TempDB is a crucial system database in SQL Server, used for various temporary storage needs, including temporary tables, sorting, and versioning. Proper configuration and management of TempDB is essential for achieving optimal performance and avoiding contention issues. Below are some best practices for managing TempDB in SQL Server:

1. Configure Multiple Data Files

- Add multiple TempDB data files: A common guideline is to configure the number of TempDB data files equal to the number of CPU cores (up to 8). If there are more than 8 cores, start with 8 data files and monitor performance, adding more files if necessary. This minimizes contention on allocation pages (PFS, GAM, SGAM).
- Evenly size all TempDB data files: Ensure that all data files are the same size to enable SQL Server to distribute workload evenly across them. This avoids overloading one file while others remain idle.

Example:

```
ALTER DATABASE tempdb MODIFY FILE (NAME = 'tempdev', SIZE = 512MB);
ALTER DATABASE tempdb ADD FILE (NAME = 'tempdev2', FILENAME = 'C:\TempDB\tempdev2.ndf', SIZE = 512MB);
-- Repeat for additional files
```

2. Place TempDB on Fast Storage

- Use high-performance disks: Place TempDB on fast storage (e.g., SSDs) to reduce I/O latency. TempDB is often a bottleneck due to frequent read and write operations.
- Avoid using system drives: Do not place TempDB on the same drive as the operating system or SQL Server binaries. Use dedicated drives for TempDB files to avoid I/O contention with other operations.

3. Set Appropriate Initial Size and Autogrowth Settings

- Set a large initial size: Pre-size TempDB data files based on the expected workload to minimize the need for autogrowth events. Autogrowth can cause performance hits due to locking during file expansion.
- Set a fixed autogrowth size: Use fixed-size autogrowth increments (e.g., 512 MB or 1 GB) instead of percentage-based growth. This ensures predictable growth behavior.
- Enable Instant File Initialization: Instant File Initialization allows SQL Server to skip zeroing out data files during growth, speeding up the process. This can be enabled by giving the SQL Server service account the “Perform Volume Maintenance Tasks” privilege.

Example:

```
ALTER DATABASE tempdb MODIFY FILE (NAME = 'tempdev', SIZE = 512MB, FILEGROWTH = 512MB);
ALTER DATABASE tempdb MODIFY FILE (NAME = 'templog', SIZE = 512MB, FILEGROWTH = 256MB);
```

4. Minimize Use of TempDB

- Optimize queries: Reduce unnecessary use of TempDB by optimizing queries that create temporary tables, table variables, and perform large sorts. Avoid using TempDB when not necessary.
- Use memory-optimized alternatives: Where possible, use in-memory solutions, such as hash joins in memory or table variables stored in memory (if available), to reduce TempDB usage.

5. Enable Trace Flag 1118 (if necessary)

- Trace Flag 1118 ensures that TempDB allocates uniform extents (instead of mixed extents) for all objects, reducing contention on SGAM pages. Starting from SQL Server 2016, this behavior is enabled by default, so you do not need to set this trace flag unless you're using an older version of SQL Server.

To enable Trace Flag 1118:

```
DBCC TRACEON(1118, -1);
```

6. Use Separate Drives for TempDB Log and Data Files

- Place TempDB log files on a separate drive: Just like other databases, the log file (`templog.ldf`) can become a bottleneck, especially in heavy write environments. Placing it on a separate physical drive from the data files can improve performance.

7. Monitor TempDB Space Usage and Performance

- Monitor space usage: Regularly monitor the space usage in TempDB to avoid running out of space, which can cause queries to fail. Use DMVs like `sys.dm_db_file_space_usage` to track space utilization.
- Monitor waits and contention: Use DMVs like `sys.dm_os_wait_stats` to identify TempDB-related waits (`PAGELATCH_XX`) and contention.

Sample DMV query:

```
SELECT session_id, wait_type, wait_time, resource_description
FROM sys.dm_exec_requests
WHERE database_id = 2 -- TempDB
AND wait_type LIKE 'PAGEIOLATCH_%'
```

8. Monitor Version Store Size

- Keep an eye on the version store: The version store in TempDB holds row versions used by features like snapshot isolation or read-committed snapshot isolation (RCSI). If these isolation levels are in heavy use, the version store can grow and cause performance issues.
- Adjust query behavior: Monitor and limit the use of isolation levels that use the version store if TempDB grows excessively.

Monitor version store size:

```
SELECT SUM(version_store_reserved_page_count) * 8 AS VersionStoreSizeKB
FROM sys.dm_db_file_space_usage;
```

9. Use Resource Governor to Limit TempDB Usage (Optional)

- In some cases, you may want to limit excessive TempDB usage by specific workloads or sessions. SQL Server's Resource Governor can be used to manage TempDB resource allocation and prevent overuse by specific workloads.

Example to configure Resource Governor:

```
CREATE WORKLOAD GROUP TempDBLimitedGroup
WITH (REQUEST_MAX_MEMORY_GRANT_PERCENT = 25);
ALTER RESOURCE GOVERNOR
CONFIGURE;
```


Conclusion

Following these best practices for TempDB management helps to optimize performance, reduce contention, and ensure SQL Server can handle heavy workloads efficiently. TempDB configuration, monitoring, and proper usage are key elements in maintaining a well-performing SQL Server environment.