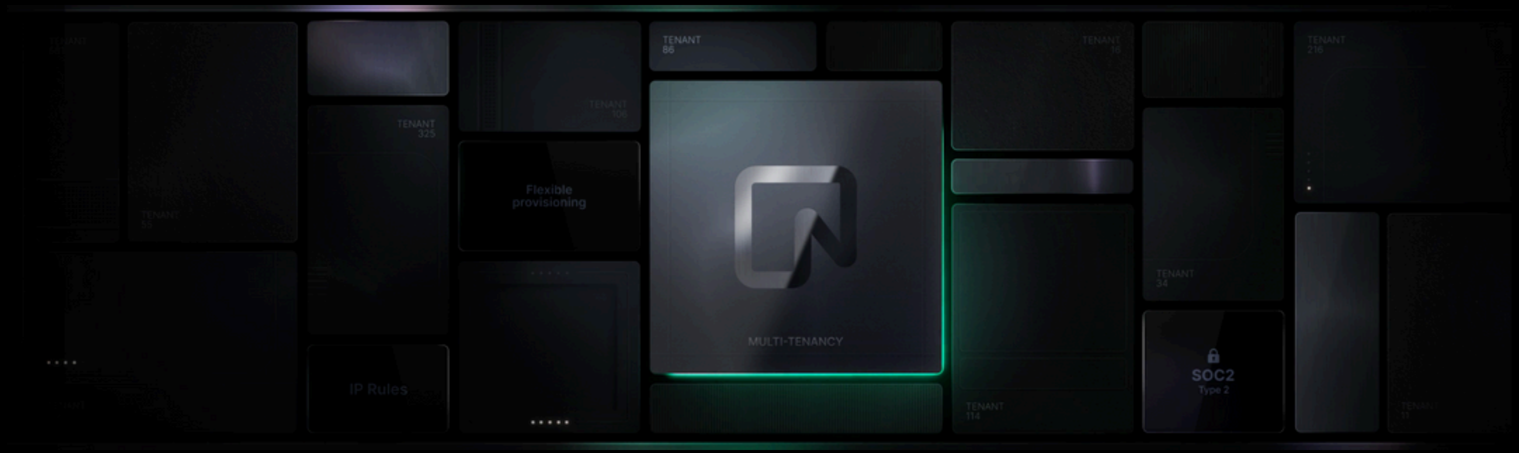
NEON

Introduction to Neon

Serverless Postgres



HASURA replit CLOUDFLARE Vercel Koyeb WunderGraph

Why serverless Postgres?

Neon takes the world's most loved database — Postgres — and delivers it as a serverless platform, enabling teams to ship reliable and scalable applications faster.

Enabling serverless Postgres begins with Neon's native decoupling of storage and compute. By separating these components, Neon can dynamically scale up during periods of high activity and down to zero when idle. Developers can be hands-off instead of sizing infrastructure manually.

This serverless character also makes Neon databases highly agile and well-suited for use cases that require automatic creation, management, and deletion of a high number of Postgres databases, like database-per-user architectures with thousands of tenants, as well as database branching workflows that accelerate development by enabling the management of dev/testing databases via CI/CD.

What "serverless" means to us

We interpret "serverless" not only as the absence of servers to manage but as a set of principles and features designed to streamline your development process and optimize operational efficiency for your database.

To us, serverless means:

- **Instant provisioning:** Neon allows you to spin up Postgres databases in seconds, eliminating the long setup times traditionally associated with database provisioning.
- **No server management:** You don't have to deal with the complexities of provisioning, maintaining, and administering servers. Neon handles it all, so you can focus on your application.
- **Autoscaling:** Compute resources automatically scale up or down based on real-time demand, ensuring optimal performance without manual intervention.
- **Usage-based pricing:** Your costs are directly tied to the resources your workload consumes—both compute and storage. There's no need to over-provision or pay for idle capacity.
- **Built-in availability and fault tolerance:** We've designed our architecture for high availability and resilience, ensuring your data is safe and your applications are always accessible.
- **Focus on business logic:** With the heavy lifting of infrastructure management handled by Neon, you can dedicate your time and effort to writing code and delivering value to your users.

Separating storage and compute

Transitioning to Neon simplifies scaling and managing Postgres by rethinking the interaction between storage and compute:

- Neon separates storage and compute, allowing it to run an untouched version of Postgres on compute nodes while using a custom-built storage system that preserves database history.
- This architecture enables database branching, allowing for easy management of separate environments for development, staging, and testing.
- With storage decoupled, compute nodes in Neon are stateless and shut down after 5 minutes of inactivity, automatically spinning up when needed.
- Neon's design also allows for instant point-in-time recovery and seamless branching. All objects can be managed programmatically via APIs.

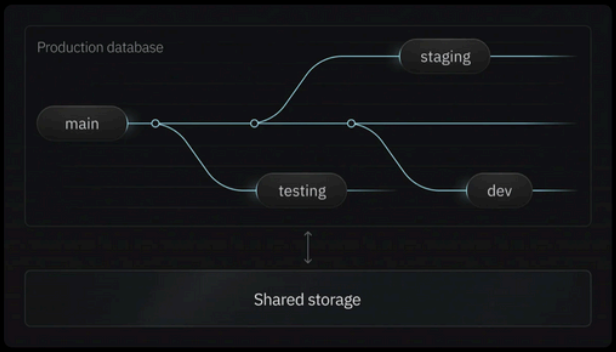
Storage

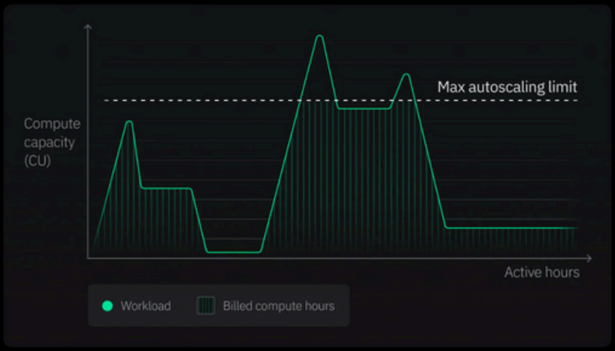
Branch-based Postgres

Databases in Neon are brancheable (both schema and data). Think of code branches, but for your data.

Branches share storage

When you branch a database, the new branch won't add to your storage bill. Branches are ready instantly, no matter how large the dataset.





Compute

Serverless compute

Compute size is measured in CUs. Database branches autoscale from 0.25 to 10 CU based on load and down to zero when inactive.

Compute usage is measured in compute hours

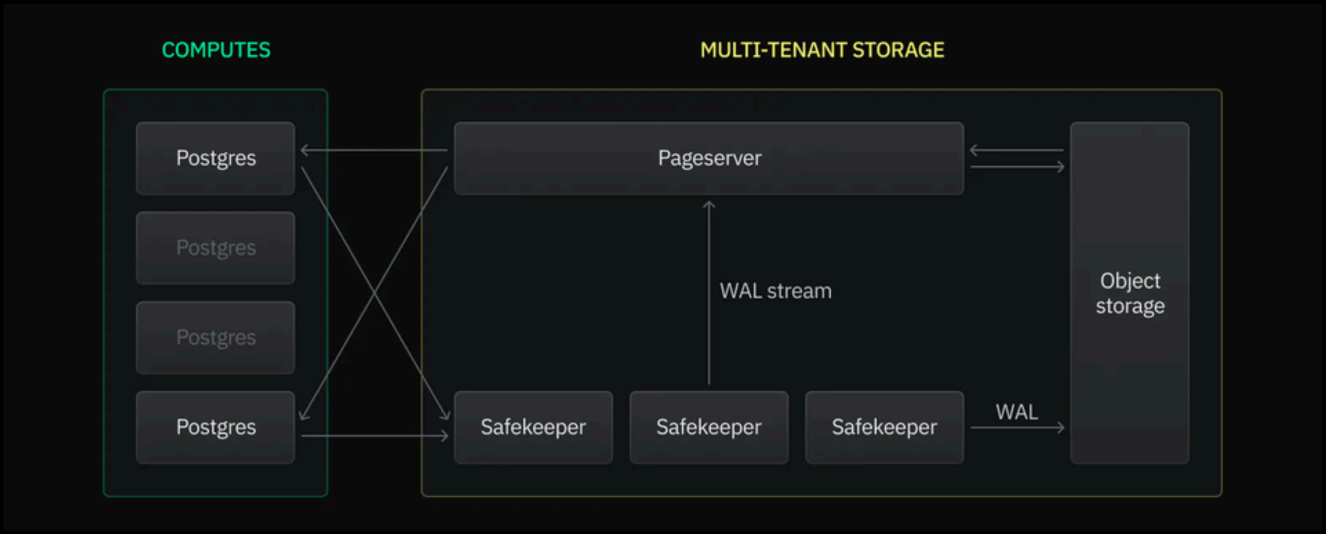
Example: a 4 CU compute running for 20 hours uses 80 compute hours. All monthly plans include generous usage, with extra compute hours billed separately.

Storage design

At a user level, you interact with Neon exactly as they would do with a traditional Postgres database, using the same SQL syntax, functions, and extensions. But under the hood, Neon is running on a custom-built storage system that adapts Postgres to modern workflows and cloud-native environments.

Compared to the traditional implementations of Postgres, Neon's approach to storage is unique in three ways:

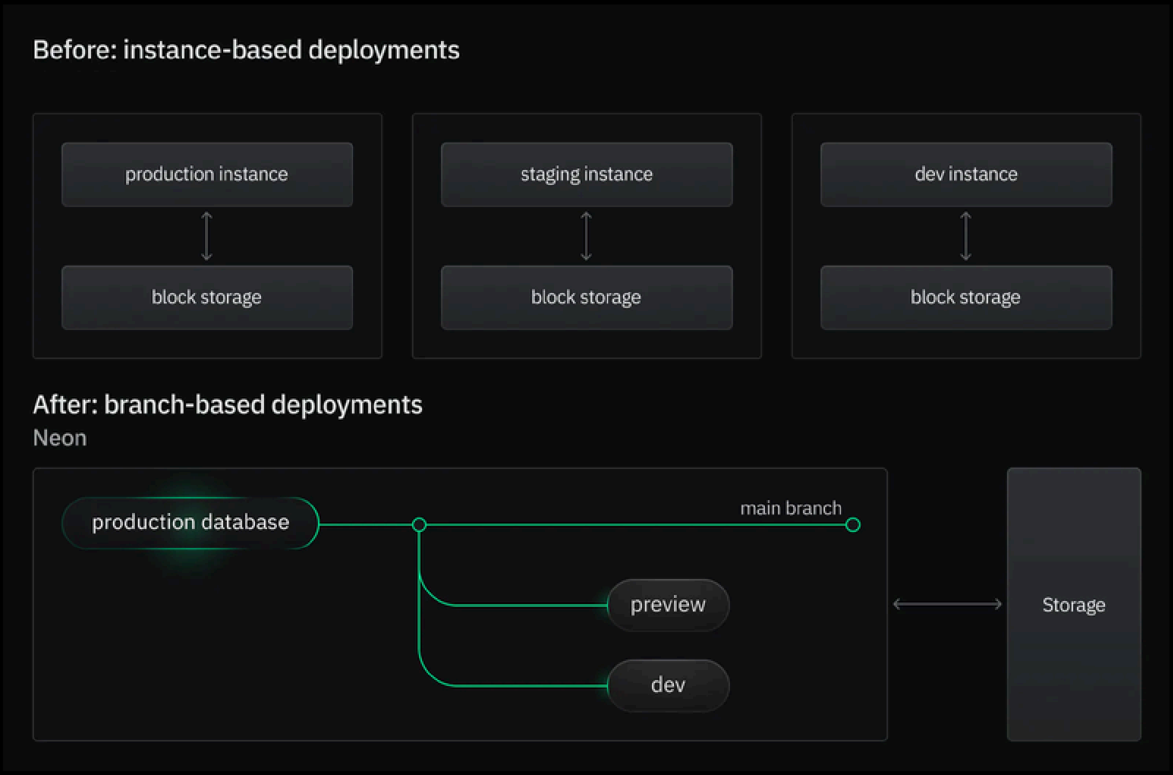
- Neon uses object storage for long-term data storage and locally attached SSDs for low latency, high-performance data.
- Neon's storage engine is modeled as a key-value store. This approach makes Neon databases very robust against failure, and it introduces many benefits around developer productivity - for example, it eliminates the need for keeping traditional backups and WAL archiving, and it allows for efficient Point-In-Time-Recovery (PITR) and database branching.
- Postgres' WAL is streamed to Neon's cloud-native storage over the network. Two separate layers (safekeepers and pageservers) are used to store and process WAL. When an ingest/update/delete operation is run in Postgres, Neon processes the incoming WAL by first sending it to safekeeper nodes, which act like a first-line storage system for the WAL ensuring that no data is lost even in the event of a crash or failure in the compute. This WAL is then processed in the pageservers, which transform it into a custom storage format compatible with object storage.



Database branching

Neon storage design enables one of the best Neon features: database branching.

- When a branch is created in Neon, it leverages the copy-on-write technology. The new branch does not duplicate data but serves as a pointer to the existing data state at the time of branching. This approach ensures efficiency in storage usage and the instant creation of branches, regardless of the database size.
- When a branch diverges from its parent, the changes are recorded in new WAL entries specific to that branch. The pageservers then process these WAL records, applying changes to a separate set of immutable files dedicated to the branch. This isolation ensures that transactions in one branch do not impact any other branch or the main database.
- The pageservers manage data storage efficiently by only storing the deltas — the changes from the parent branch. This copy-on-write mechanism minimizes additional storage requirements, as only the differences from the parent branch are stored.
- When queries are run against a branch, the pageservers first look into the branch-specific changes. If the requested data is not found within the branch's diverged state, the query seamlessly falls back to the parent data state, ensuring a unified view of the database up to the point of branching.



Compute w/ autoscaling

Neon's serverless experience is made possible by the nature of Neon's compute nodes, which are:

- **Stateless**. Since all stateful data is stored separately, multiple compute units can be attached to a single durable storage. If a compute instance shuts down, it can be quickly recreated without data loss, leveraging the separated storage layer.

- **Ephemeral**. When Neon branches are not being accessed, they can scale to zero and stop consuming compute resources. Users can configure this behavior, enabling or disabling it and/or configuring the period after which an instance will be considered idle and scaled to zero.

Compute autoscaling for performance and efficiency

Variable load patterns are common in applications, but traditional managed databases require provisioning a fixed amount of CPU and memory. With its serverless compute architecture, Neon solves this:

- Neon autoscales according to traffic, dynamically adjusting CPU and memory as needed.

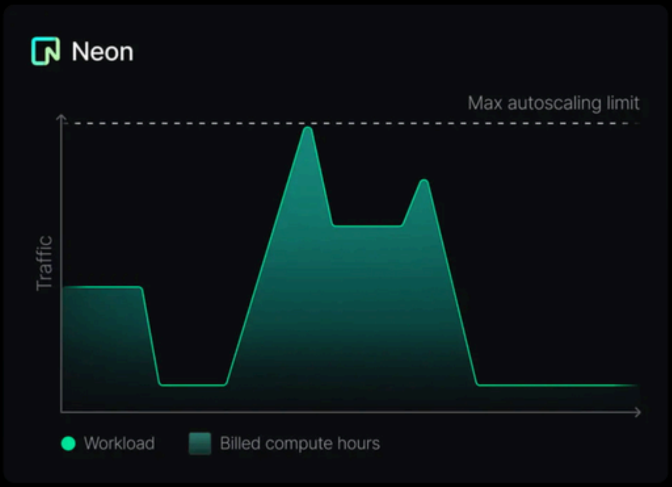
- Costs are controlled by setting a max autoscaling limit, avoiding unexpected charges.

- Developers get fast performance in production without overpaying. In a typical compute bill, 60% of costs go towards unused resources.

- There's no manual resizes or downtimes. Neon scales up and down smoothly and immediately.

- Non-prod databases scale to zero when inactive. Instead of paying for compute 24/7, you skim the costs of your supporting databases to a minimum.

- It all is built ransparency with open-source architecture.



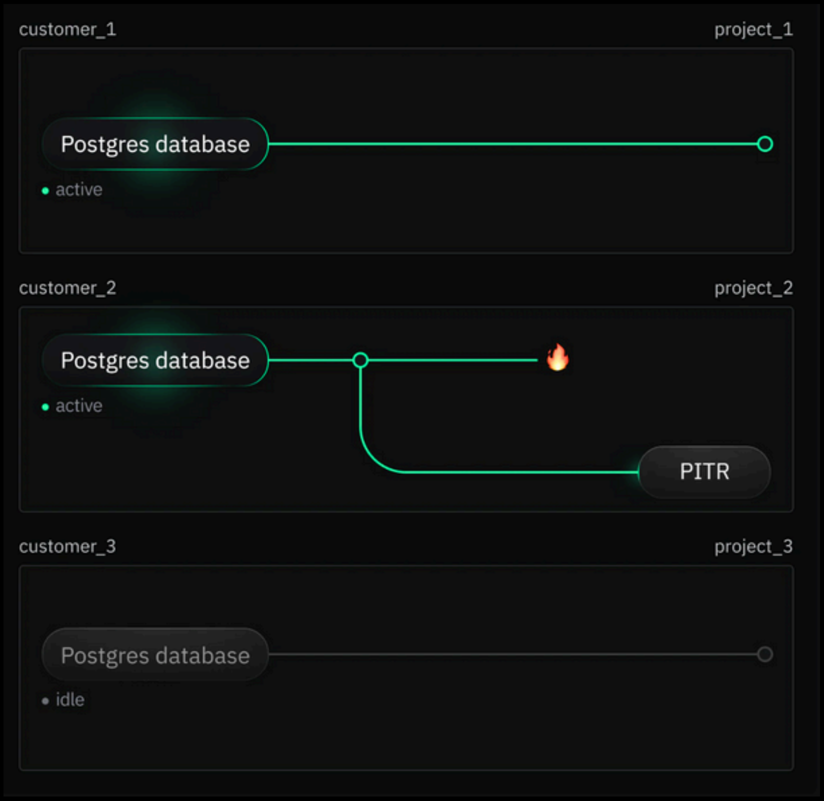
Multi-tenancy in Neon

Neon shines for handling applications that demand full data isolation, i.e. to keep one database per customer on their separate instance.

This architecture doesn't only assure that data from different customers can never mingle with each other, but it also allows them to create different encryption keys for different databases, scale them independently, run restores for specific customers without affecting operations, and upgrade them at different times if necessary.

In Neon, you can easily accomplish a design with full isolation by creating one project per customer. Such Postgres fleets are seamless to manage in Neon; there's no need to manage servers or to allocate storage. Every customer database gets scaled independently, consuming only the compute resources they need. When inactive, compute endpoints go idle automatically. Everything can be automated via the Neon API.

Since Neon supports database branching at a project level, branches can be used to run instantaneous point-in-time restores for particular customers.



Postgres for AI

Vector databases enable efficient storage and retrieval of vector data, which is an essential component in building AI applications that leverage Large Language Models (LLMs).

Neon supports the pgvector open-source extension, which enables Postgres as a vector database for storing and querying embeddings. This means you can leverage the open-source database that you trust as your vector store and forget about migrating data or adding a third-party vector storage solution.

Why serverless Postgres is good for AI

- **Fast index build.** Neon's serverless architecture with autoscaling ensures top performance, enabling developers to build HNSW indexes without specifying instance size.
- **Cost-effective.** Neon requires a lower entry cost v with up to 75% less compute costs than the compeition for typical deployments.
- **Boosted time-to-market.** Neon's database branching feature allows developers to drastically cut down development time and enable faster iteration cycles.
- **API-driven management.** Even small engineering teams can manage thousands of databases, scaling AI applications effortlessly.
- **Strong data isolation.** Neon enables database-per-customer architectures, which enhance data security and privacy, essential for LLMs.

Why pgvector

- Startups can leverage SQL, reducing the learning curve and saving time and resources.
- By using a single database API vs integrating multiple specialized databases, startups can simplify their tech stack and streamline operations.
- Postgres integrates easily with existing code and infrastructure, speeding up time to market.
- AI startups can ensure data integrity and security with Postgres' robust capabilities.
- The flexibility of Postgres allows AI startups to adapt quickly to changing market demands and maintain a competitive edge.

Visit neon.tech for more

The Neon serverless driver, designed for fast queries over HTTP

```
import { neon } from '@neondatabase/
serverless';
const sql = neon('postgresql://
usr:pass@proj.us-east-
2.aws.neon.tech/db');
const posts = await sql('SELECT * FROM
posts');
```

Reach out to us
contact@neon.tech