# Database Installation, Configuration, and Basic Setup

Comprehensive PostgreSQL training covering fundamental setup, advanced features, high availability, monitoring, and best practices

December 2024

# Day 1 Basic Setup

## PostgreSQL Setup

- Installation methods: package managers, installers, source compilation
- Key configuration files: postgresql.conf, pg_hba.conf
- Basic admin commands:

```
pg_ctl start/stop/restart
psql -U postgres -d database
CREATE USER/DATABASE
```

- Memory allocation parameters: shared_buffers, work_mem

## Practical Exercises

- ✓ Install PostgreSQL in test environment/container
- ✓ Configure basic parameters for development
- ✓ Create database and user with permissions

## MySQL Setup

- Installation overview and system requirements
- Key my.cnf configuration parameters
- Memory parameters: innodb_buffer_pool_size

## SQL Server Setup

- Installation methods: installers, Docker
- Basic configuration parameters overview
- Connection settings: TCP/IP configuration

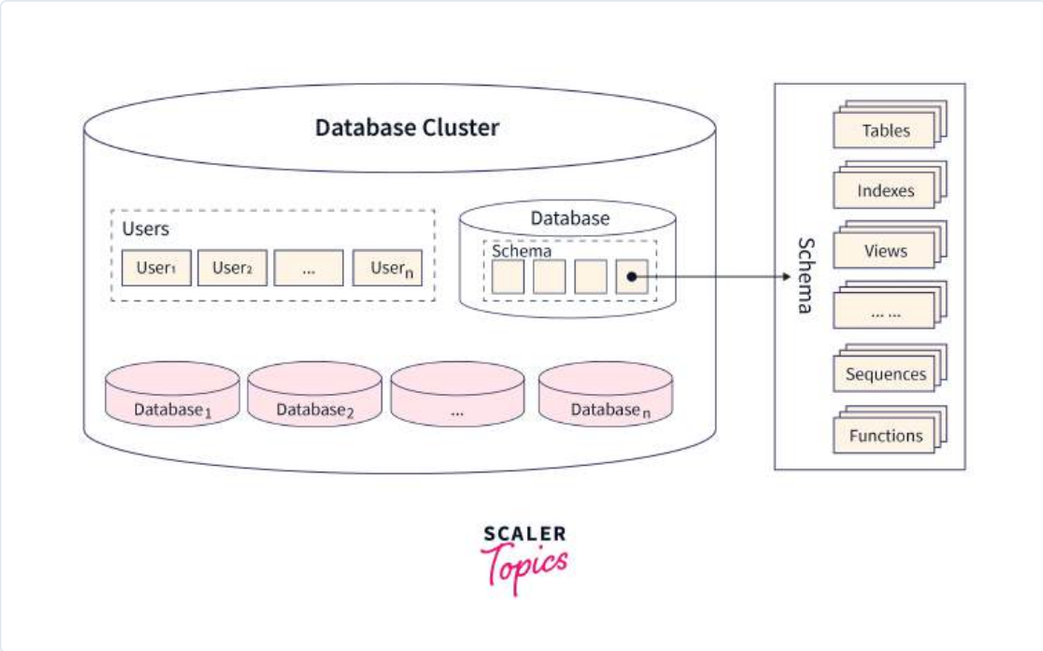
# PostgreSQL Architecture Overview

## Core Components

- **Postmaster**: Main PostgreSQL server process
- **Backend Processes**: Handle client connections
- **Background Workers**: Autovacuum, WAL writer, checkpointer
- **Shared Memory**: Caches and buffers

## Memory Architecture

- **Shared Buffers**: Primary data cache
- **WAL Buffers**: Transaction log caching
- **Work Memory**: Sort operations and hash tables
- **Maintenance Memory**: VACUUM operations




## Storage Architecture

- **Data Files**: Tables and indexes stored as files
- **WAL**: Write-Ahead Log for durability
- **pg_xact**: Transaction status information
- **Configuration Files**: postgresql.conf, pg_hba.conf

# Advanced Installation & Configuration

## Linux-based Installation

- Package manager installation (recommended)

```
# Ubuntu-based systems
sudo apt update
sudo apt install postgresql postgresql-contrib
```

- Source compilation for custom builds
- Default file locations
  - Data: /var/lib/postgresql/[version]/main/
  - Config: /etc/postgresql/[version]/main/
  - Binaries: /usr/lib/postgresql/[version]/bin/

## Post-Installation Tasks

- ✓ Verify installation status

```
sudo systemctl status postgresql
```

- ✓ Enable auto-start on boot

```
sudo systemctl enable postgresql
```

## Key Configuration Files

**postgresql.conf** - Core server parameters

- listen_addresses = '*' (network access)
- max_connections = 100 (concurrency)
- shared_buffers = 256MB (memory allocation)

**pg_hba.conf** - Authentication control

```
# TYPE DATABASE USER ADDRESS METHOD
local all all peer
host all all 127.0.0.1/32 md5
host all all 0.0.0.0/0 md5
```

## Security Considerations

- Strong password policies for roles
- SSL configuration for encrypted connections
- Network access restrictions with pg_hba.conf

# Essential Tools for Administration

## Core PostgreSQL CLI Tools

- **psql** - Interactive SQL client and scripting tool
- **pg_dump/pg_restore** - Backup and recovery utilities
- **pg_ctl** - Server process control (start/stop/restart)

```
pg_ctl start -D /data/postgres
pg_dump -Fc mydb > backup.dump
psql -c "SELECT version();"
```

- **createdb/dropdb** - Database creation/deletion
- **createuser/dropuser** - User management utilities
- **pg_basebackup** - Physical base backup for replication

## Third-party Administration Tools

- **pgAdmin** - Comprehensive web-based GUI
- **Monitoring Tools**
  - Prometheus + Grafana dashboards
  - pg_stat_monitor, pgbadger for log analysis
- **Backup & Recovery Tools**
  - Barman - Backup and recovery manager
  - pg_activity - Real-time monitoring utility
- **Performance Analysis**
  - EXPLAIN, pg_stat views, pg_stat_statements

# High Availability & Replication

## Replication Concepts

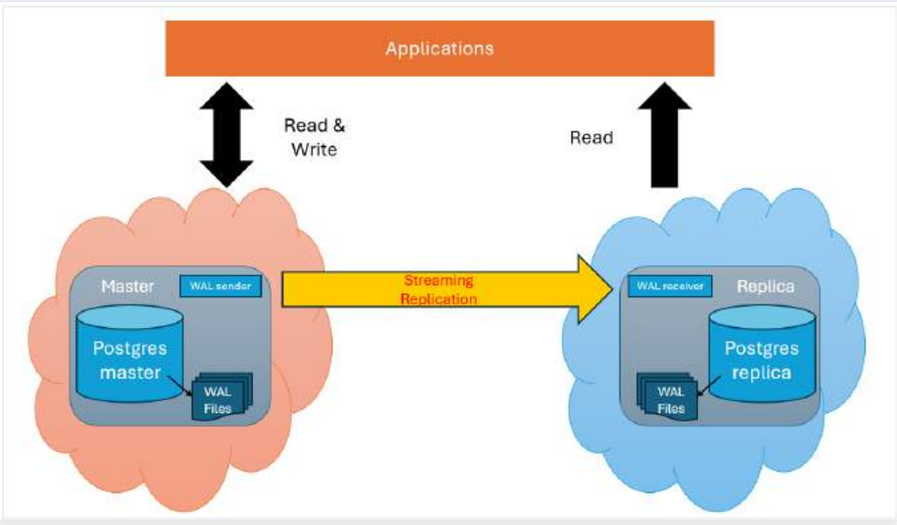
- **RTO (Recovery Time Objective)**: Maximum acceptable downtime after disaster
- **RPO (Recovery Point Objective)**: Maximum acceptable data loss measured in time
- **Synchronous vs. Asynchronous**: Trade-off between performance and data protection

## Replication Types

- **Streaming Replication**: WAL-based physical replication
- **Logical Replication**: Publish/subscribe model for selected tables

**Synchronous Replication Configuration:**

```
synchronous_commit = on
synchronous_standby_names = 'standby1, standby2'
```




## Key Benefits & Use Cases

- **Streaming Replication**: High performance, disaster recovery, read scaling
- **Logical Replication**: Data integration, migration, selective replication
- **Synchronous Replication**: Zero data loss (RPO=0) for critical applications
- **Asynchronous Replication**: Better performance, resilient to network issues

# Streaming Replication Implementation


## Primary Server Configuration

- Configure WAL settings in postgresql.conf:

```
wal_level = replica
max_wal_senders = 3
max_replication_slots = 3
wal_keep_segments = 64
```

- Create replication user with privileges:

```
CREATE ROLE replicator WITH
REPLICATION LOGIN
PASSWORD 'secure_password';
```

- Configure pg_hba.conf for replication access:

```
host replication replicator
192.168.1.11/32 md5
```

## Standby Server Setup

- Create base backup from primary server:

```
pg_basebackup -h 192.168.1.10
-D /var/lib/postgresql/13/main
-U replicator -P -W -R
```

- Configure recovery parameters:

```
primary_conninfo = 'host=192.168.1.10
port=5432 user=replicator
password=secure_password'
```

- Monitor replication status:

```
SELECT client_addr, state,
sent_lsn, write_lsn, flush_lsn
FROM pg_stat_replication;
```

# Logical Replication: Use Cases & Setup

## Publisher/Subscriber Model

Publisher → Subscriber

Logical changes flow from publisher to subscriber

- Replicates data changes at the **logical level** (vs physical WAL)
- Available since PostgreSQL 10+
- Configuration requires **wal_level = logical**
- Allows selective replication of specific tables

## Implementation Example

```
-- Create publication on publisher
CREATE PUBLICATION my_publication
FOR TABLE users, orders, products;

-- Create subscription on subscriber
CREATE SUBSCRIPTION my_subscription
CONNECTION 'host=192.168.1.10 port=5432
user=replicator dbname=mydb'
PUBLICATION my_publication;
```

## Key Use Cases

- ✓ **Cross-version replication** between different PostgreSQL versions
- ✓ **Selective data replication** for specific tables or schema subsets
- ✓ **Data migration** between incompatible storage systems
- ✓ **Multi-master replication** with conflict resolution strategies
- ✓ **Data distribution** across geographic regions

## Advanced Features

- Filter by operation types (INSERT, UPDATE, DELETE)
- Column-level filtering with row filters
- Conflict handling with custom triggers
- Monitoring via **pg_stat_subscription** view

# Patroni Clustering for High Availability

## What is Patroni?

- Sophisticated cluster management solution for PostgreSQL
- Provides automated failover and leader election
- Uses distributed consensus through etcd, Consul, or Zookeeper
- Ensures consistent cluster state across all nodes

## Patroni Configuration

- YAML configuration defines cluster topology
- TTL, loop_wait, retry_timeout parameters
- REST API for cluster status monitoring

```
patronictl -c /etc/patroni/patroni.yml list
```

## Cluster Topology

- Minimum 3 nodes recommended for fault tolerance
- etcd/Consul cluster provides distributed consensus
- HAProxy or similar for client connection routing
- Automated failover with customizable parameters

# Monitoring & Maintenance

## Built-in pg_stat Views

- **pg_stat_database**: Overall database activity and cache hit ratios
- **pg_stat_user_tables**: Table access patterns and autovacuum status
- **pg_stat_activity**: Current sessions and running queries
- **pg_stat_replication**: Replication status and lag

```
-- Monitor cache hit ratio SELECT datname, round(100.0
* blks_hit / (blks_hit + blks_read), 2) AS
cache_hit_ratio FROM pg_stat_database WHERE datname =
current_database();
```

## Logging Configuration

- Configure comprehensive logging in postgresql.conf
- Key parameters: log_destination, log_min_duration_statement
- Use CSV logging format for structured analysis

## Third-Party Monitoring Tools

- **pgAdmin**: Web-based administration and monitoring
- **Prometheus & Grafana**: Metrics collection and visualization
- **postgres_exporter**: Exposes metrics in Prometheus format
- **pgBadger**: Advanced log analysis and reporting



Example Grafana PostgreSQL monitoring dashboard

## Regular Maintenance Tasks

- VACUUM operations to reclaim storage space
- ANALYZE to update query planner statistics
- Index maintenance and reindexing

# Disaster Recovery Planning & Backups

## Backup Strategies

- Full database backups with `pg_dump` and `pg_dumpall`
- Physical backups with `pg_basebackup` for rapid recovery
- WAL archiving for Point-In-Time Recovery (PITR)
- Automated backup verification through test restores

## Geographically Distributed Replicas

- Cross-region replication for regional disaster protection
- WAN optimization: `wal_compression = on`
- Network reliability configuration with TCP keepalives

## RTO & RPO Planning

- **RTO** (Recovery Time Objective): Maximum acceptable downtime
- **RPO** (Recovery Point Objective): Maximum acceptable data loss

**RTO/RPO Optimization Strategies:**

- Synchronous replication: RPO = 0 (no data loss)
- Asynchronous replication: RPO = seconds to minutes
- Log shipping: RPO = backup interval
- PITR: RPO = transaction log retention period

## Backup Verification

- Regular automated test restores to validate backups
- Verify all backup types: full, incremental, and WAL archives
- Use `pg_verifybackup` for backup consistency checks
- Scheduled disaster recovery exercises to test procedures

# Practical Lab Scenarios

## Environment Setup

- Multi-VM/container PostgreSQL cluster setup
- Network configuration for inter-instance communication
- Shared storage setup for backups and configs

## Disaster Recovery Drills

- Backup verification and validation exercises
- Point-in-time recovery simulations
- Cross-region recovery testing

> **Lab Best Practice:** Always test disaster recovery procedures regularly to ensure they work when needed and meet RTO/RPO objectives.

## Replication Exercises

- Configure streaming replication between primary and standby
- Set up logical replication for selective table replication
- Implement and test replication slots
- Practice manual and automated failover procedures

## Monitoring & Troubleshooting

- Set up Prometheus and Grafana for PostgreSQL monitoring
- Analyze pg_stat views to identify performance bottlenecks
- Troubleshoot common database issues (locks, bloat, etc.)

# Troubleshooting & Best Practices

## Common Issues & Solutions

### Connection Refused Errors

> Check listen_addresses in postgresql.conf and client authentication in pg_hba.conf

### Authentication Failures

> Verify user exists, password is correct, and authentication method in pg_hba.conf

### Slow Queries & Performance

> Run EXPLAIN ANALYZE, check indexes, tune memory parameters, run VACUUM regularly

### Replication Lag

> Check network bandwidth, disk I/O, and tune wal_sender parameters

## Best Practices

- **Security:** Use strong passwords, restrict network access, update regularly
- **Performance:** Right-size shared_buffers (~25% RAM), tune work_mem, optimize indexes
- **Maintenance:** Schedule regular VACUUM, monitor bloat, update statistics
- **Monitoring:** Set up alerts for replication lag, disk space, connection count
- **Backups:** Test recovery regularly, maintain off-site copies, automate verification
- **Schema Design:** Normalize appropriately, use constraints, choose proper data types

# Resources, References & Q&A

## Documentation & Learning Resources

- PostgreSQL Official Documentation
- PostgreSQL Admin Guide
- Microsoft Learn SQL Server Basics
- MySQL Official Documentation

## Community & Support

- PostgreSQL Mailing Lists
- PostgreSQL on Stack Overflow
- PostgreSQL Slack Community
- PostgreSQL Events & Conferences

### Questions & Open Discussion

Do you have any questions about PostgreSQL installation, configuration, high availability, or any other topics we've covered today? Feel free to ask!