Microsoft

Key considerations for running modern enterprise applications on PostgreSQL

Contents

Executive summary 1
Introduction: Is PostgreSQL suitable for mission-critical apps? 2
Performance and scalability 4
Reliability and availability 6
Security 8
Adaptability and extension framework 9
Developer productivity: Relational semantics, JSON, and more 10
Deployment and management: on-premises or any cloud 12
Access to skilled resources 13
Benefits of the cloud 15
Summary 17

Executive summary

Today, more than ever before, data and databases are at the center of most enterprise applications. And open-source software is everywhere, at all levels of the application stack—leading many companies to ask themselves whether PostgreSQL is ready for mission-critical enterprise applications.

The answer is yes: PostgreSQL meets all key requirements for modern, mission-critical enterprise applications.

- **Performance and scalability** – The PostgreSQL engine delivers proven performance across both transactional and analytics workloads. It also scales up well across multiple cores. And with Citus, an open-source PostgreSQL extension, PostgreSQL can scale out to meet even the most demanding enterprise workloads.
- **Reliability and availability** – PostgreSQL has a proven architecture and is broadly recognized for its reliability. It also delivers all the high-availability features necessary for maintaining business continuity—including physical replication, logical (transactional replication), easily managed failover, backup, and point-in-time recovery.
- **Security** – PostgreSQL integrates with existing authentication mechanisms to provide fine-grained, role-based authorization and access control for all users, including the ability to limit access to data on a per-row or per-column basis. PostgreSQL also supports strong encryption, enabling you to protect sensitive data both at-rest and in-transit.
- **Adaptability and extensibility** – Every year, PostgreSQL ships a major release that includes broadly desired new features. In addition, PostgreSQL can be easily extended to add new functions, aggregates, data types, operators, and more. If you need an extension, the odds are good that someone else in the PostgreSQL community has already written the code, or at least a large part of it, and then shared it with the rest of the community to freely use.
- **Relational semantics and nonrelational data** – PostgreSQL combines powerful relational semantics with the flexibility of JSON to let you work with both structured and semi-structured data in a highly performant manner. And PostgreSQL is easy to put to use, with a broad range of useful tooling and frameworks across all levels of the database development stack.
- **Deployment and management** – PostgreSQL integrates with most popular open-source deployment and DevOps frameworks. It's easily containerized and can run anywhere—including on-premises data centers, hosted VMs in the cloud, or on fully managed PostgreSQL-as-a-service environments. PostgreSQL also includes comprehensive management and monitoring tools, with straightforward processes for patching and major version upgrades.
- **Access to skilled resources** – PostgreSQL has a well-established ecosystem, with plenty of skilled developers, development partners, DBAs, support partners, and cloud providers. Of course, you'll want to look for those companies whose technical staff are actively involved with maintaining PostgreSQL, or who are creating extensions that power its ecosystem.

Finally, it's worth pointing out that, when you choose to a fully-managed PostgreSQL service in the cloud, you'll have access to additional capabilities that make PostgreSQL perform and scale even better, further increase availability and security, and further simplify deployment and management.

Introduction: Is PostgreSQL suitable for mission-critical apps?

In today's digital economy, the term mission-critical is used to describe systems that support essential business functions, such as online banking, e-commerce, order processing, logistics, and enterprise resource planning. With virtually all such systems based on software, they're often called "modern enterprise applications." Regardless of what they're called, the essential role such applications play in keeping the business up-and-running remains the same. If the system goes down, the entire business—or at least a major part of it—grinds to a halt: orders can't be taken, products can't be shipped, vendors can't be paid, or reports for decision-making can't be run.

It's for this very reason that companies must be selective when choosing a platform for their enterprise applications. It's also why some continue to run parts of the business on platforms that, while they may be 20 years old, remain in use because they've proven their ability to stay up-and-running. But at some point, legacy applications need to be modernized or new ones need to be developed, forcing all companies to eventually ask themselves, "What platforms are viable options for our next enterprise application?"

On the following pages, we'll attempt to answer that question for PostgreSQL, one of the most popular open-source databases in the world. After all, today, more than ever before, data and databases are at the center of most enterprise applications. And open-source software is everywhere, at all levels of the application stack. Given that relatively new features such as native JSON support and horizontal scalability have sparked a resurgence in the popularity of PostgreSQL over the last few years, it makes sense to ask, "Is PostgreSQL ready for modern enterprise applications?"

The short answer is that yes, it is. Companies like Goldman Sachs, European Space Agency, Adobe, Salesforce, VMWare, TomTom, Pandora, Mastercard, and NTT have all publicly stated they are running on PostgreSQL. But like with all such systems, you'll need to do it right. To that end, in the rest of this paper, we'll examine the suitability of PostgreSQL for enterprise applications in the context of several key requirements:

- **Performance and scalability** – Can PostgreSQL handle modern, high-throughput enterprise application workloads? What features and capabilities allow it to do so?
- **Reliability and availability** – How can you ensure your PostgreSQL database will stay up-and-running? Does it provide the essential mechanisms needed to ensure business continuity?
- **Security** – What security mechanisms are built into PostgreSQL? How can you effectively limit access to the database—and to the specific data it contains—to no more than what each user or user role needs to do their jobs?
- **Adaptability and extensibility** – PostgreSQL is known to be extensible, but what exactly does that mean? How will its native extension framework help you build a solution that not only does all you need today, but remains flexible enough to support unknown future needs?
- **Relational semantics and support for nonrelational data** - Does PostgreSQL offer all the necessary relational semantics, such as ACID transactions, joins, two-phase commits, and foreign keys for referential integrity? And is it flexible enough to support semi-structured or unstructured data?

- **Developer tools and productivity** – What will you need for rapid, effective development on PostgreSQL? Can you leverage the tools and skills you already have in-house, or will you need to acquire and learn entirely new ones?
- **Deployment and management** – How can you predictably deploy and manage PostgreSQL? Will you need to piece together your own manual, ad-hoc processes, or does PostgreSQL work well with modern containerization, orchestration, management, and monitoring tools? And will you need to do things differently depending on where you choose to deploy?
- **Access to skilled resources** – Will you have access to the resources you need? How deep is the pool of skilled PostgreSQL developers and development partners? And what about support?

Finally, after we explore PostgreSQL in the context of each of the above requirements, we'll examine how many of them become easier to meet—and deliver significant additional benefits—when you deploy your modern enterprise applications on a fully managed PostgreSQL service in the cloud.

Performance and scalability

In the past, mission-critical workloads were largely transactional—supporting mainly back-office processing. Now they're far more varied and interactive, with employees in all sorts of roles using them throughout the day. What's more, today more than ever, modern enterprise applications are used to reach outside the walls of the enterprise to connect with and serve customers, integrate with suppliers, and support other ways of doing business in real-time. Multitenant SaaS apps, streaming real-time analytics, and hybrid transactional/analytics processing (HTAP) workloads are all valid examples.

Regardless of what an enterprise application does, it needs to perform well. App developers typically blame the database when the app is slow, and backend/database developers typically blame the way the app is written. But the truth is that an app can perform no faster than its database allows, which is why it must deliver the throughput needed to support highly responsive user experiences that are rich in data and interactivity. To achieve this, you'll need to ensure that the entire application stack—especially the database—remains performant and scalable, even with ever-growing data sets.

This requires examining several areas, starting with the database engine itself. How optimized is it? Does it have the necessary performance features, such as rich indexing, just-in-time query compilation, and query parallelization? Next, how well does the database engine scale across larger and larger machines or VMs? Finally, when scaling up is not feasible or practical, can PostgreSQL scale out across potentially dozens of nodes, and how difficult is this to implement and maintain?

Database engine

PostgreSQL excels in all of these areas, beginning with the database engine itself. In fact, since it's been around for more than 25 years, its proven, time-tested performance across both transactional and analytics workloads is pretty much a "given" by now.

Some performance features in PostgreSQL, such as indexing, caching, cost-based query optimization, and multiversion concurrency control (MVCC) have been there from the earliest days. Over the years, new index types have been added, each using a different algorithm that's best-suited to certain types of queries. With the B-tree, Hash, Gist, GIN, and BRIN index types in PostgreSQL, you'll have the means to index any type of data—including geospatial and entire JSON objects—and deliver fast, low-latency queries regardless of workload or data type.

Other performance features came later, having been added and hardened over subsequent years. Key examples include parallel queries, just-in-time (JIT) compilation, table partitioning, read replicas, unlogged (in-memory) tables, and extensions that facilitate connection pooling (via PgBouncer).

Which of these features you're more likely to put to use will depend on your workload. For example, features such as parallel queries and JIT compilation are typically more useful for analytics workloads. Table partitioning, which is often done based on timestamp, can help speed-up indexing across both workload types.

Scaling up

Over time, as you enrich the functionality of your application and its usage grows, so will your need for more database throughput. Often, the amount of memory available to your database becomes a performance bottleneck. One of the fastest and easiest ways to alleviate this is by scaling up to a larger database server or VM, with additional processor cores and more memory. Again, this is an area where PostgreSQL has continually improved over its long history, to the point where it now scales beyond

100 cores for many common workloads. Proponents of Oracle used to claim better performance than PostgreSQL on larger machines, but that claim has fallen silent.

Scaling out

There are several reasons why you may want (or need) to scale out:

- You've scaled up as much as you can and still need more memory and compute to achieve your desired throughput. If you have a read-heavy database, you may be able to offload some of that read workload to a read-only replica, but such approaches will only get you so far. You're most likely to reach the need to truly scale out to potentially dozens or hundreds of machines or VMs with mixed, HTAP-type workloads across large data sets, which are more demanding in terms of both compute cores and memory.
- Scaling out can make database instances easier to manage through smaller indices, faster backup and restore, and faster maintenance operations—such as the ability to do parallel vacuuming.
- It can help reduce your "blast radius" when a database instance is temporarily unavailable.
- You have an architecture designed for horizontal scalability from the beginning. Two good examples are the use of microservices and the need to handle massive data volumes.

In the past, the typical way to scale out a relational database was through application-level sharding, which can be difficult to do and fragile to maintain. It requires writing code to break your data set into logical partitions, which reside in different databases on different machines, and then dealing with all the additional complexity of routing queries to the right logical partition. Keys and indexing become more complex, cross-partition operations become an issue, and hotspots—the uneven distribution of data and operations—come into play. After you optimize for all of this, when need to scale out more, you'll need to re-balance and re-tune everything as you repartition your data across additional nodes.

Today, PostgreSQL provides a much easier way to scale out. Citus, an open-source PostgreSQL extension, gives you the means to support even the most demanding PostgreSQL workload, regardless of whether it's driven by very large data sets or very heavy concurrency. It achieves this by intelligently and transparently distributing your data and transactions across multiple nodes to achieve massive parallelism—along with a much bigger compute, memory, and disk footprint. Citus has been used to scale out PostgreSQL to 2 petabytes, running on more than 80 individual nodes. To your application, however, it looks and acts like a single-node PostgreSQL database.

Reliability and availability

With a mission-critical application, performance and scalability aren't worth much if you can't keep everything up and running. And if you want an always-on application, you'll need to start with an always-on database. After all, if the database goes down, so does the entire app, no matter what mechanisms you may have employed to ensure a highly available application tier.

PostgreSQL has a well-earned reputation for its ability to deliver high reliability and availability. The engine itself has been battle-hardened over its long history, with a proven architecture that's broadly acknowledged for its reliability, data integrity, and correctness. High-availability features necessary for maintaining business continuity—including replication/failover and backup/restore—have been added over the years, have also had time to mature, and are known for their robustness.

If you deploy on-premises, you'll need to setup and maintain these high-availability features on your own, including deployment and management of replicas, maintaining backups, and so on. For the utmost availability without the additional complexity that such mechanisms add, you'll want to deploy to the cloud on fully managed PaaS services, in which case these capabilities become fully automated.

Physical replication

Replication—the process of streaming database transaction logs from a primary database instance to one or more standby instances—is a well-accepted means of achieving high availability. You can also run read-only queries against the standby, as a means of achieving limited read scale-out.

Streaming replication in PostgreSQL, which was introduced in 2010, works by streaming write-ahead log (WAL) records from the primary read/write node to the replica nodes as they're generated. By default, streaming replication in PostgreSQL is asynchronous, with a small delay between committing a transaction in the primary and the changes becoming visible in the standby. As a result, if the primary server crashes, there's the chance that some transactions that were committed may not have been replicated to the standby server, resulting in potential data loss.

Cascading replication, which is also asynchronous, allows a standby server to accept replication connections and stream WAL records to other standbys. It provides a way to make replication more scalable by enabling you to grow the overall number of replicas without increasing the load on the primary (read/write) server.

Synchronous replication in PostgreSQL builds on streaming replication, providing the ability to confirm that all changes made by a transaction on the primary have been transferred to one or more standby servers. It works by delaying the commit of a write transaction until confirmation is received that the commit has been written to the on-disk WAL of both the primary and all standbys, meaning that the only possibility for data loss is if the primary and the standby both crash at the same time. The main drawback of synchronous replication: it increases the response time of the requesting transaction, with the minimum wait time being the round-trip time between the primary and the standby.

When synchronous replication was first released, its use required waiting for all synchronous standbys to respond before a write transaction was committed. PostgreSQL 10, released in 2017, introduced quorum writes—or quorum commits—for synchronous replication, enabling developers and admins to control how it works in a more granular way by specifying that a transaction should be committed when M of N (for example, three out of five) synchronous standbys have responded.

Logical (transactional) replication

Also introduced in PostgreSQL 10, logical replication lets you send changes from one database to another at a higher level, much more selectively, which it does by using a publish-subscribe mechanism to support the replication of individual tables instead of entire databases. Changes on the publisher are sent to the subscriber as they occur, with the subscriber applying the data in the same order as the publisher to guarantee transactional consistency within a single subscription. Logical replication is sometimes referred to as transactional replication, and can be used for change data capture.

Typical use cases for logical replication might include facilitating migrations, integrations with other databases, consolidating several databases into one (e.g., for analytical purposes), replication between PostgreSQL instances on different platforms, sharing a subset of the database between multiple databases, or firing triggers for individual changes as they arrive on the subscriber.

Failover

If your high availability architecture includes one or more standby replicas, you'll need a way to move your workload to them if you lose your primary read/write replica. An easy way to achieve this is by using pg_auto_failover, an open-source PostgreSQL extension. It's easy to setup, supports automated failover, and includes software-driven decision making for when to implement failover in production scenarios. As good overview of it can be found here.

Other open-source tools can also be used to manage PostgreSQL failover. For example, repmgr can be used to monitor the replication process and support high-availability operations such as switch-overs and fail-overs. Patroni is another well-known cluster manager for PostgreSQL—albeit one that also mandates the use of several other open-source technologies such as Zookeeper and etcd.

Backup and point-in-time recovery

PostgreSQL also support three fundamental approaches for backup and restore: SQL dump, file-system level backup, and continuous archiving with point-in-time recovery. For mission-critical applications, you'll likely want to employ continuous archiving with point-in-time recovery (PITR), which combines periodic file-system level backups (which you can create by using pg_basebackup) with a continuous sequence of archived WAL files. If a system crashes, you can restore its database by loading the last backup onto another machine and then "replaying" the WAL entries that have been made since that backup was created. And you don't need to play back the WAL files all the way to the end; you can stop doing so at any time to achieve point-in-time recovery.

Backup and PITR are a great example of how all the above—from replication and failover through backups through PITR—are automated when you deploy to the cloud. In fact, of all the areas discussed in this paper, reliability and availability is the one area where deployment to the cloud can shield you from the most complexity.

Security

Regardless of what an enterprise application does, if it's mission-critical, then it needs to be secure—and common security best-practices call for granting each user only the permissions they need to do their jobs. PostgreSQL integrates with existing authentication mechanisms to provide fine-grained, role-based authorization and access control for both technical personnel and end-users, including the ability to limit access to data on a per-row or per-column basis. In addition, PostgreSQL supports strong encryption, enabling you to protect sensitive data both at-rest and in-transit.

Role-based permissions

PostgreSQL provides role-based access permissions, enabling it to integrate well with enterprise-level authentication solutions such as Active Directory and LDAP. A role can be either a single database user or a group of users, depending on how you set it up. Roles can own database objects (such as tables and functions) and can be used to assign privileges on those objects to other roles. You can also grant membership in one role to another role, thereby granting the member role those privileges assigned to the other role.

Access controls and auditing

Granular access controls in PostgreSQL let you tightly control who has access to what, beginning with fine-grained control over database (admin) level access. End users can also be granted granular access privileges to different tables and functions, or even to just a subset of the data within a single table. For example, through the use of column-level access in PostgreSQL, you could allow users who work in human resources—or a subset thereof, such as all HR managers—to view the salary column contained within a table while preventing all other users from doing so. Similarly, with its row-level access controls, you could enable only the doctor who creates new database records for a patient to have access to those records.

PostgreSQL also has you covered in terms of audit capabilities. For example, the open-source PostgreSQL Audit Extension (pgAudit) enables detailed session and/or object audit logging via the standard PostgreSQL logging facility, giving you a way to produce the audit logs required for certain government, financial, or ISO certifications.

Encryption

PostgreSQL also provides robust encryption capabilities, starting with the native support for using SSL connections to encrypt all traffic between applications and the database. The pgcrypto module in PostgreSQL allows certain fields to be stored in an encrypted format (i.e., encryption-at-rest), with support for hashing and cryptographic functions such as MD5, SHA, HMAC, AES, BLOWFISH, PGP, and CRYPT. It also supports DES and 3DES if your community version of PostgreSQL was compiled with OpenSSL support. When you use one of these encryption methods, the client supplies the decryption key and the data is decrypted on the server and then sent to the client.

Adaptability and extension framework

Clearly, everything we've discussed so far are must-haves for modern mission-critical applications. But what else might you need to cover all your requirements today? And what about tomorrow? Which database features and capabilities will be required to support your future needs? It's hard to know for sure. And a database that can't adapt to future needs has very limited value. Investments in mission-critical applications are just too large to justify a short shelf life.

First, PostgreSQL ships a new major release every year, with a track record of including broadly desired new features in each major release. Just as powerful, however, is its extension framework. It's an area where PostgreSQL shines, potentially more so than any other database.

PostgreSQL is highly extensible by design. This means that its system catalog—or data dictionary as it's often called—can be extended to add new functions, aggregates, data types, operators, operator classes for indices, and packages of related objects. PostgreSQL also supports dynamic loading, enabling user-written code to be provided as a shared library, loaded, and run. Code written in SQL is even easier to add. This unparalleled level of flexibility makes PostgreSQL an ideal platform for rapid prototyping of new applications and storage structures.

While the extensibility of PostgreSQL is a powerful asset on its own, it's how that capability is embraced by the open-source community that really makes it shine. If you need an extension that does something that's not built into PostgreSQL, the odds are high that you're not the first. More likely than not, someone else saw the same need and has already written the code, or at least a large part of it, and then shared it with the rest of the community to freely use and/or further modify. The PostgreSQL Extension Community, or PGXN as it's known to PostgreSQL developers, is a good starting point when looking for an extension that delivers certain functionality.

The powerful PostGIS extension is a great example—it's why PostgreSQL is the world's most widely used open-source geospatial database. Other popular extensions include pg_cron (for scheduling cron jobs on PostgreSQL), pg_trgm (for fuzzy string matching), and xml2 (for parsing and querying xml objects). Of the hundreds of PostgreSQL extensions that are available, several are referenced throughout this paper. However, there are many other useful extensions for you to explore.

Even if there's not an extension that already does what you need, if you start developing one and share it, you'll probably find others in the community who are willing to help you get it across the finish line. And that's the real beauty of PostgreSQL extensibility in the hands of the open-source community. As more developers discover a useful extension, more of them contribute to its evolution, help find and patch bugs, optimize its performance, and so on. Before long, everyone has access to robust new functionality that's been thoroughly vetted. Many of the innovative features that are built-into PostgreSQL today started as—or still exist as—extensions and have evolved in just this way, which serves as further evidence just how powerful and empowering the extensibility of PostgreSQL can be.

Developer productivity: Relational semantics, JSON, and more

Up until now, we've focused on what PostgreSQL delivers—that is, the key subset of features that make it suitable for mission-critical applications. But how do you put those capabilities to use? Does PostgreSQL empower developers with an efficient development model, with the right database semantics and other properties? And do your database developers have the right skills and tools to put PostgreSQL to use? The answers to both of these questions is yes.

Relational semantics and JSON data

Modern enterprise applications require a database that provides robust relational semantics. PostgreSQL more than meets this need, empowering developers with powerful relational semantics such as ACID, transactions, JOINs, Foreign Keys (for referential integrity), and support for ANSI SQL.

Even better, PostgreSQL combines these powerful relational semantics with the with the flexibility of JSON. Specifically, PostgreSQL lets you store nonrelational data as a JSONB data type (where B stands for Binary representation of JSON). This allows you to efficiently store semi-structured JSON documents and index the keys and nested values within those document objects, making it possible to work with both structured and semi-structured data in a highly performant manner.

Developer productivity – tooling and frameworks

PostgreSQL is easy to put to use, with a broad range of useful tooling and frameworks across all levels of the database development stack. Here's a quick breakdown:

- **Language bindings.** With PostgreSQL, you won't need to learn a new development language. Every major development language has bindings for PostgreSQL, including C, C#, Go, Ruby, Node.js, and Python.
- **Stored procedures.** You can embed business logic in the PostgreSQL database layer of your application using stored procedures—providing a powerful way to improve application performance by performing complex calculations entirely within the database. This is accomplished using PL/pgSQL, an easy-to-use, loadable procedural language that can be used to create functions and triggers, add control structures to the SQL language, and perform complex calculations. It inherits all user-defined types, functions, and operators, and can be defined to be trusted by the server. And it's similar to Oracle's PL/SQL, which helps facilitate migrations from Oracle to PostgreSQL.
- **Development frameworks.** PostgreSQL is supported in popular open-source development frameworks, which can help you further accelerate application development. Popular examples include Django for Python, Active Record for Ruby on Rails, Hibernate for Java, and .NET Core—the list is virtually endless.
- **Tools and IDEs.** PostgreSQL works with popular development tools and IDEs. If you prefer, you can use a text-based code editor like GNU Emacs or one of its derivatives. If you prefer a more modern, visual development experience, popular open-source tools like Visual Studio Code, Azure Data Studio, Eclipse, and pgAdmin all work with PostgreSQL.
- **Database drivers.** Existing applications can connect to PostgreSQL using a PostgreSQL-compatible ODBC, JDBC, or ADO.NET driver. Such drivers are broadly available, enabling PostgreSQL to "just work" with leading reporting, BI, and analytics tools like Business Objects,

Microsoft Power BI, and Tableau; common data integration and movement tools like Attunity, Informatica, and Spark ETL; and popular data science tools like Python, R, and SAS.

- **Foreign data wrappers (FDWs).** FDWs can be a good integration tool. They let you create foreign tables in a PostgreSQL database that serve as proxies for some other data source. When you query a foreign table, the FDW queries the external data source and returns the results as if they were coming from a table in your PostgreSQL database. PostgreSQL ships with postgres_fdw, which allows you to connect to other PostgreSQL databases. You can also define your own FDWs or choose from a broad range of existing, open-source wrappers (such as those you can find on PGXN) to connect to Oracle, MySQL, MongoDB, and so on.
- **Documentation.** PostgreSQL often receives accolades for its documentation, which is both comprehensive and easy to navigate.

Deployment and management: on-premises or any cloud

After you've built your mission-critical application, you'll need to deploy and manage it—all using well-defined, predictable, and repeatable process. Again, the popularity and maturity of PostgreSQL will give you virtually endless options and flexibility.

Deployment

PostgreSQL integrates with popular open-source deployment and DevOps frameworks. For example, you can containerize the PostgreSQL database engine along with any extensions you've used into Docker containers, orchestrate container behavior (such as what to do in case of failover) using Kubernetes, and automate the deployment and configuration of everything across various staging, build, test, and production infrastructures using tools such as Terraform, Puppet, or Chef.

Portability—the ability to run PostgreSQL wherever you want—is another of its big selling points. You can develop within a VM on a laptop, containerize that environment, and then deploy to an on-premises data center, in every location across a chain of grocery stores, or even to a ship at sea. You can also run PostgreSQL just as easily—likely even easier—on hosted VMs in the cloud, or on fully managed PostgreSQL-as-a-service environments.

Management and monitoring

The PostgreSQL interactive terminal (psql) is the most popular command-line tool. A terminal-based front-end to PostgreSQL, it lets you interactively submit queries and see the results. Inputs to psql can also come from a file or from command line arguments. In addition, psql provides a number of meta-commands and shell-like features to facilitate scripts and task automation.

If you don't deploy to a fully-managed cloud service, you'll need to handle patching for minor version updates on your own. Fortunately, the process is straightforward. Major version upgrades with new features are released annually; otherwise, throughout the year, only minor version updates (which are limited to bug fixes) are released. Minor version updates are always binary-compatible with the underlying major version, requiring only momentary downtime to overwrite the binaries.

Major version upgrades usually change the internal format of system tables and data files. They can be implemented with minimal downtime by using pg_upgrade. For even less downtime, you can use logical replication to stream data to an upgraded PostgreSQL server replica. When the upgraded instance is all caught-up, you can then simply fail-over to upgraded server.

PostgreSQL also includes built-in monitoring capabilities. For example, the pg_stat_statements module provides a tool for tracking execution statistics for all SQL statements executed by the server. For additional monitoring capabilities, almost all third-party monitoring tools have PostgreSQL drivers, including New Relic, Datadog, and SolarWinds—or you can "roll your own" monitoring environment using open-source solutions like Nagios or Grafana. All managed PostgreSQL services in the cloud provide their own built-in monitoring and metrics, which you can integrate with other solutions.

Access to skilled resources

We've covered the key technical requirements for mission-critical enterprise applications and how you can meet them with PostgreSQL. However, there's one more thing you'll need: an ecosystem of skilled resources. Can you find the developers needed to build your mission-critical applications—and the DBAs for managing them? And when you need external support, where will you turn? Just as important, how can you identify those resources who truly know and understand PostgreSQL, both in terms of technical expertise and familiarity with the PostgreSQL ecosystem at large?

Developers, development partners, and DBAs

As we've said, PostgreSQL has been around for more than 20 years. It's already the forth-most-popular database in the world according to DB-Engines. And it's rapidly gaining on the first three, having tripled in popularity over the past seven years while theirs has essentially remained flat. For a glimpse into the trajectory of PostgreSQL popularity and where it may be headed, extrapolate this chart over the next seven years—and make sure to note the logarithmic scale.

The point of all this: PostgreSQL is already broadly established, with an ecosystem that includes plenty of skilled developers, development partners, and DBAs. If you're unfamiliar with the open-source ecosystem, or are just beginning to explore it, this may not be immediately obvious. Its pervasiveness alone is the best evidence of the available talent pool, which is continuing to grow every day.

There's an even broader pool of skilled resources that can easily transition to PostgreSQL. For example, most developers who have worked on a relational database with SQL APIs can easily learn PostgreSQL. Similarly, DBAs who have worked with Oracle will find that PostgreSQL behaves and can be managed very similarly.

Support

Next, there's support. You'll need it… that's a given. But where will you get it, and when there's a problem, will you have a single entity to hold accountable? Again, this is an area where both the maturity and increasing popularity of PostgreSQL comes into play. PostgreSQL is supported by many companies today, including names like EnterpriseDB, 2ndQuadrant, Credativ, and Cybertec. They're well-known and well-respected within the PostgreSQL community—both in terms of their technical expertise and their contributions to the PostgreSQL project itself.

Of course, if you deploy to a managed PostgreSQL service in the cloud, you'll be able to rely on a simple service-level-agreement (SLA) from your chosen cloud provider to achieve the level of overall database availability you'll need.

Committers and contributors

In open-source-speak, *committers* are those people who review submitted code for PostgreSQL, "commit" it into the source code repository, and then work with other *contributors* to test, refine, and eventually incorporate it into the next official build. In short, they're the real experts on PostgreSQL.

Where you really want to look for committers and contributors is when selecting a support partner or cloud provider. You'll want to work with those companies whose technical staff are actively involved with maintaining PostgreSQL, or who are creating the extensions that power the PostgreSQL ecosystem.

Take cloud providers, for example. Is the cloud provider just providing a managed service on the PostgreSQL bits they've downloaded, or are they actively contributing to its improvement for everyone's benefit? Similarly, if a cloud provider is building on PostgreSQL to deliver incremental functionality but keeping all those efforts behind closed doors, what will your adoption of their service mean in terms of vendor lock-in, or the ability to draw on the open-source community at large when it comes to evolving and supporting those features?

Benefits of the cloud

As mentioned throughout this paper, building and running mission-critical applications on PostgreSQL is easier to do when you deploy to a fully-managed PostgreSQL service in the cloud. Need to scale up? Just select a larger VM. Need to scale out? Drag this slider. Want to setup streaming replication and create a read-replica of your database in another region? It takes just a few clicks.

In addition, depending on which cloud provider you choose, you'll have access to extensive capabilities that complement PostgreSQL to make it perform and scale even better (and easier), further increase availability and security, and further simplify deployment and management. For example, here's what you'll have access to when you use the Azure Database for PostgreSQL service from Microsoft. Some of the following capabilities are exposed through the service itself, while others are designed to seamlessly work with and complement it.

In terms of scalability and performance, Azure Database for PostgreSQL lets you:

- Scale out horizontally, without downtime. Azure Database for PostgreSQL Hyperscale (Citus) lets you scale out to hundreds of nodes by simply dragging a slider. Data is transparently redistributed in the background, with incoming queries automatically parallelized across all scale-out nodes.
- Easily scale up or down. Storage is provisioned and can be scaled up separately—or can be configured to grow automatically as needed.
- Create read-replicas to enhance performance in minutes.
- Easily optimize query performance through features such as Query Store, Query Performance Insight, and Performance Recommendations.

In terms of reliability and availability, Azure Database for PostgreSQL gives you

- Built-in high availability, with an industry leading financially backed 99.99-percent service-level agreement.
- Automatic full, differential, and transaction-log backups—and point-in-time restore.
- Turn-key replication and automated fail-over.

In terms of security, Azure Database for PostgreSQL:

- Makes it easy to implement and configure security best-practices such database firewall rules, using SSL to encrypt database connections, virtual private networking, and role-based access.
- Can be used with Azure Advanced Threat Protection to help detect suspicious database activities, potential vulnerabilities, and anomalous database access and query patterns.

In terms of extensibility, Azure Database for PostgreSQL includes:

- A curated catalog of popular third-party extensions, which are maintained and upgraded for you. We validate their functionality and reliability to make sure you'll get what you expect.
- Access to extensions authored and maintained by Microsoft, including pg_cron, postgresql-hll, TopN, and Citus for scaling out. All are open source and have been freely given to the PostgreSQL community.

In terms of developer tools and productivity, Azure Database for PostgreSQL:

- Helps speed-up development and test processes by letting you provision new database instances in minutes.
- Lets you easily build front-end apps that connect to your database—for example, you could use Azure App Services or Azure Kubernetes Service (AKS) to build a Python app for end-to-end testing.
- Makes it easy to use other Azure services—such as Azure Database Migration Service or Azure Stream Analytics—to capture streaming data and move it into your database. Similarly, you can use Azure Data Factory to move data out of your database and into another one, such as a data warehouse for advanced analytics.
- Gives you access to powerful productivity aids in Visual Studio Code, such as IntelliSense code completion and integrated debugging. Such capabilities go hand-in-hand with Azure Data Studio, a productivity tool for developers and DBAs. Both are open source, cross-platform, and include extensions for PostgreSQL.

In terms of deployment and management, PostgreSQL lets you:

- Provision a new PostgreSQL database instance in minutes—you can do it using the Azure Portal, the Azure CLI, or an Azure Resource Manager template. You can even create an Azure Database for PostgreSQL server and bind it to an Azure Kubernetes Service cluster—all with a few simple steps.
- Take advantage of Azure Arc Data Services to deploy, run, and manage Azure Database for PostgreSQL Hyperscale on-premises, at the edge, in multi-cloud environments, or any combination thereof.

In terms of skilled resources, Azure Database for PostgreSQL:

- Is a fully managed service, meaning you won't have to deal with day-to-day operational tasks such as deploying new hardware, performing backups, applying security patches, or monitoring replication.
- Is supported by a team with more top-level PostgreSQL committers than all other cloud providers combined.

Summary

PostgreSQL meets all the requirements for modern, mission-critical enterprise applications. It performs and scales well, both vertically and horizontally, and delivers everything needed for high availability and comprehensive security. What's more, it's easily and virtually endlessly extensible, works with a broad range of development tools and frameworks, and provides access to a broad ecosystem of skilled developers, DBAs, and support resources—not to mention an entire open-source community that's invested in and committed to its success.

Many of the things that make PostgreSQL a strong choice for mission-critical enterprise applications are easier to put to use—and work even better—when you deploy on Azure Database for PostgreSQL. It lets you deploy in minutes and scale horizontally with Hyperscale (Citus), with turnkey high-availability features like replication, backups, and point-in-time restore. Deployment and management tasks also get much easier, often taking just a few clicks or command-line statements.

When you deploy on Azure Database for PostgreSQL from Microsoft, you'll be running on the cloud platform that lets you deploy and run PostgreSQL anywhere. Just as important, you'll be supported by a team that maintains key open-source extensions for PostgreSQL and has more top-level PostgreSQL committers than any other cloud provider—people who choose to work at Microsoft because Microsoft recognizes the value of open-source software and is committed to investing in its success.

If you'd like to learn more about Azure Database for PostgreSQL, you can check it out here—or you can sign up for a free Azure account and get started with Azure Database for PostgreSQL today.

For more information

- Check out our documentation on Azure Database for PostgreSQL
- Review the latest product updates for Azure Database for PostgreSQL
- Read some popular blog articles on Azure Database for PostgreSQL:
    - Latest updates to Azure Database for PostgreSQL
    - Auditing: Know what's going on in your Postgres database
    - Database sharding explained in plain English
    - Azure Database for PostgreSQL - Hyperscale (Citus) now generally available
    - Introducing Postgres Projects in Azure Data Studio
    - Microsoft Azure Welcomes PostgreSQL Committers
    - PostgreSQL to Azure DB for PostgreSQL – Hyperscale (Citus) online migrations now generally available
    - Introducing pg_auto_failover: Open-source extension for automated failover and high-availability in PostgreSQL