Datenbankkonfiguration

Enterprise applications

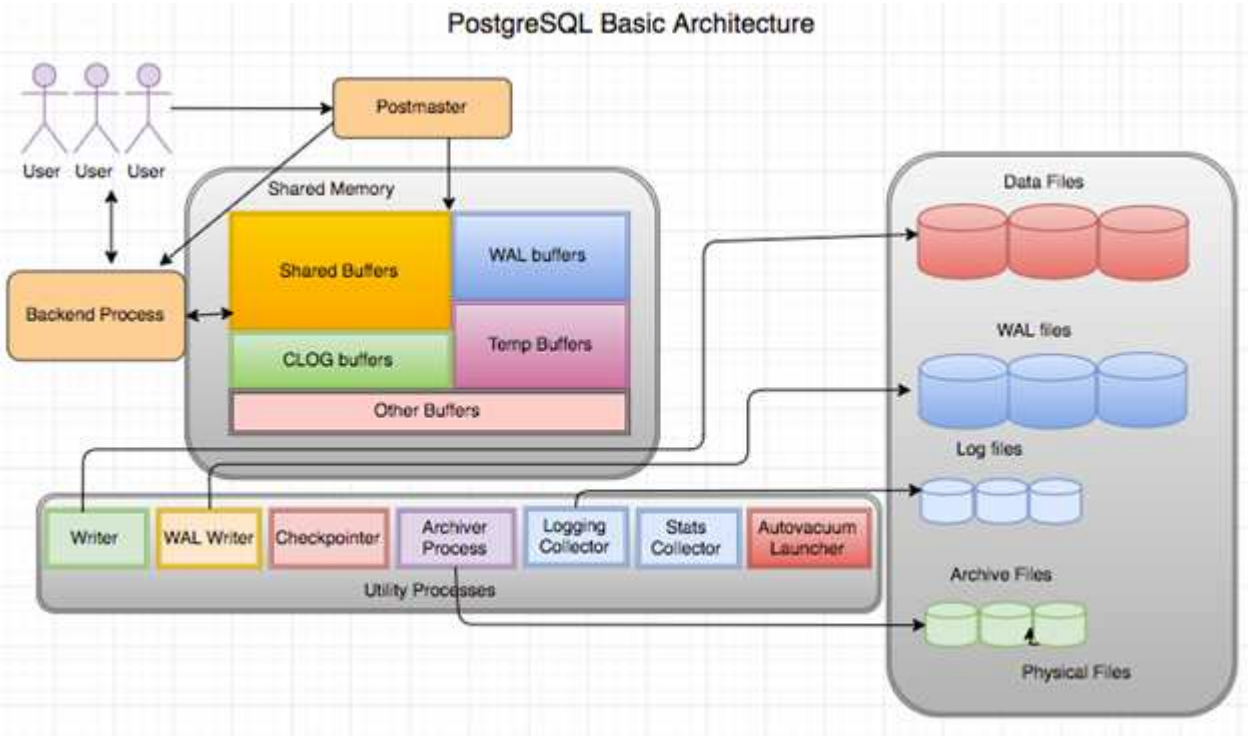
NetApp
February 10, 2026

Inhalt

Datenbankkonfiguration

Der Netapp Architektur Sind

PostgreSQL ist ein RDBMS, das auf Client- und Serverarchitektur basiert. Eine PostgreSQL-Instanz wird als Datenbank-Cluster bezeichnet, bei dem es sich um eine Sammlung von Datenbanken und nicht um eine Sammlung von Servern handelt.



In einer PostgreSQL-Datenbank gibt es drei Hauptelemente: Den Postmaster, das Frontend (Client) und das Backend Der Client sendet Anfragen an den Postmaster mit Informationen wie IP-Protokoll und zu welcher Datenbank eine Verbindung hergestellt werden soll. Der Postmaster authentifiziert die Verbindung und leitet sie zur weiteren Kommunikation an den Back-End-Prozess weiter. Der Back-End-Prozess führt die Abfrage aus und sendet Ergebnisse direkt an das Frontend (Client).

Eine PostgreSQL-Instanz basiert auf einem Multiprocess-Modell statt auf einem Multithread-Modell. Es gibt mehrere Prozesse für verschiedene Jobs, und jeder Prozess hat seine eigene Funktionalität. Zu den wichtigsten Prozessen gehören der Clientprozess, der WAL Writer-Prozess, der Background Writer-Prozess und der Checkpointer-Prozess:

- Wenn ein Client-Prozess (Vordergrund) Lese- oder Schreibanforderungen an die PostgreSQL-Instanz sendet, werden keine Daten direkt auf die Festplatte geschrieben oder gelesen. Zuerst werden die Daten in gemeinsam genutzten Puffern und WAL-Puffern (Write-Ahead Logging) gepuffert.
- Ein WAL-Schreibprozess manipuliert den Inhalt der gemeinsam genutzten Puffer und WAL-Puffer, um in die WAL-Protokolle zu schreiben. WAL-Protokolle sind in der Regel Transaktionsprotokolle von PostgreSQL und werden sequenziell geschrieben. Um die Reaktionszeit aus der Datenbank zu verbessern, schreibt PostgreSQL zunächst in die Transaktionsprotokolle und bestätigt den Client.
- Um die Datenbank in einen konsistenten Zustand zu versetzen, überprüft der Background Writer-Prozess den gemeinsam genutzten Puffer regelmäßig auf fehlerhafte Seiten. Anschließend überträgt es die Daten auf die Datendateien, die auf NetApp Volumes oder LUNs gespeichert sind.

- Der Checkpointer-Prozess läuft auch periodisch (seltener als der Hintergrundprozess) und verhindert jegliche Änderung der Puffer. Er signalisiert dem WAL Writer-Prozess, den Checkpoint-Datensatz zu schreiben und an das Ende der WAL-Protokolle zu löschen, die auf der NetApp-Festplatte gespeichert sind. Er signalisiert auch, dass der Background Writer-Prozess alle fehlerhaften Seiten auf die Festplatte schreibt und auf diese schreibt.

Initialisierungsparameter

Sie erstellen mithilfe von ein neues Datenbankcluster `initdb` Programm. An `initdb` Skript erstellt die Datendateien, Systemtabellen und Vorlagendatenbanken (template0 und template 1), die den Cluster definieren.

Die Vorlagendatenbank stellt eine Bestandsdatenbank dar. Es enthält Definitionen für Systemtabellen, Standardansichten, Funktionen und Datentypen. `pgdata` Fungiert als Argument für den `initdb` Skript, das den Speicherort des Datenbank-Clusters angibt.

Alle Datenbankobjekte in PostgreSQL werden intern von den jeweiligen OIDs verwaltet. Tabellen und Indizes werden auch von einzelnen OIDs verwaltet. Die Beziehungen zwischen Datenbankobjekten und ihren jeweiligen OIDs werden je nach Objekttyp in entsprechenden Systemkatalogtabellen gespeichert. OIDs von Datenbanken und Heap-Tabellen werden beispielsweise in gespeichert `pg_database` Und `pg_class. Sie können die OIDs durch Abfragen auf dem PostgreSQL-Client ermitteln.

Jede Datenbank verfügt über eigene einzelne Tabellen und Indexdateien, die auf 1 GB beschränkt sind. Jede Tabelle hat zwei zugehörige Dateien, die jeweils mit dem Suffix versehen sind `_fsm` Und `_vm`. Sie werden als freie Raumkarte und Sichtbarkeitskarte bezeichnet. Diese Dateien speichern die Informationen über die freie Speicherkapazität und haben Sichtbarkeit auf jeder Seite in der Tabellendatei. Indizes verfügen nur über individuelle freie Speicherplatzkarten und haben keine Sichtbarkeits-Karten.

Der `pg_xlog/pg_wal` Das Verzeichnis enthält die Write-Ahead-Protokolle. Mit Write-Ahead-Protokollen werden die Zuverlässigkeit und Performance der Datenbank verbessert. Immer wenn Sie eine Zeile in einer Tabelle aktualisieren, schreibt PostgreSQL die Änderung zuerst in das Write-Ahead-Protokoll und schreibt später die Änderungen auf die eigentlichen Datenseiten auf eine Festplatte. Der `pg_xlog` Das Verzeichnis enthält normalerweise mehrere Dateien, aber initdb erstellt nur die erste. Zusätzliche Dateien werden bei Bedarf hinzugefügt. Jede xlog-Datei ist 16 MB lang.

Einstellungen

Es gibt mehrere PostgreSQL-Tuning-Konfigurationen, die die Performance verbessern können.

Die am häufigsten verwendeten Parameter sind:

- `max_connections = <num>`: Die maximale Anzahl von Datenbankverbindungen, die gleichzeitig verfügbar sind. Verwenden Sie diesen Parameter, um den Austausch auf Festplatte zu beschränken und die Leistung zu unterbinden. Je nach Anwendungsanforderung können Sie diesen Parameter auch für die Einstellungen des Verbindungspools anpassen.
- `shared_buffers = <num>`: Die einfachste Methode zur Verbesserung der Leistung Ihres Datenbankservers. Die Standardeinstellung ist niedrig für die meisten modernen Hardware. Sie wird während der Bereitstellung auf ca. 25 % des verfügbaren RAM auf dem System eingestellt. Diese Parametereinstellung hängt davon ab, wie sie mit bestimmten Datenbankinstanzen funktioniert; Sie müssen die Werte möglicherweise durch Versuch und Fehler erhöhen oder verringern. Bei einer hohen

Einstellung wird die Performance jedoch wahrscheinlich beeinträchtigt.

- `effective_cache_size = <num>`: Dieser Wert teilt PostgreSQL's Optimizer mit, wie viel Speicher PostgreSQL für das Caching von Daten zur Verfügung hat und hilft bei der Bestimmung, ob ein Index verwendet werden soll. Ein größerer Wert erhöht die Wahrscheinlichkeit, einen Index zu verwenden. Dieser Parameter sollte auf die Größe des zugewiesenen Speichers eingestellt werden `shared_buffers` Und die Menge an verfügbarem BS-Cache. Dieser Wert liegt häufig bei mehr als 50 % des gesamten Systemspeichers.
- `work_mem = <num>`: Dieser Parameter steuert die Speichermenge, die in Sort-Operationen und Hash-Tabellen verwendet werden soll. Wenn Sie eine starke Sortierung in Ihrer Anwendung ausführen, müssen Sie möglicherweise den Speicherplatz erhöhen, aber seien Sie vorsichtig. Es handelt sich nicht um einen systemweiten Parameter, sondern um einen pro-Operation-Parameter. Wenn eine komplexe Abfrage mehrere Sortieroperationen enthält, verwendet sie mehrere Work_mem-Speichereinheiten, und mehrere Back-Ends könnten dies gleichzeitig tun. Diese Abfrage kann oft dazu führen, dass Ihr Datenbankserver ausgetauscht wird, wenn der Wert zu groß ist. Diese Option wurde zuvor in älteren PostgreSQL-Versionen als sort_mem bezeichnet.
- `fsync = <boolean> (on or off)`: Dieser Parameter legt fest, ob alle WAL-Seiten mit fsync() synchronisiert werden sollen, bevor eine Transaktion durchgeführt wird. Wenn Sie sie deaktivieren, kann die Schreibleistung manchmal verbessert werden, und wenn Sie sie einschalten, erhöht sich der Schutz vor dem Risiko von Beschädigungen, wenn das System abstürzt.
- `checkpoint_timeout`: Der Checkpoint-Prozess überträgt die Daten auf die Festplatte. Dies beinhaltet viele Lese-/Schreibvorgänge auf der Festplatte. Der Wert wird in Sekunden festgelegt, und niedrigere Werte verringern die Absturzwiederherstellungszeit, und höhere Werte können die Belastung der Systemressourcen verringern, indem die Checkpoint-Anrufe reduziert werden. Legen Sie je nach Wichtigkeit der Anwendung, Auslastung und Verfügbarkeit der Datenbank den Wert von Checkpoint_Timeout fest.
- `commit_delay = <num>` Und `commit_siblings = <num>`: Diese Optionen werden zusammen verwendet, um die Leistung zu verbessern, indem mehrere Transaktionen, die auf einmal begehen, ausgeschrieben werden. Wenn mehrere commit_Geschwister-Objekte aktiv sind, sobald Ihre Transaktion abgeschlossen ist, wartet der Server auf commit_delay Mikrosekunden, um zu versuchen, mehrere Transaktionen gleichzeitig zu begehen.
- `max_worker_processes / max_parallel_workers`: Konfigurieren Sie die optimale Anzahl von Arbeitern für Prozesse. Max_Parallel_Workers entspricht der Anzahl der verfügbaren CPUs. Je nach Anwendungsdesign erfordern Abfragen möglicherweise weniger Mitarbeiter für parallele Vorgänge. Es ist besser, den Wert für beide Parameter gleich zu halten, aber den Wert nach dem Testen anzupassen.
- `random_page_cost = <num>`: Dieser Wert steuert die Art und Weise, wie PostgreSQL nicht-sequentielle Datenträger liest. Ein höherer Wert bedeutet, dass PostgreSQL eher einen sequenziellen Scan anstelle eines Indexscans verwendet, was darauf hinweist, dass Ihr Server über schnelle Festplatten verfügt.Ändern Sie diese Einstellung, nachdem Sie andere Optionen wie planbasierte Optimierung, Staubsaugen, Indexieren auf Abfragen oder Schema überprüft haben.
- `effective_io_concurrency = <num>`: Dieser Parameter legt die Anzahl der gleichzeitigen Festplatten-I/O-Operationen fest, die PostgreSQL gleichzeitig auszuführen versucht. Wenn Sie diesen Wert erhöhen, erhöht sich die Anzahl der I/O-Vorgänge, die jede einzelne PostgreSQL-Sitzung parallel initiieren möchte. Der zulässige Bereich ist 1 bis 1,000 oder Null, um die Ausgabe asynchroner I/O-Anfragen zu deaktivieren. Derzeit wirkt sich diese Einstellung nur auf Bitmap-Heap-Scans aus. Solid State Drives (SSDs) und anderer speicherbasierter Storage (NVMe) können oft zahlreiche gleichzeitige Anforderungen verarbeiten, sodass der beste Nutzen aus Hunderten von Laufwerken zu ziehen ist.

Eine vollständige Liste der PostgreSQL-Konfigurationsparameter finden Sie in der PostgreSQL-Dokumentation.

TOAST

TOAST steht für die Oversized-Attribute Storage-Technik. PostgreSQL verwendet eine feste Seitengröße (üblicherweise 8 KB) und erlaubt nicht, dass Tupel mehrere Seiten umfassen. Daher ist es nicht möglich, große Feldwerte direkt zu speichern. Wenn Sie versuchen, eine Zeile zu speichern, die diese Größe überschreitet, bricht TOAST die Daten großer Spalten in kleinere „Stücke“ und speichert sie in einem TOAST Tisch.

Die großen Werte der getoasteten Attribute werden nur dann herausgezogen (wenn sie überhaupt ausgewählt sind), wenn der Ergebnissatz an den Client gesendet wird. Die Tabelle selbst ist viel kleiner und kann mehr Zeilen in den gemeinsam genutzten Puffer-Cache passen als ohne Out-of-Line Storage (TOAST).

VAKUUM

Im normalen PostgreSQL-Betrieb werden Tupel, die durch eine Aktualisierung gelöscht oder veraltet gemacht werden, nicht physisch aus ihrer Tabelle entfernt; sie bleiben vorhanden, bis VAKUUM ausgeführt wird. Daher müssen Sie regelmäßig VAKUUM betreiben, insbesondere auf häufig aktualisierten Tabellen. Der belegte Speicherplatz muss dann zur Wiederverwendung durch neue Zeilen zurückgewonnen werden, um einen Ausfall von Festplattenspeicher zu vermeiden. Er gibt jedoch nicht den Speicherplatz an das Betriebssystem zurück.

Der freie Platz innerhalb einer Seite ist nicht fragmentiert. VACUUM schreibt den gesamten Block neu, verpackt die restlichen Zeilen und hinterlässt einen einzigen zusammenhängenden Block freien Speicherplatz auf einer Seite.

Dagegen verdichtet VACUUM FULL Tabellen aktiv, indem eine völlig neue Version der Tabellendatei ohne Totraum geschrieben wird. Diese Aktion minimiert die Größe des Tisches, kann aber lange dauern. Außerdem wird zusätzlicher Speicherplatz für die neue Kopie der Tabelle benötigt, bis der Vorgang abgeschlossen ist. Ziel des routinemäßigen VAKUUMS ist es, die VOLLE VAKUUMAKTIVITÄT zu vermeiden. Bei diesem Prozess werden nicht nur Tabellen auf der Mindestgröße gespeichert, sondern auch der Festplattenspeicherplatz weiterhin gleichmäßig genutzt.

Tablespaces

Bei der Initialisierung des Datenbank-Clusters werden automatisch zwei Tablespaces erstellt.

Der `pg_global` Tablespace wird für freigegebene Systemkataloge verwendet. Der `pg_default` Tablespace ist der Standard-Tablespace der Datenbanken temple1 und template0. Wenn die Partition oder das Volume, auf der das Cluster initialisiert wurde, nicht mehr genügend Speicherplatz hat und nicht erweitert werden kann, kann ein Tablespace auf einer anderen Partition erstellt und verwendet werden, bis das System neu konfiguriert werden kann.

Ein stark genutzter Index kann auf einer schnellen, hochverfügbaren Festplatte wie einem Solid-State-Gerät platziert werden. Darüber hinaus kann eine Tabelle mit archivierten Daten, die selten verwendet oder nicht Performance-kritisch sind, auf einem kostengünstigeren, langsameren Festplattensystem wie SAS- oder SATA-Laufwerken gespeichert werden.

Tablespaces sind Bestandteil des Datenbank-Clusters und können nicht als eigenständige Erfassung von Datendateien behandelt werden. Sie sind von Metadaten im Hauptdatenverzeichnis abhängig und können daher nicht an einen anderen Datenbankcluster angeschlossen oder einzeln gesichert werden. Wenn Sie einen Tablespace verlieren (durch Dateilöschung, Festplattenfehler usw.), kann der Datenbankcluster möglicherweise unlesbar werden oder nicht mehr starten. Wenn ein Tablespace auf einem temporären

Dateisystem wie einer RAM-Festplatte platziert wird, besteht die Gefahr, dass der gesamte Cluster zuverlässig ist.

Nach der Erstellung kann ein Tablespace aus jeder beliebigen Datenbank verwendet werden, wenn der anfordernde Benutzer über ausreichende Berechtigungen verfügt. PostgreSQL verwendet symbolische Links, um die Implementierung von Tablespaces zu vereinfachen. PostgreSQL fügt dem eine Zeile hinzu `pg_tablespace` Tabelle (eine clusterwide-Tabelle) und weist dieser Zeile eine neue Objektkennung (OID) zu. Schließlich verwendet der Server die OID, um einen symbolischen Link zwischen Ihrem Cluster und dem angegebenen Verzeichnis zu erstellen. Das Verzeichnis `$PGDATA/pg_tblspc` Enthält symbolische Links, die auf jeden nicht integrierten Tablespace verweisen, der im Cluster definiert ist.