Abstract

PostgreSQL comes with variants that include PostgreSQL, PostgreSQL Plus, and EDB Postgres Advanced Server (EPAS). PostgreSQL is typically deployed as the back-end database for multitier applications. It is supported by common middleware packages (such as PHP, Java, Python, Tcl/Tk, ODBC, and JDBC) and has historically been a popular choice for open-source database management systems. This document covers the configuration requirements and guidance on tuning and storage configuration for deploying PostgreSQL on NetApp® ONTAP® data management software.

To determine whether the environment, configurations, and versions specified in this guide can support your environment, see the Interoperability Matrix Tool (IMT).
# TABLE OF CONTENTS

1 Introduction ........................................................................................................................................... 3

2 PostgreSQL Architecture ..................................................................................................................... 3

3 Storage Design Considerations for PostgreSQL .............................................................................. 4
   3.1 Storage Virtual Machines .................................................................................................................. 4
   3.2 Logical Interface ...................................................................................................................................... 4
   3.3 Volume Layout ................................................................................................................................................ 6
   3.4 Storage Efficiencies ....................................................................................................................................... 8
   3.5 Data Protection ............................................................................................................................................... 9

4 PostgreSQL Database Best Practices .............................................................................................. 10
   4.1 Tablespace ................................................................................................................................................... 11
   4.2 File System Options ...................................................................................................................................... 11
   4.3 PostgreSQL Performance Best Practices ..................................................................................................... 11
   4.4 Database Configuration Best Practices ........................................................................................................ 12

5 Best Practices Summary ................................................................................................................... 13

Appendix A: Host Configuration ............................................................................................................. 15
   SELinux ................................................................................................................................................................ 15

Appendix B: PostgreSQL Page and File System Layout ...................................................................... 16
   TOAST ................................................................................................................................................................. 16
   VACUUM .............................................................................................................................................................. 16
   PostgreSQL File System Layout........................................................................................................................... 17

Appendix C: Storage Efficiency .............................................................................................................. 17

LIST OF FIGURES
   Figure 1) Basic architecture of PostgreSQL. .................................................................................................................. 3
   Figure 2) Dedicated storage layout for PostgreSQL database. ..................................................................................... 6
   Figure 3) Shared storage layout. .................................................................................................................................... 7
   Figure 4) Virtualized database layouts with shared storage. ....................................................................................... 7
   Figure 5) Storage efficiency flowchart. ........................................................................................................................ 8
   Figure 6) User-defined tablespaces stored on multiple disks (NFS, SAN). ................................................................. 12
   Figure 7) Data storage in PostgreSQL. ................................................................................................................... 16
   Figure 8) Directory structure ........................................................................................................................................ 17
   Figure 9) Compression and compaction...................................................................................................................... 18
1 Introduction

As data grows exponentially, data management becomes more complex for enterprises like yours. This complexity increases your licensing, operational, support, and maintenance costs. To reduce the overall TCO, you might consider switching from commercial to open-source databases with reliable, high-performing back-end storage. PostgreSQL is an advanced open-source database that is widely used in academic, commercial, and large enterprises. It comes with a new set of features that addresses the gaps in other relational database management systems (RDBMSs). NetApp® ONTAP® proprietary software is used in storage disk arrays such as NetApp FAS and NetApp AFF, NetApp ONTAP Select, and NetApp Cloud Volumes ONTAP. With the release of ONTAP 9, you can unify data management across flash, disk, and cloud. This unification enables you to simplify your storage environment and build the foundation for a Data Fabric that makes it easy to move your data anywhere.

Note: This document is intended for customers who use PostgreSQL databases in both the physical and virtualized environments. It contains information about best practices for successfully deploying and managing PostgreSQL databases on ONTAP 9.x. The recommendations in this report are generic, so they are not specific to any configuration. Depending on your business needs, some suggestions might require changes. You must evaluate your environment against the official documentation for PostgreSQL, hypervisors, OS, and ONTAP storage.

2 PostgreSQL Architecture

PostgreSQL is an RDBMS based on client and server architecture. A PostgreSQL instance is known as a database cluster, which is a collection of databases as opposed to a collection of servers. Figure 1 shows a brief overview of the PostgreSQL architecture.

![PostgreSQL Basic Architecture](image)

There are three major elements in a PostgreSQL database: the postmaster, the front end (client), and the back end. The client sends requests to the postmaster with information such as IP protocol and which database to connect to. The postmaster authenticates the connection and passes it to the back-end...
process for further communication. The back-end process executes the query and sends results directly to the front end (client).

A PostgreSQL instance is based on a multiprocess model instead of a multithreaded model. It spawns multiple processes for different jobs, and each process has its own functionality. The major processes include the client process, the WAL writer process, the background writer process, and the checkpointer process:

- When a client (foreground) process sends read or write requests to the PostgreSQL instance, it doesn't read or write data directly to the disk. It first buffers the data in shared buffers and write-ahead logging (WAL) buffers.
- A WAL writer process manipulates the content of the shared buffers and WAL buffers to write into the WAL logs. WAL logs are typically transaction logs of PostgreSQL and are sequentially written. Therefore, to improve the response time from the database, PostgreSQL first writes into the transaction logs and acknowledges the client.
- To put the database in a consistent state, the background writer process checks the shared buffer periodically for dirty pages. It then flushes the data onto the data files that are stored on NetApp volumes or LUNs.
- The checkpointer process also runs periodically (less frequently than the background process) and prevents any modification to the buffers. It signals to the WAL writer process to write and flush the checkpoint record to the end of WAL logs that are stored on the NetApp disk. It also signals the background writer process to write and flush all dirty pages to the disk.

3 Storage Design Considerations for PostgreSQL

This section addresses key topics of NetApp ONTAP storage design such as storage virtual machines (SVMs), logical interface (LIF) design, volume layouts, storage efficiencies, and data protection for deploying a PostgreSQL database. For more information about ONTAP deployments and configurations, see the ONTAP Administration guide.

3.1 Storage Virtual Machines

SVMs contain data volumes and one or more LIFs through which they serve data to the clients. A cluster must have at least one SVM to serve data. SVMs securely isolate the shared virtualized data storage and network, and each SVM appears as a single dedicated server to the clients.

In a multitenant PostgreSQL environment, each tenant's data (database instance) can be given a dedicated SVM on ONTAP storage. NetApp recommends a limit of approximately 125 ONTAP SVMs per cluster node, but usually the LIF maximums are reached before the SVM limit is reached.

You can also achieve multitenancy to some extent by having dedicated volumes from the same SVM but isolating them by network segments (LIFs, ports).

3.2 Logical Interface

When you create LIFs for a PostgreSQL database, consider the following questions:

- **Performance**: Is the network bandwidth sufficient?
- **Resiliency**: Are there any single points of failure in the design?
- **Manageability**: Can the network be scaled nondisruptively?

**NFS LIF Design**

Although the NFS protocol has limited ability to define multiple paths to data, there are a few guidelines that help the database achieve performance and resiliency over an NFS protocol.
Binding a LIF to a physical port results in more granular control over the network configuration, because a given IP address on an ONTAP system is associated with only one network port at a time. Resiliency is then accomplished through the configuration of failover groups and failover policies. Enabling Link Aggregation Control Protocol (LACP) and IP load balancing are the most common approaches for NFS databases running in larger environments. However, when there are fewer NFS clients, enabling LACP and IP load balancing on the ONTAP interface groups doesn’t seem to be efficient.

**LIF Failover**

The behavior of LIFs during network disruption is controlled by failover policies and failover groups. ONTAP enables management of LIF failover based on broadcast domains. Therefore, you can define all the ports that have access to a given subnet and enable ONTAP to select an appropriate failover LIF. This approach has limitations in a high-speed database storage network environment because of the lack of predictability.

For example, an environment can include both 1Gb ports for routine file system access and 10Gb ports for data file I/O. If both types of ports exist in the same broadcast domain, LIF failover can result in moving data file I/O from a 10Gb port to a 1Gb port. Therefore, NetApp recommends choosing individual ports for LIF failover.

**Key Recommendations for LIF Failover**

- Configure a failover group as user defined.
- Populate the failover group with ports on the storage failover (SFO) partner controller. The LIFs then follow the aggregates during a storage failover, which means you avoid creating indirect traffic.
- Use failover ports with performance characteristics that match those of the original LIF. For example, a LIF on a single physical 10Gb port should include a failover group with a single 10Gb port. A four-port LACP LIF should fail over to another four-port LACP LIF. These ports would be a subset of the ports defined in the broadcast domain.

For more information about failover policies and failover groups, see the [ONTAP Network Management Guide](#).

**SAN LIF Design**

All modern SAN implementations allow clients to use multipathing for accessing data over multiple network paths and to select the optimal paths for access. As a result, LIF design is simpler to address because SAN clients automatically load-balance I/O across the best available paths. If a path becomes unavailable, the client automatically selects a different path. The resulting simplicity of design makes SAN LIFs more manageable.

You do not have to migrate a LIF in a SAN environment when volumes are relocated. After the volume move has completed, ONTAP sends a notification to the SAN about a change in paths, and the SAN clients automatically reoptimize.

**Key Recommendations**

- Do not create more paths than you need. Too many paths make overall management more complicated and can cause problems with path failover on some hosts.
- For a multipathing environment, it is better to have a NetApp LUN for the host zoned from different switches for high availability.
- In Data ONTAP 8.3 and later ONTAP versions, the selective LUN-mapping data is the default. With SLM, any new LUN is automatically advertised from the node that owns the underlying aggregate and the node’s high availability (HA) partner. This arrangement removes the need to create port sets or configure zoning to limit port accessibility.
• It is better to avoid indirect traffic in an I/O-intensive database environment where every microsecond of latency is critical but the visible performance effect is negligible for typical workloads.
• Any FC zone should never contain more than one initiator. Such an arrangement might appear to work at first, but cross talk between initiators eventually interferes with performance and stability.
• LUN mobility events (such as volume or LUN moves that involve moving a LUN from one HA pair in the cluster to another) should include a step to confirm that the LUN is being presented. This step can be done using the destination storage controllers before the mobility event is initiated.
• Modify the Selective LUN Map reporting-nodes list before moving.

3.3 Volume Layout

A well-designed volume layout ensures great database performance and easy management of the database. You should consider multiple factors when defining the volume layout for the database, such as the size of the database, the growth rate of the database, and the backup frequency.

Dedicated Storage Layout
• In a mission-critical environment, NetApp recommends keeping data files and WAL logs on dedicated volumes or LUNs for better performance in both the SAN and NFS layouts.
• Isolation of data files and WAL log volumes also enables faster recovery of the database.
• Multiple tablespaces of a PostgreSQL database can also be spread across dedicated volumes to get the individual database performance.
• If an aggregate is busy serving other applications in an ONTAP cluster, you should isolate PostgreSQL database on a dedicated aggregate of a node that isn’t busy (Figure 2).

Figure 2) Dedicated storage layout for PostgreSQL database.

Shared Storage Layout
• For a noncritical environment, sharing multiple databases on the same volume or LUN helps to reduce the operational and management cost (Figure 3).
• Hosting data and WAL logs of multiple databases on a single LUN or volume optimizes the NetApp Snapshot™ consolidation process by creating a single Snapshot copy for the entire dataset.
  
  Note: This layout might not be ideal for mission-critical systems.
Virtualized Layout

When considering virtualization for your databases, you should base storage decisions on business needs. You can provision storage to a virtual machine (VM) in three ways:

- iSCSI LUNs managed by the iSCSI initiator of the VM or NFS file systems mounted directly on a guest VM
- FC raw device mappings (RDMs)
- ONTAP volumes or LUNs managed by a hypervisor (VMware) as NFS and VMFS datastores
Key Recommendations

- When a VM owns its file systems, it is easier for you to identify the source of the file systems for their data.
- There can be a positive performance effect from channeling all I/Os through a hypervisor datastore.
- Choosing datastores might improve the overall manageability and operational overhead, but it adds complexity because of the extra set of timeouts, parameters, log files, and potential bugs.
- Choosing NFS over VMware Virtual Machine File System (VMFS) datastores helps in using the NetApp SnapRestore® technology in a disaster scenario.

3.4 Storage Efficiencies

Storage efficiency enables you to store the maximum amount of data in the smallest possible space at the lowest possible cost. However, enabling some of these technologies might affect the databases. The flowchart in Figure 5 shows the considerations for storage efficiency.

Figure 5) Storage efficiency flowchart.

Key Recommendations

- If there is no more than a single tenant sharing the storage dynamic disk pool (aggregate), it is a good practice to have thin provisioning enabled across volumes. In contrast, in a multitenant environment where tenants are sharing a storage pool (aggregate), there is a higher chance of running out of space because of uncertainty in the tenant’s usage pattern. Therefore, NetApp recommends that you use thick provisioning in multitenant environments.
- If a volume is thick provisioned, you must completely disable all efficiency features for that volume, including compression and removal of deduplication, by using the `sis undo` command. The volume should not appear in the `volume efficiency show` output. If it does, the volume is still partially configured for efficiency features and there is a chance you might run out of space, resulting in database I/O errors.
- In a virtualized layout, use either thin or thick lazy zeroed provisioning for datastores.
• For PostgreSQL workloads, NetApp recommends disabling deduplication for data and WAL log volumes. Enabling deduplication, compression, or both for pg_dump or tar file system backup volumes might save space; however, it depends on the customer environment.

• **LUN alignment** refers to optimizing I/O for the underlying file system layout. On a NetApp system, storage is organized in 4KB units. Align an 8KB block on a PostgreSQL data file to exactly two 4KB blocks. If an error in LUN configuration shifts the alignment by 1KB in either direction, each 8KB postgresql block would exist on three different 4KB storage blocks rather than two. This arrangement would cause increased latency and cause additional I/O to be performed within the storage system.

### 3.5 Data Protection

One of the major aspects of storage design is enabling protection for PostgreSQL volumes. Customers can protect their PostgreSQL databases either by using the dump approach or by using file system backups. This section explains the different approaches of backing up individual databases or the entire cluster.

**Native Data Protection**

There are three approaches to backing up PostgreSQL data:

• SQL Server dump
• File-system-level backup
• Continuous archiving

The idea behind the SQL Server dump method is to generate a file with SQL Server commands that, when returned to the server, can re-create the database as it was at the time of the dump. PostgreSQL provides the utility programs pg_dump and pg_dump_all for creating individual and cluster-level backup. These dumps are logical and do not contain enough information to be used by WAL replay.

An alternative backup strategy is to use file-system-level backup, in which administrators directly copy the files that PostgreSQL uses to store the data in the database. This method is done in offline mode: the database or cluster must be shut down. Another alternative is to use pg_basebackup and pg_stop to run hot streaming backup of the PostgreSQL database.

**Snapshot Technology**

NetApp has developed a full portfolio of technologies and tools to help organizations build or adapt their backup and recovery plans to respond to all business demands. With the NetApp Snapshot technology that is included in ONTAP software, you can create backups or execute restore operations for datasets of any size in a matter of seconds. For PostgreSQL, you must enable Snapshots for data files, WAL files, and archived WAL files to provide full or point-in-time recovery.

For PostgreSQL databases, the average backup time with Snapshot copies is in the range of a few seconds to a few minutes. This backup speed is 60 to 100 times faster than pg_basebackup and other file-system-based backup approaches.

Snapshot copies on NetApp storage can be both crash-consistent and application-consistent. A crash-consistent Snapshot copy is created on storage without quiescing the database, whereas an application-consistent Snapshot copy is created while the database is in backup mode. NetApp also ensures that subsequent Snapshot copies are incremental-forever backups to promote storage savings and network efficiency.

Because Snapshot copies are rapid and do not affect system performance, you can schedule multiple Snapshot copies daily instead of creating a single daily backup as with other streaming backup technology. When a restore and recovery operation is necessary, the system downtime is reduced by two key features:
• NetApp SnapRestore data recovery technology means that the restore operation is executed in seconds.
• Aggressive recovery point objectives (RPOs) mean that fewer database logs must be applied and forward recovery is also accelerated.

For backing up PostgreSQL, you must ensure that the data volumes are copied simultaneously with (consistency-group) WAL and the archived logs. While you are using Snapshot technology to copy WAL files, make sure that you run `pg_stop` to flush all the WAL entries that must be archived. If you flush the WAL entries during the restore, then you only need to stop the database, unmount, or delete the existing data directory and perform a SnapRestore operation on storage. After the restore is done, you can mount the system and bring it back to its current state. For point-in-time recovery, you can also restore WAL and archive logs; then PostgreSQL decides the most consistent point and recovers it automatically.

**Storage Replication**

Storage replication is suitable for low to medium recovery time objective (RTO) requirements. The Snapshot copies on primary storage can be replicated to secondary or tertiary storage—that is, a NetApp disaster recovery and backup destination (NetApp SnapMirror® and SnapVault® technology). This method works for a quick application-transparent failover and long-term retention of backups and clones.

SnapMirror data replication software provides synchronous and asynchronous replication. The replication is configured on the storage **volume** level, whereas NetApp MetroCluster™ high-availability and disaster recovery provide synchronous replication that works on the storage **system** level. MetroCluster integrates with SnapMirror to support distance and SLA requirements for applications, and delivers zero data loss (RPO) and near-zero failover time (RTO under 120 seconds) for a distance up to 300km.

SnapMirror offers capabilities that are cost efficient compared with traditional offerings. For example, if you want to protect thousands of PostgreSQL databases for a quick failover along with other workloads, SnapMirror offers better efficiency and TCO than individual application failover techniques.

NetApp disaster recovery backup storage can run either on premises, in the cloud, or in colocated data centers, enabling enterprises to easily move data anywhere in the Data Fabric with high transparency between copy infrastructures.

**Data Protection Software**

NetApp SnapCenter® plugin for PostgreSQL database, combined with Snapshot and NetApp FlexClone® technologies, offer you benefits such as:

• Rapid backup and restore.
• Space-efficient clones.
• The ability to build a speedy and effective disaster recovery system.
• A premium bundle that does not require separate licenses. Licenses are covered by the NetApp SnapManager® suite license.

**Note:** You might prefer to choose NetApp’s premium backup partners such as Veeam Software and Commvault under the following circumstances:

− Managing workloads across a heterogenous environment
− Storing backups to either cloud or tape for long-term retention
− Support for a wide range of OS versions and types

**4 PostgreSQL Database Best Practices**

This section covers more best practices for using the PostgreSQL database.
4.1 Tablespace

- Two tablespaces are automatically created when the database cluster is initialized. The \textit{pg\_global} tablespace is used for shared system catalogs. The \textit{pg\_default} tablespace is the default tablespace of the template1 and template0 databases. If the partition or volume on which the cluster was initialized runs out of space and cannot be extended, a tablespace can be created on a different partition and used until the system can be reconfigured.

- An index that is heavily used can be placed on a fast, highly available disk, like a solid-state device. Also, a table storing archived data that is rarely used or not performance critical can be stored on a less expensive, slower disk system like SAS or SATA drives.

- Tablespaces are a part of the database cluster and cannot be treated as an autonomous collection of data files. They depend on metadata contained in the main data directory, and therefore cannot be attached to a different database cluster or backed up individually. Similarly, if you lose a tablespace (through file deletion, disk failure, and so on), the database cluster might become unreadable or unable to start. Placing a tablescape on a temporary file system like a RAM disk risks the reliability of the entire cluster.

- After it is created, a tablespace can be used from any database if the requesting user has sufficient privileges. PostgreSQL uses symbolic links to simplify the implementation of tablespaces. PostgreSQL adds a row to the \textit{pg\_tablespace} table (a clusterwide table) and assigns a new object identifier (OID) to that row. Finally, the server uses the OID to create a symbolic link between your cluster and the given directory. The directory \$PGDATA/pg\_tblspc contains symbolic links that point to each of the non-built-in tablespaces defined in the cluster.

- For better performance, NetApp recommends keeping a maximum of one tablespace per logical file system. Also, you cannot control the location of individual files in a logical file system. However, PostgreSQL does not enforce any such limitation, and it is not directly aware of the file system boundaries on your system. It just stores files in the directories you tell it to use.

4.2 File System Options

- NetApp recommends using version 4 of the NFS protocol, because it has an integrated locking feature that version 3 lacks. Also, it has new features such as statefulness, improved security, and strong authentication.

- PostgreSQL does not recommend soft mounting of an NFS file system.

- PostgreSQL also advises mounting the NFS file system synchronously to avoid a data inconsistency problem. See the PostgreSQL documentation for more details.

- Use the following mount option while mounting an NFS file system:

\begin{verbatim}
nfs4 rw, hard,nointr,bg,vers=4,sync,proto=tcp,noatime,rsize=65536,wsize=65536
\end{verbatim}

- NetApp recommends having four to eight LUNs to support data file I/O.

- A NetApp LUN stores data in 4KB physical blocks. If you don’t set up the same block size, I/O is not aligned with physical blocks correctly and might end up writing to two different disks in the RAID group. Because writing to two disks results in latency, NetApp recommends using 4KB block sizes to create the partition on discovered LUN.

4.3 PostgreSQL Performance Best Practices

Designing an efficient and high-performing database file system layout is important from both a performance and scalability perspective, but database size is not a factor. A mistaken perception is that huge databases need high-performance disk architecture. Even if the database size is 50GB, you need a good disk architecture.

Here are some tips for designing the database layout:

- Ensure that the database has multiple tablespaces, with tables and indexes grouped according to the transaction rates.
• The tablespace must be placed across multiple disk file systems for balanced I/O. To make this balance possible, use logical volume manager (LVM) or place critical tables on different volumes or LUNs. (The volumes or LUNs can be on the same aggregate or different aggregates). Placing tables in this way ensures that multiple CPUs come into play to perform transactions across multiple disks as described in Figure 6.

Figure 6) User-defined tablespaces stored on multiple disks (NFS, SAN).

• Consider placing the `pg_xlog` or `pg_wal` directory on a separate disk (volume or LUN) on a high-transaction database.

• Use I/O monitoring tools to understand the I/O statistics across all the disks and manage the database objects accordingly.

4.4 Database Configuration Best Practices

There are several PostgreSQL tuning configurations that can improve performance. The most commonly used parameters are as follows:

• `max_connections = <num>`: The maximum number of database connections to have at one time. Use this parameter to restrict swapping to disk and killing the performance. Depending on your application requirement, you can also tune this parameter for the connection pool settings.

• `shared_buffers = <num>`: The simplest method for improving the performance of your database server. The default is low for most modern hardware. It is set during deployment to approximately 25% of available RAM on the system. This parameter setting varies depending on how it works with particular database instances; you might have to increase and decrease the values by trial and error. However, setting it high is likely to degrade performance.

• `effective_cache_size = <num>`: This value tells PostgreSQL's optimizer how much memory PostgreSQL has available for caching data and helps in determining whether to use an index. A larger value increases the likelihood of using an index. This parameter should be set to the amount of memory allocated to `shared_buffers` plus the amount of OS cache available. Often this value is more than 50% of the total system memory.

• `work_mem = <num>`: This parameter controls the amount of memory to be used in sort operations and hash tables. If you do heavy sorting in your application, you might need to increase the amount of memory, but be cautious. It isn’t a systemwide parameter, but a per-operation one. If a complex query has several sort operations in it, it uses multiple `work_mem` units of memory, and multiple back ends could be doing this simultaneously. This query can often lead your database server to swap if the value is too large. This option was previously called `sort_mem` in older versions of PostgreSQL.

• `max_fsm_pages = <num>`: This parameter helps control the free space map. When something is deleted from a table, it isn’t removed from the disk immediately; it is marked as “free” in the free space map. The space can then be reused for any new `INSERT` operations that you do on the table.
If your setup has a high rate of `DELETE` and `INSERT` operations, you might need to increase this value to avoid table bloat.

- `fsync = <boolean>`: This parameter determines whether all your WAL pages should be synchronized to disk by using `fsync()` before a transaction is committed. Turning it off can sometimes improve write performance.

- `commit_delay = <num>` and `commit_siblings = <num>`: These options are used together to help improve performance by writing out multiple transactions that are committing at once. If there are several `commit_siblings` objects active at the instant your transaction is committing, the server waits for `commit_delay` microseconds to try to commit multiple transactions at once.

- `random_page_cost = <num>`: This value controls the way PostgreSQL views nonsequential disk reads. A higher value means PostgreSQL is more likely to use a sequential scan instead of an index scan, indicating that your server has fast disks.

- `effective_io_concurrency = <num>`: This parameter sets the number of concurrent disk I/O operations that PostgreSQL attempts to execute simultaneously. Raising this value increases the number of I/O operations that any individual PostgreSQL session attempts to initiate in parallel. The allowed range is 1 to 1,000, or zero to disable issuance of asynchronous I/O requests. Currently, this setting only affects bitmap heap scans. Solid-state drives (SSDs) and other memory-based storage (NVMe) can often process many concurrent requests, so the best value can be in the hundreds.

See the PostgreSQL documentation for a complete list of PostgreSQL configuration parameters.

### 5 Best Practices Summary

The following are the best practices and recommendations for deploying PostgreSQL on ONTAP data management software.
Best Practices

- In a multitenant PostgreSQL environment, each tenant's data (database or instance) can have a dedicated SVM on ONTAP storage.
- Enabling LACP and IP load balancing is the most common approach during NFS LIF design. Otherwise, failover policies and groups should be configured in the regular method.
- For a SAN multipathing environment, it is better to have a NetApp LUN host zoned from different switches to ensure high availability.
- Any FC zone should never contain more than one initiator. Crosstalk between initiators eventually interferes with performance and stability.
- During LUN or volume moves in a HA cluster, the LUN must be presented using the destination storage controllers before the mobility event is initiated. Modify the selective LUN map reporting-nodes list before moving.
- In a mission critical environment, NetApp recommends keeping data files and WAL logs on dedicated volumes or LUNs for better performance. This applies to both the SAN and NFS layouts.
- Isolation of data files and WAL logs in a dedicated volume also enables faster recovery of the database.
- For better database performance, isolate the critical tablespaces on a dedicated volume or LUN.
- Choosing datastores might improve the overall manageability and operational overhead.
- Datastores might add extra layer of complexity by having another set of timeouts, parameters, log files, and potential bugs.
- Choosing NFS over VMFS datastore helps when using the NetApp SnapRestore technology in a disaster scenario.
- An index that is heavily used can be placed on a fast, highly available disk, such as an expensive solid-state device. Also, a table storing archived data that is rarely used or not performance critical can be stored on a less expensive, slower disk system.
- Tablespaces depend on metadata in the main data directory, and therefore cannot be attached to a different database cluster or backed up individually. Similarly, if you lose a tablespace (through file deletion, disk failure, and so on), the database cluster might become unreadable or unable to start. Placing a tablespace on a temporary file system like a RAM disk risks the reliability of the entire cluster.
- NetApp does not recommend making more than one tablespace per logical file system, because you cannot control the location of individual files in a logical file system.
- NetApp recommends using version 4 of the NFS protocol, because it has an integrated locking feature that version 3 lacks. Also, it has new features such as statefulness, improved security, and strong authentication.
- PostgreSQL does not recommend soft mounting of an NFS file system.
- PostgreSQL also advises mounting the NFS file system synchronously to avoid a data inconsistency problem. For more information, see the PostgreSQL documentation.
- NetApp recommends having four to eight LUNs to support data file I/O. Fewer than four LUNs might create performance issues because of limitations in host SCSI/FC implementations.
- Ensure that the database has multiple tablespaces, with tables and indexes grouped according to the transaction rates.
- The tablespace must be placed across multiple disk file systems for balanced I/O. This balance ensures that multiple CPUs come into play to perform transactions across multiple disks.
Appendix A: Host Configuration

- Verify that FC host bus adapters (HBAs) in the ESX host are running with updated drivers, firmware, and the correct BIOS version.
- Install NetApp Host Utilities on the host to view direct and indirect paths for a given LUN. The kit also provides extended information about the LUN SVM information.
- Always use a rescan SCSI bus script on the host to:
  - Find new LUNs
  - Discard stale LUNs
  - Update new paths to LUNs
- To establish multiple routes to the storage if there is path failure, install and configure multipathing software on the host operating system.
- If you are not using multipathing software, you should limit each LUN to a single path.
- The Linux kernel allows low-level control over the scheduling of I/O to block devices. The defaults in Linux versions might vary considerably. In general, NetApp customers and internal testing show better results with NoOps scheduler for databases. You should perform benchmarking to determine which I/O schedulers are optimal for your use case.

SELinux

Security-Enhanced Linux (SELinux) is a Linux kernel security model that provides a mechanism for supporting access control security policies.

The `/etc/selinux/config` configuration file controls whether SELinux is enabled or disabled, and if enabled, whether SELinux operates in permissive mode or enforcing mode. The `SELINUX` variable can be set to disabled, permissive, or enforcing mode.

- The disabled option completely disables the SELinux kernel and application code, leaving the system running without any SELinux protection.
- The permissive option enables the SELinux code and permits policy-denied accesses but audits them.
- The enforcing option enables the SELinux code and causes it to enforce access denials and audit them.
- In our testing, most deployments observed errors while mounting an NFS file system on a Linux host or while starting the PostgreSQL daemon. Some common errors include:

  Error message: not found; Error message: access denied;
  Error message: Permission denied; Cannot mount / access an NFS volume or file system; Error Message: Cannot find home Directory

You can resolve these errors by correcting the SELinux configuration files and verifying that the export options on NetApp ONTAP are correct. For further troubleshooting steps, see this [NetApp Knowledgebase article](https://www.netapp.com)

To learn more about FC configuration, see the following documentation:

- **SAN Configuration**
  Describes supported FC, iSCSI, and FCoE topologies for connecting host computers to storage controllers in clusters.

- **SAN Administration**
  Describes how to configure and manage the iSCSI, FCoE, and FC protocols for clustered SAN environments, including configuration of LUNs, initiator groups (igroups), and targets.
Appendix B: PostgreSQL Page and File System Layout

Every table and index is stored as an array of pages of a fixed size (usually 8KB), although you can select a different page size by changing the configuration. Each page contains a header that stores metadata about the block. The header provides details about checksum, start of free space, and end of free space. The item after the header is an array identifier composed of offset-length pairs pointing to the actual items.

The structure used to store the table is a heap file. Heap files are lists of unordered records of variable size. The heap file is structured as a collection of pages or blocks, each containing a collection of items. Figure 7 takes a closer look at how PostgreSQL stores data across rows and in each page or block of a table.

Figure 7) Data storage in PostgreSQL.

TOAST

TOAST stands for The Oversized-Attribute Storage Technique. PostgreSQL uses a fixed page size (commonly 8KB) and does not allow tuples to span multiple pages. Therefore, it is not possible to store large field values directly. When you attempt to store a row that exceeds this size, TOAST breaks up the data of large columns into smaller “pieces” and stores them in a TOAST table.

The large values of toasted attributes are pulled out (if selected at all) only at the time the result set is sent to the client. The table itself is much smaller and can fit more rows into the shared buffer cache than it could without any out-of-line storage (TOAST).

VACUUM

In normal PostgreSQL operation, tuples that are deleted or made obsolete by an update are not physically removed from their table; they remain present until VACUUM is run. Therefore, you must run VACUUM periodically, especially on frequently updated tables. The space it occupies must then be reclaimed for reuse by new rows, to avoid disk space outage. However, it does not return the space to the operating system.

The free space inside a page is not fragmented. VACUUM rewrites the entire block, efficiently packing the remaining rows and leaving a single contiguous block of free space in a page.

In contrast, VACUUM FULL actively compacts tables by writing a completely new version of the table file with no dead space. This action minimizes the size of the table but can take a long time. It also requires extra disk space for the new copy of the table until the operation completes. The goal of routine VACUUM is to avoid VACUUM FULL activity. This process not only keeps tables at their minimum size, but also maintains steady-state usage of disk space.
**PostgreSQL File System Layout**

You create a new database cluster by using the `initdb` program. An `initdb` script creates the data files, system tables, and template databases (template0 and template1) that define the cluster. The template database represents a stock database. It contains definitions for system tables, standard views, functions, and data types. `pgdata` acts as an argument to the `initdb` script that specifies the location of the database cluster.

All the database objects in PostgreSQL are internally managed by respective OIDs. Tables and indexes are also managed by individual OIDs. The relationships between database objects and their respective OIDs are stored in appropriate system catalog tables, depending on the type of object. For example, OIDs of databases and heap tables are stored in `pg_database` and `pg_class`, respectively. You can determine the OIDs by issuing queries on the PostgreSQL client.

Every database has its own individual tables and index files that are restricted to 1GB. Each table has two associated files, suffixed respectively with `_fsm` and `_vm`. They are referred to as the free space map and the visibility map. These files store the information about free space capacity and have visibility on each page in the table file. Indexes only have individual free space maps and don’t have visibility maps.

The `pg_xlog/pg_wal` directory contains the write-ahead logs. Write-ahead logs are used to improve database reliability and performance. Whenever you update a row in a table, PostgreSQL first writes the change to the write-ahead log, and later writes the modifications to the actual data pages to a disk. The `pg_xlog` directory usually contains several files, but `initdb` creates only the first one. Extra files are added as needed. Each xlog file is 16MB long.

**Appendix C: Storage Efficiency**

Figure 9 shows how compression and compaction works for databases on ONTAP storage.
Figure 9) Compression and compaction.

Without compression:
- Volume A: 8KB, 8KB, 8KB
- Volume B: 4KB, 4KB
- Volume C: 3x 1KB

After inline adaptive compression:
- 11 blocks

After inline data compaction:
- 7 blocks

After inline data compaction:
- 4 blocks
Refer to the **Interoperability Matrix Tool (IMT)** on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer’s installation in accordance with published specifications.

**Copyright Information**

Copyright © 2019 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Data contained herein pertains to a commercial item (as defined in FAR 2.101) and is proprietary to NetApp, Inc. The U.S. Government has a non-exclusive, non-transferrable, non-sublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b).

**Trademark Information**

NETAPP, the NETAPP logo, and the marks listed at [http://www.netapp.com/TM](http://www.netapp.com/TM) are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.