Advanced PostgreSQL DBA with High Availability

Duration: 10 Days
Prerequisites

Participants should have:

- Basic knowledge of PostgreSQL administration
- Working knowledge of SQL
- Basic Linux command-line experience
- Understanding of database concepts such as transactions, indexes, backup, and recovery
- Basic networking knowledge for replication and HA labs

Day 1 – PostgreSQL Architecture & Advanced Configuration

Module 1: PostgreSQL Architecture

PostgreSQL Server Architecture

- PostgreSQL server architecture
- PostgreSQL database cluster
- Database vs Schema
- Tablespaces
- Data directory
- Client-server communication

PostgreSQL Process Architecture

- Postmaster / main server process
- Backend processes
- WAL Writer
- Checkpointer
- Background Writer
- Autovacuum Launcher
- Autovacuum Workers
- WAL Sender
- WAL Receiver

PostgreSQL Memory Architecture

- Shared memory
- shared_buffers
- work_mem
- maintenance_work_mem
- Per-session memory
- Impact of concurrent connections

PostgreSQL Storage Architecture

- Data directory

- Relation files
- Table and index storage
- Tablespaces
- WAL directory
- pg_wal
- Temporary files

Module 2: Advanced Configuration Management

- postgresql.conf
- pg_hba.conf
- ALTER SYSTEM
- postgresql.auto.conf
- Parameter hierarchy
- Parameter source and context
- SHOW
- current_setting()
- pg_settings
- pg_file_settings
- Finding modified parameters
- Detecting invalid configuration

Connection Configuration

- max_connections
- Connection limits
- Excessive connections
- Connection management
- Connection pooling concepts

Memory Configuration

- shared_buffers
- work_mem
- maintenance_work_mem
- effective_cache_size
- Per-query vs per-session memory

Lab – PostgreSQL Architecture & Configuration

Participants will:

- Connect to a PostgreSQL server and identify the cluster.
- Locate the PostgreSQL data directory.
- Identify postgresql.conf, pg_hba.conf, and postgresql.auto.conf.
- Identify running PostgreSQL processes.
- Map backend and background processes to DBA activities.
- Inspect important memory parameters.
- Modify selected parameters using configuration files.

- Modify parameters using ALTER SYSTEM.
- Reload configuration with pg_reload_conf().
- Identify parameters requiring server restart.
- Introduce an invalid configuration setting and troubleshoot it.
- Validate the final server configuration.

Day Outcome: Participants understand PostgreSQL internal architecture and can safely inspect and modify production configuration.

Day 2 – Security, Authentication, Authorization & Auditing

Module 3: Advanced Security Administration

PostgreSQL Security Architecture

- Authentication
- Authorization
- Roles
- Privileges
- Encryption
- Auditing
- Security boundaries

Role Administration

- CREATE ROLE
- CREATE USER
- LOGIN roles
- Group roles
- Role membership
- GRANT
- REVOKE
- Role inheritance
- SET ROLE

Administrative Role Attributes

- SUPERUSER
- CREATEDB
- CREATEROLE
- REPLICATION
- LOGIN
- CONNECTION LIMIT

Privilege Management

- Database privileges
- Schema privileges
- Table privileges
- Sequence privileges

- Function privileges
- Column privileges
- ALTER DEFAULT PRIVILEGES

Least-Privilege Security

- Application roles
- Read-only roles
- Read/write roles
- DBA roles
- Separation of duties
- Reducing unnecessary SUPERUSER access

Authentication Configuration

pg_hba.conf

- Authentication rules
- Connection types
- Database/user matching
- IP address matching
- Authentication methods
- Rule ordering

SCRAM Authentication

- SCRAM-SHA-256
- Password authentication
- MD5 vs SCRAM
- SCRAM configuration
- MD5-to-SCRAM migration
- Secure password management

Row-Level Security

- RLS architecture
- Enabling RLS
- Policies
- USING
- WITH CHECK
- DML policies
- FORCE ROW LEVEL SECURITY

TLS / SSL Security

- TLS architecture
- Server certificates
- Private keys
- CA certificates
- ssl
- ssl_cert_file

- ssl_key_file
- ssl_ca_file
- hostssl
- TLS verification

Certificate Authentication

- Client certificates
- cert authentication
- pg_ident.conf
- Certificate-to-user mapping
- Mutual TLS
- Certificate lifecycle

Module 4: Logging & Auditing

- PostgreSQL logging architecture
- Logging destinations
- Logging collector
- Log rotation
- Log retention
- log_connections
- log_disconnections
- log_statement
- log_min_duration_statement
- log_duration
- log_line_prefix
- Error logging
- Connection auditing
- Authentication auditing
- DDL/DML auditing
- Privileged activity monitoring
- pgAudit overview
- Centralized logging concepts

Lab – PostgreSQL Security & Auditing

Participants will:

- Create application and administrative roles.
- Build read-only and read/write access models.
- Configure role inheritance.
- Apply schema and table privileges.
- Configure default privileges.
- Configure pg_hba.conf.
- Test permitted and denied connections.
- Configure SCRAM authentication.
- Create an RLS-enabled table.
- Build and validate RLS policies.

- Configure PostgreSQL logging.
- Generate authentication, DDL, and query activity.
- Analyze PostgreSQL logs for failed connections and slow queries.

Day Outcome: Participants can implement enterprise-level PostgreSQL authentication, authorization, logging, and auditing.

Day 3 – Monitoring, Locks & Production Troubleshooting

Module 5: PostgreSQL Monitoring

Monitoring Architecture

- Server monitoring
- Database monitoring
- Session monitoring
- Query monitoring
- Resource monitoring

Important Monitoring Views

- pg_stat_activity
- pg_stat_database
- pg_stat_bgwriter
- pg_stat_wal
- pg_stat_replication
- pg_stat_progress_vacuum

Session Monitoring

- Active sessions
- Idle sessions
- Idle-in-transaction sessions
- Long-running sessions
- Connection counts
- User/session identification

Query Monitoring

- Active queries
- Long-running queries
- Waiting queries
- Query duration
- Query wait events

Lock & Blocking Analysis

- PostgreSQL locks
- Blockers
- Blocked sessions
- Lock chains

- pg_locks
- pg_blocking_pids()
- Deadlocks

Session Management

- pg_cancel_backend()
- pg_terminate_backend()
- Cancel vs terminate
- Production considerations

Troubleshooting Methodology

- Identify symptoms
- Identify affected sessions
- Identify wait events
- Detect blocking chains
- Determine root blocking session
- Safely resolve production blocking

Lab – Blocking, Deadlock & Session Troubleshooting

Participants will:

- Generate multiple concurrent sessions.
- Create a blocking transaction.
- Identify the blocked session.
- Identify the blocker.
- Build a blocking chain.
- Examine pg_locks.
- Use pg_blocking_pids().
- Generate a long-running query.
- Examine wait events.
- Cancel a running query.
- Terminate a backend session.
- Simulate a deadlock.
- Review PostgreSQL deadlock logging.

Day Outcome: Participants can diagnose and resolve common PostgreSQL production incidents involving sessions, locks, waits, and blocked workloads.

Day 4 – MVCC, VACUUM, Autovacuum & Database Bloat

Module 6: PostgreSQL MVCC

MVCC Architecture

- Multi-Version Concurrency Control
- Tuple versions
- Readers and writers

- Transaction visibility
- Snapshots

Transaction Internals

- Transaction IDs
- xmin
- xmax
- Tuple visibility
- Long-running transactions

Dead Tuples & Bloat

- Why dead tuples occur
- Dead tuple accumulation
- Table bloat
- Index bloat
- Performance impact

VACUUM Operations

- VACUUM
- VACUUM FULL
- Vacuum behavior
- Dead tuple cleanup
- Space reuse
- Vacuum limitations

Autovacuum

- Autovacuum Launcher
- Autovacuum Workers
- Vacuum threshold
- Analyze threshold
- Scale factors
- Per-table configuration
- Autovacuum tuning

Transaction ID Management

- Transaction ID aging
- Freeze
- relfrozenxid
- Transaction ID wraparound
- Emergency vacuum

Lab – MVCC, Vacuum & Bloat

Participants will:

- Create tables with update/delete workloads.
- Generate dead tuples.

- Inspect table statistics.
- Identify dead tuple accumulation.
- Detect long-running transactions.
- Perform VACUUM.
- Compare standard VACUUM with VACUUM FULL.
- Examine autovacuum activity.
- Modify autovacuum thresholds.
- Apply table-specific autovacuum settings.
- Investigate table bloat.
- Investigate index bloat.
- Review transaction ID age.
- Identify tables approaching vacuum/freeze thresholds.

Day Outcome: Participants can diagnose MVCC-related problems and tune vacuum/autovacuum behavior for high-volume databases.

Day 5 – Query Optimization, Indexing & Performance Tuning

Module 7: Query Performance Analysis

Query Execution Lifecycle

- Query parsing
- Planning
- Execution
- Planner statistics
- Cost estimation

EXPLAIN

- EXPLAIN
- EXPLAIN ANALYZE
- Estimated rows vs actual rows
- Cost
- Planning time
- Execution time

Execution Plans

- Sequential Scan
- Index Scan
- Index-Only Scan

Index Optimization

PostgreSQL Index Types

- B-tree
- Hash
- GIN

- GiST
- BRIN

Index Problems

- Unused indexes
- Duplicate indexes
- Over-indexing
- Index bloat
- Index maintenance
- REINDEX

Statistics

- ANALYZE
- Planner statistics
- default_statistics_target
- Column statistics
- Extended statistics

Performance Configuration

- shared_buffers
- work_mem
- maintenance_work_mem
- effective_cache_size
- random_page_cost
- Checkpoint parameters
- WAL parameters
- Autovacuum parameters

Query Statistics

- pg_stat_statements
- Query execution statistics
- High-frequency queries
- High-duration queries

Lab – Query & Index Performance Tuning

Participants will:

- Load a sample workload.
- Execute queries without supporting indexes.
- Capture execution plans.
- Compare estimated and actual rows.
- Create B-tree indexes.
- Compare Sequential Scan and Index Scan.
- Test Index-Only Scan.
- Identify unused indexes.
- Identify duplicate indexes.

- Perform ANALYZE.
- Adjust statistics where required.
- Install and configure pg_stat_statements.
- Identify expensive SQL.
- Tune selected configuration parameters.
- Compare query performance before and after tuning.

Day Outcome: Participants can systematically identify poorly performing SQL and optimize queries, indexes, statistics, and configuration.

Day 6 – PostgreSQL Extensions, FDW & Routine DBA Operations

Module 8: PostgreSQL Extensions

Extension Architecture

- What is a PostgreSQL extension?
- Built-in functionality vs extensions
- Installing extensions
- Database-specific extensions
- Extension versioning
- Extension upgrades
- Installed extension inspection
- Removing extensions

Important DBA Extensions

- pg_stat_statements
- pgcrypto
- uuid-ossp
- pg_trgm
- citext
- hstore
- tablefunc

Foreign Data Wrappers

FDW Concepts

- FDW architecture
- file_fdw
- postgres_fdw

postgres_fdw

- Foreign server
- User mapping
- Foreign tables
- Foreign schema
- IMPORT FOREIGN SCHEMA

- Querying remote tables
- Remote DML

FDW Performance

- Query pushdown
- Join pushdown
- Aggregate pushdown
- Network overhead
- fetch_size
- Remote execution

Routine DBA Maintenance

Daily Health Checks

- PostgreSQL availability
- Connection utilization
- Database growth
- Replication status
- Long-running transactions
- Blocking sessions

Maintenance Activities

- VACUUM
- ANALYZE
- REINDEX
- Autovacuum monitoring
- Log management
- Log rotation

Lab – Extensions, FDW & DBA Health Checks

Participants will:

- Install and remove PostgreSQL extensions.
- Enable pg_stat_statements.
- Install postgres_fdw.
- Configure two PostgreSQL databases for FDW testing.
- Create a foreign server.
- Create user mappings.
- Create foreign tables.
- Import a remote schema.
- Query remote PostgreSQL data.
- Perform remote DML.
- Examine FDW query plans.
- Observe query pushdown.
- Build a daily PostgreSQL DBA health-check query set.
- Perform VACUUM, ANALYZE, and REINDEX maintenance.

Day Outcome: Participants can extend PostgreSQL capabilities, integrate PostgreSQL databases using FDW, and implement routine production maintenance procedures.

Day 7 – Backup, Restore & Point-in-Time Recovery

Module 9: PostgreSQL Backup Architecture

Backup Strategy

- Backup objectives
- Recovery requirements
- Logical backup
- Physical backup
- Full backup
- Base backup
- WAL backup

Logical Backup

- pg_dump
- pg_dumpall
- Object-level backup
- Database-level backup

Restore

- psql
- pg_restore
- Restore workflow
- Restore validation

Physical Backup

- pg_basebackup
- Base backup architecture
- WAL relationship with physical backup

Module 10: Point-in-Time Recovery

PITR Concepts

- Point-in-Time Recovery
- WAL-based recovery
- Timestamp recovery
- Named restore points
- Recovery target concepts
- Recovery validation

Lab 1 – Logical Backup & Restore

Participants will:

- Create sample databases and schemas.
- Perform a database backup with pg_dump.
- Perform selected-object backup.
- Restore SQL-format backup with psql.
- Restore archive-format backup using pg_restore.
- Perform cluster-wide logical backup using pg_dumpall.
- Validate restored users and objects.

Lab 2 – Physical Backup

Participants will:

- Prepare PostgreSQL for physical backup.
- Perform pg_basebackup.
- Inspect the backup structure.
- Verify backup consistency.
- Prepare the backup for recovery testing.

Lab 3 – Complete PITR Scenario

Participants will:

- Create a database and application dataset.
- Take a physical base backup.
- Generate database transactions.
- Generate WAL.
- Record a recovery timestamp or restore point.
- Simulate accidental data deletion.
- Stop the database.
- Restore the base backup.
- Configure recovery.
- Replay WAL.
- Recover to the required timestamp / restore point.
- Validate recovered application data.

Day Outcome: Participants can design, perform, and validate logical/physical backups and recover PostgreSQL to a specific point in time.

Day 8 – PostgreSQL Streaming Replication & Replication Troubleshooting

Module 11: Streaming Replication

Replication Architecture

- High Availability concepts
- Primary/standby architecture
- Streaming replication
- WAL Sender
- WAL Receiver
- Replication connections

Replication Configuration

- wal_level
- max_wal_senders
- max_replication_slots
- hot_standby
- primary_conninfo
- Replication users

Building a Standby

- Primary preparation
- Replication role
- pg_basebackup
- Standby initialization
- Standby startup
- Replication validation

Synchronous Replication

- Synchronous replication
- synchronous_commit
- synchronous_standby_names
- Data durability
- Performance trade-offs

Asynchronous Replication

- Async replication
- Replication lag

Replication Slots

- Physical replication slots
- Slot monitoring

Replication Monitoring

- pg_stat_replication
- pg_stat_wal_receiver
- WAL LSN
- Replay LSN
- Replication lag
- WAL lag
- Standby status

Lab – Build and Troubleshoot Streaming Replication

Participants will:

- Prepare a primary PostgreSQL server.
- Create a replication user.
- Configure replication parameters.

- Configure authentication for replication.
- Create a standby using pg_basebackup.
- Start streaming replication.
- Generate transactions on primary.
- Validate replicated data.
- Monitor WAL Sender and WAL Receiver.
- Calculate replication lag.
- Configure a physical replication slot.
- Stop and restart the standby.
- Observe WAL accumulation.
- Test asynchronous replication.
- Configure synchronous replication.
- Compare synchronous and asynchronous behavior.

Day Outcome: Participants can build, monitor, tune, and troubleshoot PostgreSQL streaming replication.

Day 9 – PostgreSQL Upgrade & Patroni HA Architecture

Module 12: PostgreSQL Upgrade

Upgrade Concepts

- Minor upgrades
- Major upgrades
- Upgrade planning
- Pre-upgrade checks
- Application compatibility
- Extension compatibility

pg_upgrade

- pg_upgrade architecture
- Old cluster
- New cluster
- Compatibility checks
- Upgrade process
- Post-upgrade validation

Upgrade Lab

Participants will:

- Review an existing PostgreSQL cluster.
- Inventory databases and extensions.
- Perform compatibility checks.
- Prepare the target PostgreSQL cluster.
- Run pg_upgrade checks.
- Perform a controlled upgrade exercise.

- Validate upgraded databases.
- Validate applications/extensions.
- Review rollback considerations.

Module 13: Patroni High Availability Architecture

PostgreSQL HA Without Patroni

- Streaming replication limitations
- Manual failover
- Manual promotion
- Split-brain risks
- Application endpoint problems

Patroni Overview

- Patroni architecture
- Patroni responsibilities
- Automatic failover
- Automatic leader management
- Advantages of Patroni

Distributed Configuration Store

- DCS concept
- etcd
- Consul
- ZooKeeper
- Leader key
- Cluster state

Patroni Components

- Patroni
- PostgreSQL
- DCS
- REST API
- patronictl

Patroni HA Architecture

- Leader
- Replica
- DCS
- Patroni agent
- Health checking
- Leader election
- Failover mechanism

Lab – Patroni / etcd Preparation

Participants will:

- Review the proposed HA node architecture.
- Prepare PostgreSQL nodes.
- Validate network connectivity.
- Install etcd.
- Configure etcd.
- Start the etcd service.
- Validate DCS health.
- Install Patroni and required dependencies.
- Verify Patroni installation.
- Prepare Patroni YAML configuration.

Day Outcome: Participants understand PostgreSQL upgrade procedures and the architecture and components required to implement Patroni-based HA.

Day 10 – Build Patroni HA Cluster, Failover & Final DBA Practice

Module 14: Building a Patroni Cluster

HA Lab Architecture

- Node layout
- Primary/replica design
- DCS architecture
- Application connection endpoint

Patroni Configuration

- PostgreSQL configuration
- DCS configuration
- REST API configuration
- Replication configuration
- Authentication
- Patroni YAML configuration

Cluster Initialization

- Initialize first node
- Add replicas
- Verify cluster state
- Identify leader
- Validate replication

Patroni Cluster Administration

- patronictl operations
- Cluster status
- Member roles
- Leader identification

- Replica status
- Controlled switchover
- Automatic failover
- Node rejoin
- Split-brain protection
- Cluster health validation

Final Comprehensive HA Lab

Participants will build and test a working Patroni HA environment.

Part 1 – Build the Cluster

- Configure DCS.
- Configure Patroni on PostgreSQL nodes.
- Bootstrap the Patroni cluster.
- Join replica nodes.
- Verify the leader.
- Verify replicas.
- Check replication.

Part 2 – Application Workload

- Connect to the active primary.
- Create test data.
- Generate continuous transactions.
- Verify replication to standby nodes.

Part 3 – Automatic Failover

- Identify the current leader.
- Stop the primary PostgreSQL/Patroni node.
- Observe Patroni health checks.
- Observe leader election.
- Identify the newly promoted leader.
- Verify applicati