Summary of the Resource Hog Detection and Notification Script Journey

**Goal**

The ultimate goal was to create a robust SQL Server resource monitoring and notification system running on an orchestration server (AZRDBA01, an Azure VM with SQL Server) to:

* Detect resource-intensive sessions (HighCPU, HighIO, LongRunning) across multiple SQL Server instances and Azure SQL Databases via Linked Servers.
* Log issues in tables (dbo.ResourceHogLog, dbo.NotificationLog, dbo.ErrorLog).
* Send email notifications to session owners (e.g., stephenm@consumerdirectcare.com) or the DBA team (dbo.DBATeam) with details (SPID, query text, duration, etc.).
* Initially target CMS\_LYNSQLDEVSL01 (SQL Server) and CMS\_u2\_sql\_PreProd\_master (Azure SQL, ~20 databases), with scalability for the entire environment.

The system aimed to improve database performance by proactively alerting users and DBAs to resource hogs, with plans for future auto-kill functionality.

**Techniques Tried**

* Initial Script for SQL Server Instances:
* Approach: Queried sys.dm\_exec\_sessions and sys.dm\_exec\_requests via Linked Servers to detect sessions with high CPU (>95% of total), high I/O (>95%), or long-running queries (>1 minute).
* Tables: Created dbo.ResourceHogLog (session details), dbo.NotificationLog (email records), dbo.DBATeam (DBA emails), and dbo.ErrorLog (errors).
* Notifications: Used sp\_send\_dbmail to email session owners (domain accounts like NIGHTINGALE\StephenM) or dbo.DBATeam for non-domain accounts.
* Debugging: Added #CurrentUsage, #RawDMVData, and #CursorDebug for tracing.
* Time Zone Handling:
* Adjusted DATEDIFF calculations with @TimeZoneOffsetMinutes to account for UTC (orchestration server) vs. PDT (CMS\_LYNSQLDEVSL01).
* Email Enhancements:
* Added CC to dbo.DBATeam for domain accounts.
* Formatted emails with HTML for clarity (SPID, query text, duration, etc.).

**Failures and How We Overcame Them**

* Missing dbo.ErrorLog Table:
  + Failure: The script assumed dbo.ErrorLog existed, but it wasn’t in your system databases, causing silent email failures.
  + Solution: Added CREATE TABLE IF NOT EXISTS dbo.ErrorLog to ensure error logging, revealing issues like invalid @Recipients.
  + Impact: Enabled debugging of subsequent failures.
* Email Syntax Error (@cc):
  + Failure: Used @cc instead of @copy\_recipients in sp\_send\_dbmail, causing errors:
  + Failed to send email for SPID 52: @cc is not a parameter for procedure sp\_send\_dbmail.
  + Solution: Corrected to @copy\_recipients, ensuring emails sent to stephenm@consumerdirectcare.com and CC’d dbo.DBATeam.
  + Impact: Emails started flowing, as seen in your 25-email success.
* Timing Mismatch for LongRunning:
  + Failure: Emails reported >400 minutes instead of ~2-3 minutes due to a time zone mismatch:
  + Orchestration server: UTC.
  + CMS\_LYNSQLDEVSL01:
  + Local: PDT (UTC-7).
  + Via Linked Server: UTC.
  + DATEDIFF with @TimeZoneOffsetMinutes = -420 over-corrected, as start\_time was UTC via Linked Server.
  + Solution: Removed @TimeZoneOffsetMinutes for SQL Server instances, aligning start\_time and GETDATE() in UTC.
  + Impact: Fixed duration calculations, ensuring ~3-minute reports.
* Query Detection Issues:
  + Failure: Test queries showed No active query in QueryText, indicating sys.dm\_exec\_requests missed active requests (sql\_handle or start\_time NULL).
  + Solution: Switched to JOIN (not LEFT JOIN) on sys.dm\_exec\_requests to focus on active queries, increased @MaxIterations to keep queries running, and lowered @LongRunningThresholdMinutes to 0.5.
  + Impact: Captured test queries reliably, as seen with SPID 52.
  + Invalid Column References:
  + Failure: Referenced s.sql\_handle in sys.dm\_exec\_sessions, which doesn’t exist:
  + Error querying CMS\_LYNSQLDEVSL01: Invalid column name 'sql\_handle'.
  + Solution: Used only r.sql\_handle from sys.dm\_exec\_requests, with OUTER APPLY for query text.
  + Impact: Eliminated query errors, populating #CurrentUsage.

**Ongoing Considerations**

* Time Zone Consistency:
  + Ensure Linked Server queries return consistent time zones (UTC for CMS\_LYNSQLDEVSL01).
  + Monitor for PDT vs. PST changes (daylight saving time) on local servers.
  + If new SQL Server instances use different time zones, add server-specific offsets.
* Query Activity:
  + Queries must be active (sys.dm\_exec\_requests) to trigger LongRunning. Idle or completed queries show No active query.
  + Consider sys.dm\_exec\_query\_stats for recent query history if needed.
* Scalability:
  + Your accidental full-environment run proved the script scales to multiple SQL Server instances (CMS\_AZRSQLC01, CMS\_LYNSQLC01, etc.).
  + Optimize for large estates by filtering Linked Servers (name = 'CMS\_LYNSQLDEVSL01') or batching queries.
* Organizational Challenges:
  + Azure SQL exclusion is temporary due to DENY permissions on cms\_admin. Document permission needs (VIEW DATABASE STATE, SELECT on master.dbo.sys.dm\_exec\_sessions) for future negotiations.
  + Engage stakeholders to clarify cms\_admin restrictions and plan re-inclusion.
* Future Azure SQL Integration:
  + When permissions are restored, use a master database view (dbo.vw\_ResourceHogs) to aggregate sys.dm\_exec\_requests and sys.dm\_exec\_sessions.
  + Implement CREATE OR ALTER to handle schema deployments.
  + Consider a SQL Agent job on AZRDBA01 to push view creation scripts if Linked Server restrictions persist.
* Auto-Kill Logic:
  + Planned for V2 but paused. Reintroduce with strict filters (e.g., exclude cms\_admin, service accounts) to avoid killing critical sessions.

**Goal Reaffirmed**

* The goal remains to proactively monitor and manage SQL Server resource usage by:
  + Detecting HighCPU, HighIO, and LongRunning sessions across SQL Server instances.
  + Logging details for auditing and analysis (dbo.ResourceHogLog, dbo.NotificationLog).
  + Notifying users and DBAs via email to optimize or terminate problematic queries.
  + Scaling to the entire on-premises environment (as proven by your 25-email run!).
  + Eventually re-integrating Azure SQL Databases when permissions and feature limitations are resolved.

**Next Steps**

* Your full-environment success suggests the script is production-ready for SQL Server instances. To polish it:
  + Test Timing: Re-run on CMS\_LYNSQLDEVSL01 to confirm LongRunning emails show ~3 minutes.
  + Filter Servers: Add name = 'CMS\_LYNSQLDEVSL01' to the cursor for controlled testing, unless you want estate-wide monitoring.
  + Document for Azure SQL: Prepare a proposal for your team to restore cms\_admin permissions and test master database views.
  + Auto-Kill: Discuss if you’re ready to add the auto-kill logic from V1 plans.
  + Run the script on CMS\_LYNSQLDEVSL01 and debug using:
    - dbo.ErrorLog.
    - #CurrentUsage, dbo.ResourceHogLog, dbo.NotificationLog.
    - Email timing (~3 minutes?) and any estate-wide insights from the 25 emails.