“Develop a passion for learning.”

6. Monitoring

Introduction to PostgreSQL

AGENDA

- Base monitoring
- Advanced monitoring
- Tools

SYSTEM TOOLS MONITORING

- Generally PostgreSQL instances run on Unix
  - There are a number of Unix system monitoring tools that can be used
  - These treat PostgreSQL like any other process
  - Useful for monitoring resource usage
- ‘ps’ - monitors system processes, checks the status of PostgreSQL processes
  - “ps aux | grep postgres”
- ‘top and htop’ - monitor real-time system performance, including CPU, memory usage being for each running processes.
- ‘vmstat’ - Monitor system performance metrics, including CPU, memory, and I/O statistics.
  - Check for I/O bottlenecks and memory swapping that could impact PostgreSQL performance.

SYSTEM TOOLS MONITORING

- ‘iostat’ - monitor disk I/O statistics
  - Identify disk read/write patterns and potential I/O bottlenecks that can affect database performance.
- ‘ netstat’ - Monitor network connections and statistics
- ‘pg_top’ - A PostgreSQL-specific tool based on top for monitoring active sessions, queries, and performance statistics.
  - View running queries, resource consumption by each session, and overall database activity.

BASIC MONITORING

- Basic monitoring in PostgreSQL focuses on ensuring the general health and performance of the database.
- The key areas include checking system resources, database connectivity, instance availability, and basic performance metrics.
- Instance Availability and Connection Monitoring
  - pg_isready, simple command-line tool used to check the availability of a PostgreSQL instance. Equivalent to a "ping" for the database.
  - MetricsK Status of the server (up or down), response time for connection attempts.
- System Resource Monitoring
  - CPU and Memory: Monitor CPU usage, memory usage, and swap space to ensure the database has sufficient resources.
  - Disk I/O: Monitor read/write latency, IOPS, and throughput using tools like iostat, vmstat, or system-specific monitoring solutions.
  - Disk Space: Regularly check disk space to prevent out-of-space errors, which can severely impact PostgreSQL, as it relies heavily on disk for data storage and temporary operations.

BASIC MONITORING

- Database Logs
  - Log Files: PostgreSQL logs important events such as connection attempts, errors, warnings, and slow queries.
  - Configuration: Set appropriate logging levels in postgresql.conf (log_min_duration_statement, log_error_verbosity, etc.) to capture relevant events without overwhelming the system with logs.
  - Log Analysis: Use tools like pgBadger or custom scripts to parse and analyze logs for insights into errors, performance issues, and suspicious activities.
- Basic Performance Metrics
  - Tools: pg_stat_activity, pg_stat_database, pg_stat_user_tables.
  - Key Metrics: Active sessions and connections.
  - Database-wide statistics such as transaction counts, number of deadlocks, and cache hit ratios.
  - Table-level statistics like sequential scans, index scans, and tuples read/returned.
  - Usage: These views provide an overall view of database activity and can be queried regularly or integrated into a monitoring dashboard.

ADVANCED MONITORING

- Goes beyond basic checks
  - Involves detailed analysis of performance, query optimization, resource contention, and deep insights into PostgreSQL’s internal operations.
- Advanced Performance Monitoring
  - Dynamic Performance Views: PostgreSQL’s pg_stat_* views are similar to Oracle’s V$ views.
  - pg_stat_activity: Shows all current connections and their states, useful for identifying blocking and long-running queries.
  - pg_stat_statements: Tracks execution statistics for all SQL statements executed by the server, helping identify slow queries, high resource consumers, and optimization opportunities.

ADVANCED MONITORING

- Advanced Performance Monitoring
  - pg_stat_bgwriter: Provides insights into background writer activity, checkpoints, and buffer management, which are critical for understanding I/O performance.
  - Query Optimization: Utilize EXPLAIN and EXPLAIN ANALYZE to review query plans and execution times, identifying areas where indexes, rewrites, or configuration changes could improve performance.
- Resource Contention and Wait Event Analysis
  - Wait Events: PostgreSQL has a wait event monitoring system that tracks what processes are waiting on, similar to Oracle’s wait events.
  - Blocking and Deadlock Detection: Identify and resolve lock contention and deadlocks using pg_locks and pg_blocking_pids() functions.
  - Buffer and Cache Analysis: Monitor buffer usage and cache hit ratios through pg_buffercache and pg_stat_database. Low cache hit ratios can indicate inefficient queries or inadequate memory allocation.

ADVANCED MONITORING

- Replication and High Availability Monitoring
  - Streaming Replication: Monitor replication status using views like pg_stat_replication, which shows the state of replication, lag, and connection status of replicas.
  - Failover and Recovery: Tools like repmgr or Patroni can be used to manage and monitor high availability setups, providing alerts and automation for failover scenarios.
- System-Level Metrics Integration
  - Prometheus and Grafana: Use Prometheus with exporters like node_exporter and postgres_exporter for detailed time-series data on system and database performance.
  - Detailed Metrics: Monitor CPU, memory, disk I/O, and network metrics alongside database-specific metrics for a holistic view of system health.
- Alerting and Automated Responses
  - Threshold-Based Alerts: Set up alerts for critical metrics such as high CPU usage, long-running queries, replication lag, and disk space.
  - Automated Responses: Implement automated actions in response to certain alerts, such as killing runaway queries, initiating failovers, or adjusting configuration parameters.

ADVANCED MONITORING

- Replication and High Availability Monitoring
  - Streaming Replication: Monitor replication status using views like pg_stat_replication, which shows the state of replication, lag, and connection status of replicas.
  - Failover and Recovery: Tools like repmgr or Patroni can be used to manage and monitor high availability setups, providing alerts and automation for failover scenarios.
- System-Level Metrics Integration
  - Prometheus and Grafana: Use Prometheus with exporters like node_exporter and postgres_exporter for detailed time-series data on system and database performance.
  - Detailed Metrics: Monitor CPU, memory, disk I/O, and network metrics alongside database-specific metrics for a holistic view of system health.
- Alerting and Automated Responses
  - Threshold-Based Alerts: Set up alerts for critical metrics such as high CPU usage, long-running queries, replication lag, and disk space.
  - Automated Responses: Implement automated actions in response to certain alerts, such as killing runaway queries, initiating failovers, or adjusting configuration parameters.

ADVANCED MONITORING

- Advanced Log Analysis
  - Log Aggregation and Analysis: Use tools like ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk to aggregate and analyze PostgreSQL logs for patterns, anomalies, and trends.
  - Custom Alerts from Logs: Set up alerts based on specific log entries, such as repeated connection failures, frequent checkpoint warnings, or slow queries exceeding a defined threshold.
- Key Differences from Oracle Monitoring
  - Schema and Object Ownership: PostgreSQL's schema design and object ownership can influence monitoring strategies, as it has a more granular permissions model compared to Oracle.
  - Autovacuum and Vacuum Monitoring: Unlike Oracle's automated segment management, PostgreSQL relies on autovacuum processes to reclaim space, which needs to be closely monitored and tuned to avoid performance degradation.
  - Configuration Flexibility: PostgreSQL allows extensive tuning via configuration files (postgresql.conf), requiring regular review and adjustment based on monitored performance data.

MONITORING TOOLS

- Sampling of the most common tools
  - Links to the tools are in the Notes file in the repository
- *PdAdmin*: A popular open-source tool that provides a graphical interface for managing PostgreSQL databases. It includes built-in monitoring and graphing tools.
- *pg_stat_statements*: A PostgreSQL extension that tracks execution statistics of all SQL statements executed by a server.
  - Available within PostgreSQL; it needs to be enabled by adding it to the shared_preload_libraries in the postgresql.conf file.
- *PgBadger*: A fast PostgreSQL log analyzer that generates detailed reports on performance based on log files.

MONITORING TOOLS

- *pg_top*: Similar to the Unix top command but for PostgreSQL, providing a real-time view of database processes, queries, and statistics.
- *PostgreSQL Exporter for Prometheus*: Collects PostgreSQL metrics and exports them to Prometheus for monitoring and alerting.
- *Pgmetrics*: A command-line tool that collects various statistics and configurations from a running PostgreSQL server and displays them in a detailed report format.
- *Percona*: A comprehensive monitoring tool that supports PostgreSQL and offers insights into database performance with advanced dashboards and alerts.
- *pg_stat_kcache:* A PostgreSQL extension that provides statistics on CPU and I/O usage for all SQL statements.

MONITORING TOOLS

- *TimescaleDB*: A time-series database based on PostgreSQL that includes additional features for monitoring and managing time-series data.
- *pgCluu*: A PostgreSQL clusters performance monitoring tool that collects, stores, and visualizes statistics from PostgreSQL and the operating system.

LAB 6

- The lab materials and instructions are in the repository

End Module