support.

COMMERCIAL

CYBERTEC PGEE

Commercial PostgreSQL Enterprise Edition with TDE, auditing, compliance tooling and enterprise support.

COMMERCIAL



Cloud-native PostgreSQL

MULTI-CLOUD

AWS

Amazon RDS for PostgreSQL

Managed upstream PostgreSQL

MANAGED PG

Amazon Aurora PostgreSQL-Compatible

Proprietary AWS engine with PostgreSQL compatibility

PG-COMPATIBLE

GOOGLE CLOUD

Cloud SQL for PostgreSQL

Managed upstream PostgreSQL

MANAGED PG

AlloyDB for PostgreSQL

Google-built compatible engine • OLTP + columnar HTAP

PG-COMPATIBLE

MICROSOFT AZURE

Azure Database for PostgreSQL

Flexible Server • managed PostgreSQL

MANAGED PG

Elastic Clusters

PostgreSQL + Citus for horizontal scale

PG-COMPATIBLE

Other active projects

Neon

serverless PostgreSQL • compute/storage separation • branching/PITR

PolarDB

Alibaba Cloud vendor engine • shared storage / cloud-native architecture

OrioleDB

open-source PostgreSQL extension and storage-engine project

Azure Cosmos DB for PostgreSQL is on a retirement path; Microsoft directs new distributed PostgreSQL projects toward Elastic Clusters.

Analytics & Distribution

Cloud warehouses, MPP analytical engines and globally distributed SQL systems.





Analytical options: choose by workload and operating model

ANALYTICS MAP

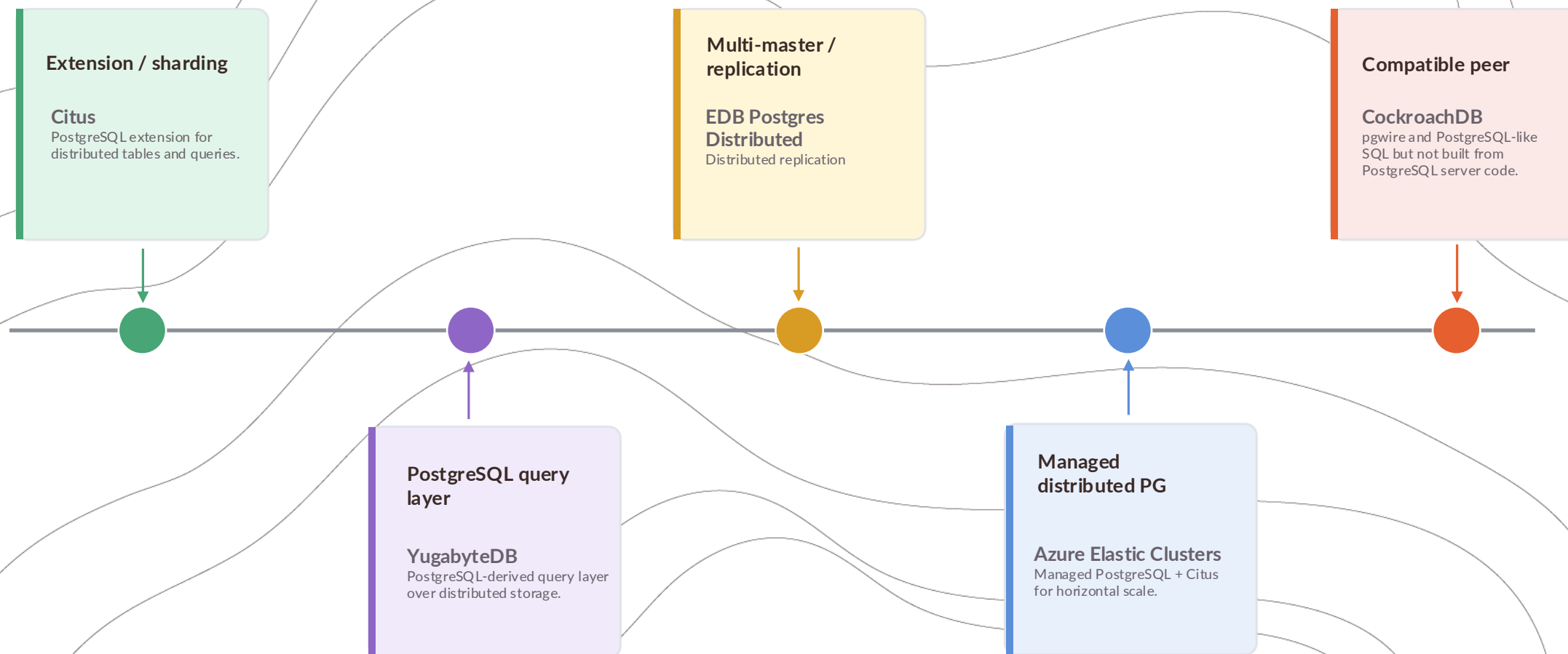


ILLUSTRATIVE POSITIONING • NOT A BENCHMARK



Distributed databases

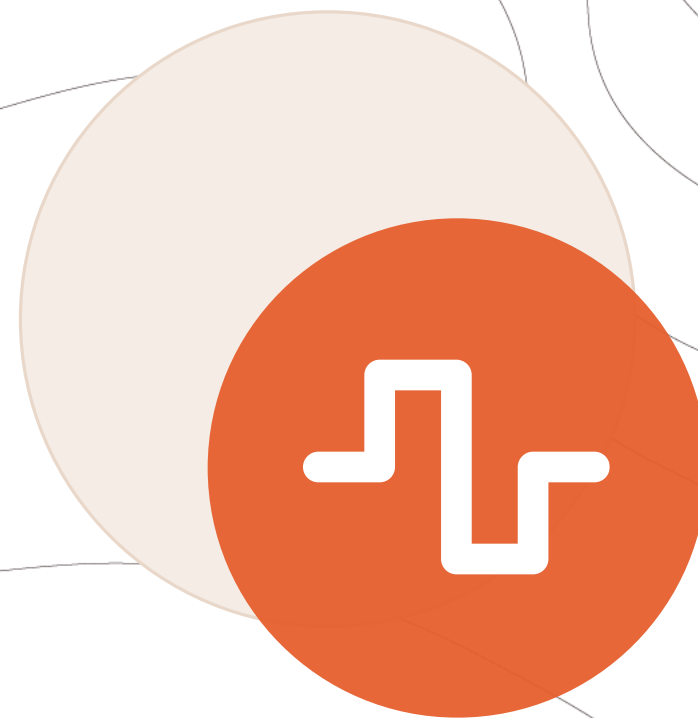
DISTRIBUTED SQL



A PostgreSQL driver may connect to several systems, but transactions, extensions, topology and operations still differ.

Specialized Workloads

Time-series, real-time analytics and MongoDB-compatible document services.





Time-series / real-time analytics

TimescaleDB

PostgreSQL extension
Open-core / Tiger Data
Hypertables, compression and
continuous aggregates

YMatrix

PostgreSQL / Greenplum-derived
Vendor platform
Time-series, OLAP and streaming
workloads



Document / MongoDB-compatible

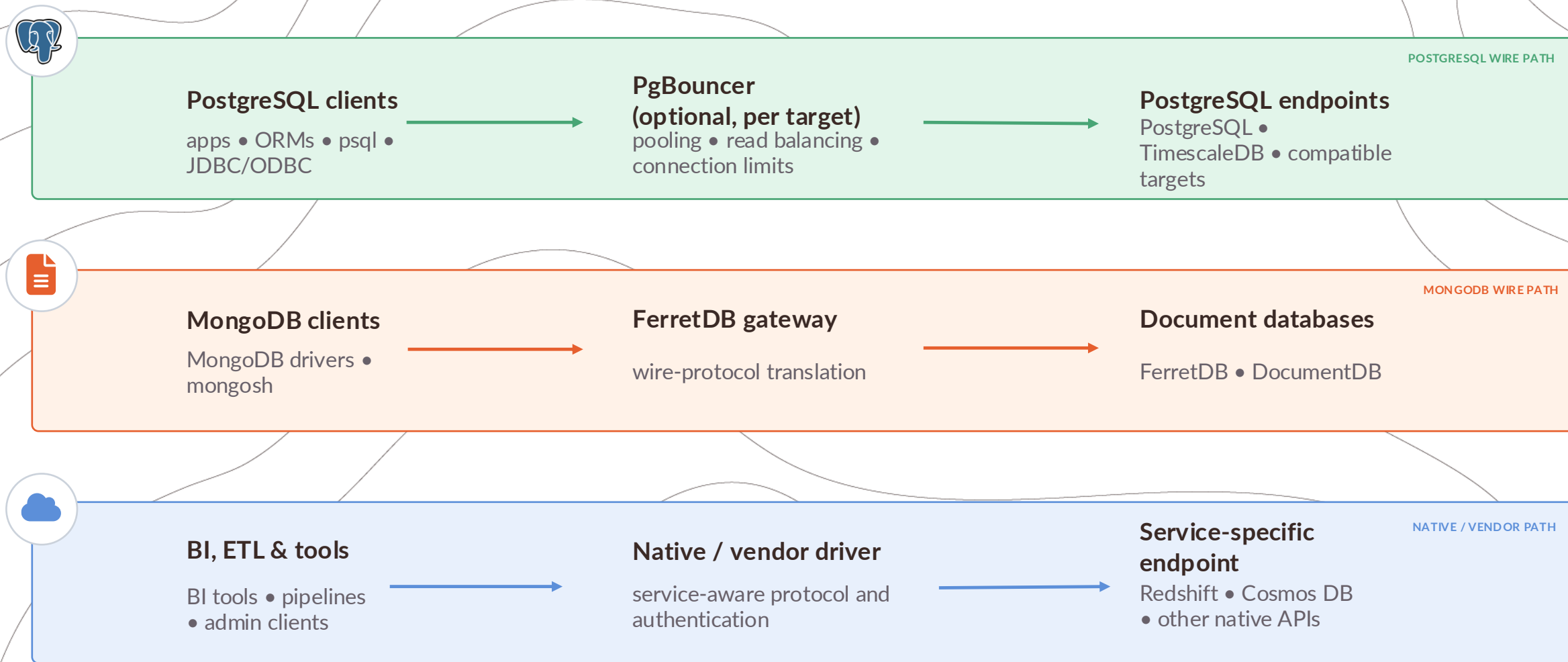
FerretDB

Apache 2.0 gateway maintained
by FerretDB Inc.
MongoDB API over PostgreSQL +
DocumentDB

Azure DocumentDB

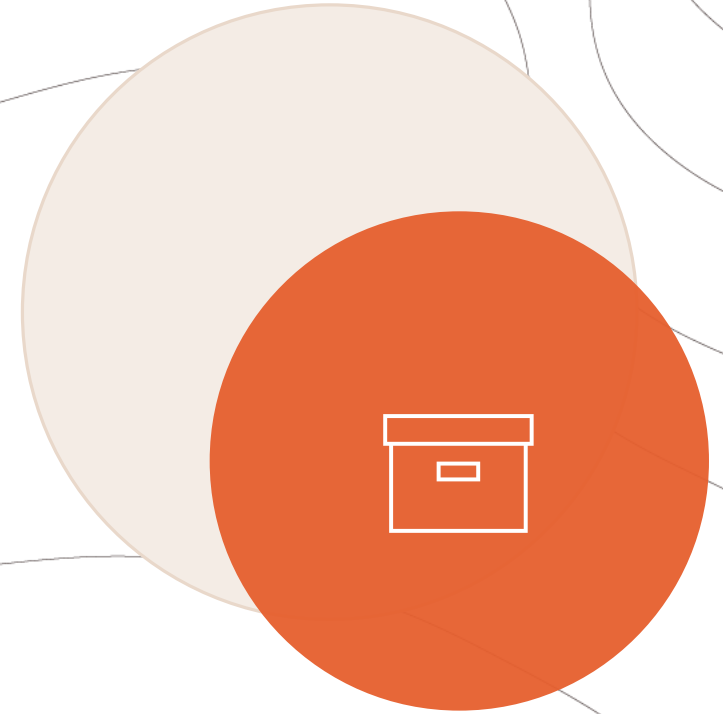
Managed Microsoft document
database built on the
DocumentDB engine

Connection paths: route by protocol and operational behavior



Backups

Logical, physical and snapshots



SELF-HOSTED

Native PG tools for backup and replication

- Your data – your responsibility
- Different forks – different tools
- Plan your backups cadence based on your needs
- Don't forget about security standards

CLOUD PLATFORM

Cloud services for managed Postgres

- If you do this manually you do something wrong
- Vendor lock is not a joke
- Someone else will think about your data safety



Backup & restore command matrix

COMMANDS

Database / variant	Backup command	Restore command	Primary scope
PostgreSQL family	pg_dump / pg_dumpall	pg_restore / psql	Logical
Managed PostgreSQL	pg_dump + provider snapshot	pg_restore + PITR	Logical + managed
Cloudberry / Greenplum	gpbbackup	gprestore	Parallel MPP
YugabyteDB YSQL	ysql_dump / ysql_dumpall	ysqlsh -f backup.sql	Logical YSQL
openGauss	gs_dump / gs_dumpall	gs_restore / gsql -f	Logical
FerretDB	mongodump	mongorestore	MongoDB API
Amazon Redshift	create-cluster-snapshot	restore-from-cluster-snapshot	AWS snapshot

Data Lakes & Lakehouse

Object storage, open table formats and PostgreSQL

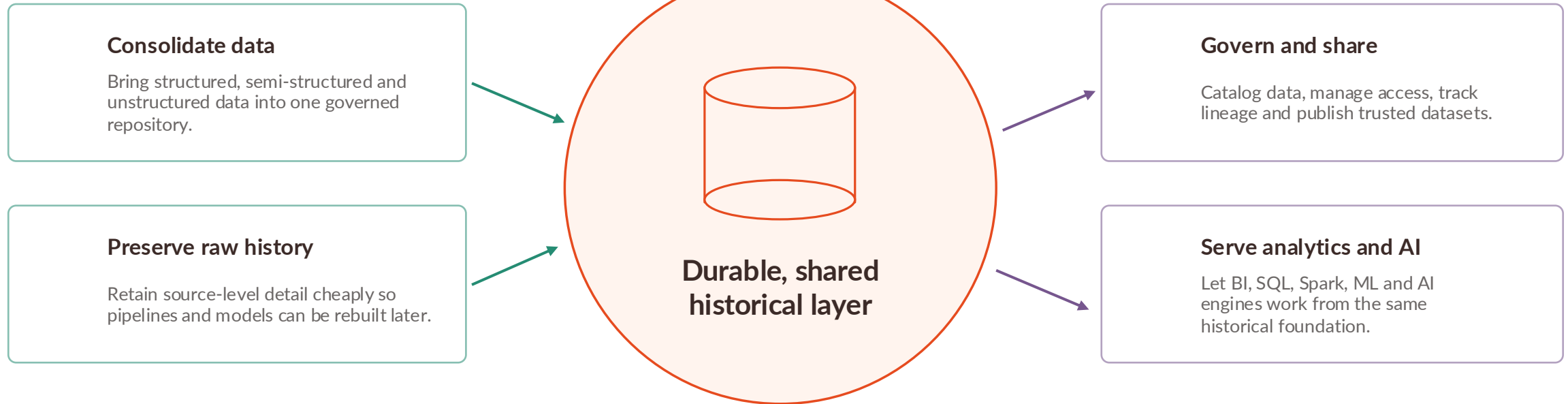




Why enterprises build Data Lakes

PURPOSE &
PLATFORMS

MAIN PURPOSE



PLATFORM EXAMPLES

AWS

Amazon S3 + Lake Formation

Object storage, catalog, security and governed access.

AZURE

Azure Data Lake Storage Gen2

Blob storage with hierarchical namespace and fine-grained security.

GOOGLE CLOUD

Cloud Storage + Iceberg Lakehouse

Open table access and cross-engine interoperability.

DATABRICKS

Delta Lake + Unity Catalog

Lakehouse pattern on cloud object storage.

SNOWFLAKE

Iceberg Tables + Horizon Catalog

Governed open tables on cloud object storage.



Reference architecture

LAKEHOUSE

OPERATIONAL SOURCES

PostgreSQL family

OLTP • enterprise editions

Neon

serverless PostgreSQL • branches

TimescaleDB

hot metrics and events

YugabyteDB / FerretDB

distributed OLTP / MongoDB API



MOVEMENT

CDC / ETL

- Logical replication
- Data transfer
- Batch exports



CENTRAL DATA LAKE

Object storage

Parquet / ORC
• Iceberg catalog

S3 • ADLS •
GCS • MinIO



CONSUMPTION

Cloudberry

PXF external tables

Amazon Redshift

Spectrum over S3

Calcite / Trino / ML

Multi-engine processing



Advantages and disadvantages for Data Lakes

LIMITATIONS

EASIER WITH A PG-CENTERED ESTATE

Common skills and integration patterns

- Shared SQL vocabulary, schemas and data types
- PostgreSQL drivers, JDBC/ODBC and familiar BI tools
- Logical replication and CDC patterns across many variants
- Simpler schema mapping into Parquet / Iceberg models
- Reusable administration and observability knowledge
- FDWs can federate selected external sources for SQL access

LOWER INTEGRATION FRICTION

STILL REQUIRES A DATA PLATFORM

Storage, governance and recovery boundaries

- No common physical table or file format across engines
- No automatic cross-database transactions or query optimizer
- No shared catalog, lineage, data quality or access policy
- No universal backup, snapshot or point-in-time recovery model
- No automatic workload routing between OLTP, time-series and MPP
- Protocol compatibility does not guarantee extension compatibility

GOVERNANCE IS SEPARATE

AI Agents

Typical workflow, advantages and disadvantages



REQUEST LAYER

AI agent / developer tool

Typical tasks

- Inspect schemas and relationships
- Generate SQL, migrations and scripts
- Analyze data and query plans
- Propose changes

NATURAL LANGUAGE → TOOLS

CONTROL LAYER

Database tool gateway

- 1 Discover** information_schema + pg_catalog
- 2 Profile** engine, version, features and limits
- 3 Validate** SQL parser, policy checks and EXPLAIN
- 4 Execute** read-only, branch/test or approved write

COMMON CORE + ENGINE PROFILE

EXECUTION TARGETS

Postgres-based estate

PostgreSQL / Neon

OLTP, branches and test environments

TimescaleDB / Citus

specialized extensions and scale-out

YugabyteDB / Cloudberry

distributed SQL and MPP analytics



Advantages for AI agents

ADVANTAGES



One discovery model

information_schema and pg_catalog cover most schema, relation and privilege discovery.



Reusable SQL and drivers

PostgreSQL clients, ORMs, JDBC/ODBC and scripting patterns can be reused across many targets.



Consistent safety controls

Roles, row-level security, read-only transactions, statement timeouts and lock timeouts.



Plan-driven feedback

EXPLAIN provides machine-readable plans for checking scans, joins, estimates and query cost before execution.



Disposable test environments

Neon branches let agents test migrations and destructive queries without changing the parent branch.



Relational + vector context

pgvector keeps embeddings next to transactional data, joins, filters and access-control logic.

Result: fewer adapters, more reusable prompts and policies, and easier validation across the database estate.



Disadvantages and design constraints

RISKS

WHERE COMMONALITY BREAKS

Database and platform differences still matter



Compatibility is uneven

SQL, extensions, catalogs and semantics differ across MPP, distributed and cloud-derived engines.



Operations still diverge

Backup, restore, failover, scaling and control-plane APIs remain product-specific.

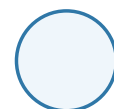


Session behavior can surprise agents

PgBouncer transaction pooling breaks several session-scoped features and assumptions.

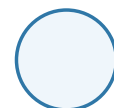
AI-SPECIFIC RISKS

A common protocol is not a safety boundary



Wrong engine for the workload

A Postgres-only bias can force object storage, large OLAP or document workloads into the wrong system.



Generated SQL can be unsafe

Hallucinated columns, unbounded scans, DDL and destructive statements still require validation and approval.



Context becomes stale

Schema changes, privileges and runtime statistics can invalidate cached agent assumptions.

1

Choose by workload

OLTP, distributed SQL, MPP analytics, time-series and document workloads need different engines.

2

Reuse the common PostgreSQL layer

Shared SQL skills, drivers, metadata patterns and tooling reduce integration friction across the estate.

3

Keep platform boundaries explicit

Compatibility does not unify backup, HA, data lakes, pooling or AI-agent safety controls.

POSTGRES- FIRST

not Postgres-only

Thank you!