# Configuration (Managed Instance)

Last updated by | Vitor Tomaz | Aug 5, 2020 at 12:42 PM PDT

```
//*****************************************************************************************************************
// Configurations, failover, table size
//*****************************************************************************************************************
// C.01
// Primary Node and Secondary Node History and code_package_version history
// B-instance has AppName like b-
// Run this entire with UNION other wise B-instance will mess up the query
// If this query returns no rows, two possiblities
//  a. wrong  center
//  b. standard edition used. in this case, use another query C.01.A
let ResourceStats = materialize(MonDmRealTimeResourceStats
//| where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
| filter replica_type == 0
| summarize AppFirstAppearTime=min(TIMESTAMP), (AppLastAppearTime, physical_database_guid)
=arg_max(TIMESTAMP, physical_database_guid)
   by AppName, LogicalServerName, database_name);
let clPhysicalDbGuid = toscalar(MonDmRealTimeResourceStats | project physical_database_guid);
let dbNameForHadrReplicaStates = iif(isnotempty(clPhysicalDbGuid), clPhysicalDbGuid,
'{LogicalDatabaseName}');
let PoolInfo = ResourceStats
| join kind = inner
(
MonResourcePoolStats
//| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where LogicalServerName=~ "{LogicalServerName}"
| summarize by LogicalServerName, resource_pool_name, AppName
) on AppName
| join kind= leftouter (
MonAnalyticsRPSnapshot
| where  logical_server_name == "{LogicalServerName}"
| summarize min(TIMESTAMP) by logical_server_name,resource_pool_name=name,
resource_pool_dtu_guarantee, database_dtu_cap
) on resource_pool_name
| project AppFirstAppearTime, logical_server_name , database_name , AppName , resource_pool_name,
resource_pool_dtu_guarantee , database_dtu_cap ;
(MonDmDbHadrReplicaStates
| where LogicalServerName =~ "{LogicalServerName}" and logical_database_name  =~
dbNameForHadrReplicaStates
| where is_primary_replica == 1  and is_forwarder ==0
| where AppName notcontains "b-"
| order by TIMESTAMP asc nulls first
| serialize
| extend prevNodeName=prev(NodeName)
```

```
| extend nextNodeName = next(NodeName)
| extend isFirst= (NodeName != prevNodeName)
| extend isLast=(NodeName != nextNodeName)
| where isFirst == true or isLast == true
| extend EndTime=next(TIMESTAMP)
| extend StartTime=TIMESTAMP
| where isFirst ==true
| extend codeversion=extract("^(.*)-WASD", 1, code_package_version)
| extend PrimaryNodeName=NodeName, PrimaryNodeFirstAppearTime=StartTime,
PrimaryNodeLastAppearTime=EndTime
| extend fake="fake"
| project ClusterName,  LogicalServerName, logical_database_name, PrimaryNodeName, AppName,
PrimaryNodeFirstAppearTime, PrimaryNodeLastAppearTime,  codeversion, fake,
primary_group_database_id=group_database_id
)
|union
(MonDmDbHadrReplicaStates
| where LogicalServerName =~ "{LogicalServerName}"  and  logical_database_name  =~ "
{LogicalDatabaseName}"
| where AppName contains "b-"
| where is_primary_replica == 1 and   is_forwarder ==0
| order by TIMESTAMP asc nulls first
| serialize
| extend prevNodeName=prev(NodeName)
| extend nextNodeName = next(NodeName)
| extend isFirst= (NodeName != prevNodeName)
| extend isLast=(NodeName != nextNodeName)
| where isFirst == true or isLast == true
| extend EndTime=next(TIMESTAMP)
| extend StartTime=TIMESTAMP
| where isFirst ==true
| extend codeversion=extract("^(.*)-WASD", 1, code_package_version)
| extend PrimaryNodeName=NodeName, PrimaryNodeFirstAppearTime=StartTime,
PrimaryNodeLastAppearTime=EndTime
| extend fake="fake"
| project ClusterName,  LogicalServerName,  logical_database_name, PrimaryNodeName, AppName,
PrimaryNodeFirstAppearTime, PrimaryNodeLastAppearTime,  codeversion,fake,
primary_group_database_id=group_database_id )
| join kind= leftouter  (
   MonDmDbHadrReplicaStates
   | where LogicalServerName =~ "{LogicalServerName}"  and  logical_database_name  =~ "
{LogicalDatabaseName}"
   | extend SecondaryNodeName=NodeName
   | where is_primary_replica == 0 and is_forwarder ==0
   | project Secondary_TIMESTAM=TIMESTAMP, SecondaryNodeName , AppName,
secondary_group_database_id = group_database_id
) on $left.AppName==$right.AppName  and $left.primary_group_database_id ==
$right.secondary_group_database_id  // group_database_id different of replica is for geo partner
//| extend SecondaryNodeName=case (primary_group_database_id!=secondary_group_database_id and
```

```
PrimaryNodeName==SecondaryNodeName, "", SecondaryNodeName)  // Standard edition can have geo replication
| where (Secondary_TIMESTAM >= PrimaryNodeFirstAppearTime and Secondary_TIMESTAM <= PrimaryNodeLastAppearTime and PrimaryNodeName != SecondaryNodeName) or isnull(SecondaryNodeName) == true or SecondaryNodeName == ""
| summarize  SecondaryNodeNames=makeset(SecondaryNodeName) by bin(PrimaryNodeFirstAppearTime, 1s), bin(PrimaryNodeLastAppearTime,1s),  ClusterName,LogicalServerName, logical_database_name, AppName, codeversion, PrimaryNodeName
| join kind= leftouter (
  MonResourcePoolStats
| where LogicalServerName=~ "{LogicalServerName}"
| summarize by AppName, resource_pool_name
 ) on AppName
| join kind= leftouter (
   PoolInfo
) on AppName
| project  ClusterName, AppName, PrimaryNodeName, SecondaryNodeNames, format_datetime(PrimaryNodeFirstAppearTime,'yyyy-MM-dd HH:mm:ss'), format_datetime(PrimaryNodeLastAppearTime,'yyyy-MM-dd HH:mm:ss'),  codeversion, resource_pool_name, resource_pool_dtu_guarantee, database_dtu_cap
|order by PrimaryNodeFirstAppearTime asc nulls last , PrimaryNodeLastAppearTime asc nulls last

// C.01.A
// Primary Node History without cluster
// use above query when possible
let ResourceStats = materialize(MonDmRealTimeResourceStats
//| where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
| filter replica_type == 0
| summarize AppFirstAppearTime=min(TIMESTAMP), (AppLastAppearTime, physical_database_guid) =arg_max(TIMESTAMP, physical_database_guid)
    by AppName, LogicalServerName, database_name);
let clPhysicalDbGuid = toscalar(MonDmRealTimeResourceStats | project physical_database_guid);
let dbNameForHadrReplicaStates = iif(isnotempty(clPhysicalDbGuid), clPhysicalDbGuid, '{LogicalDatabaseName}');
let PrimaryNodes=
(MonDmDbHadrReplicaStates
| where LogicalServerName =~ "{LogicalServerName}"  and  logical_database_name  =~ dbNameForHadrReplicaStates
| where is_primary_replica == 1
| where AppName notcontains "b-"
| order by TIMESTAMP asc nulls first
| serialize
| extend prevNodeName=prev(NodeName)
| extend nextNodeName = next(NodeName)
| extend isFirst= (NodeName != prevNodeName)
| extend isLast=(NodeName != nextNodeName)
| where isFirst == true or isLast == true
| extend EndTime=next(TIMESTAMP)
```

```
| extend StartTime=TIMESTAMP
| where isFirst ==true
| extend codeversion=extract("^(.*)-WASD", 1, code_package_version)
| extend PrimaryNodeName=NodeName, PrimaryNodeFirstAppearTime=StartTime,
PrimaryNodeLastAppearTime=EndTime
| project ClusterName,  PrimaryNodeName, AppName, PrimaryNodeFirstAppearTime,
PrimaryNodeLastAppearTime,  codeversion )
|union
(MonDmDbHadrReplicaStates
| where LogicalServerName =~ "{LogicalServerName}"  and  logical_database_name  =~
dbNameForHadrReplicaStates
| where AppName contains "b-"
| where is_primary_replica == 1
| order by TIMESTAMP asc nulls first
| serialize
| extend prevNodeName=prev(NodeName)
| extend nextNodeName = next(NodeName)
| extend isFirst= (NodeName != prevNodeName)
| extend isLast=(NodeName != nextNodeName)
| where isFirst == true or isLast == true
| extend EndTime=next(TIMESTAMP)
| extend StartTime=TIMESTAMP
| where isFirst ==true
| extend codeversion=extract("^(.*)-WASD", 1, code_package_version)
| extend PrimaryNodeName=NodeName, PrimaryNodeFirstAppearTime=StartTime,
PrimaryNodeLastAppearTime=EndTime
| project ClusterName,  PrimaryNodeName, AppName, PrimaryNodeFirstAppearTime,
PrimaryNodeLastAppearTime,  codeversion );
let AllNodes=
PrimaryNodes
| join kind= leftouter (
   MonDmDbHadrReplicaStates
   | where LogicalServerName =~ "{LogicalServerName}"  and  logical_database_name  =~
dbNameForHadrReplicaStates
   | extend c1=""
   | where is_primary_replica == 0
   | extend SecondaryNodeName=NodeName
   | project TIMESTAMP, SecondaryNodeName , c1 , AppName
) on AppName
| where (TIMESTAMP >= PrimaryNodeFirstAppearTime and TIMESTAMP <= PrimaryNodeLastAppearTime and
PrimaryNodeName != SecondaryNodeName) or isnull(SecondaryNodeName) == true or SecondaryNodeName
== ""
| extend fake="fake"
| summarize  SecondaryNodeNames=makeset(SecondaryNodeName) by bin(PrimaryNodeFirstAppearTime, 1s),
bin(PrimaryNodeLastAppearTime,1s), ClusterName, AppName, codeversion, PrimaryNodeName;
AllNodes
| order by PrimaryNodeFirstAppearTime asc nulls last , PrimaryNodeLastAppearTime asc nulls last
```

```
        // C.02
// Resource Caps such as # of cores, memory, IO etc
MonDmDbResourceGovernance
| where AppName =~ "{AppName}"
| extend MaxCPUCores = case ( primary_bucket_fill_rate_cpu == 0,  (primary_group_max_cpu*min_cores)/100 ,
primary_bucket_fill_rate_cpu/1000.0)
| extend MaxMemoryMB = case (min_memory==0, max_db_memory/1024.0, min_memory/1024.0)
| extend MaxMemoryGB = round(MaxMemoryMB /1024.0,2)
| extend MaxIOPS = primary_group_max_io
| extend WorkersLimit = primary_group_max_workers
| extend SessionsLimit=max_sessions
| extend MaxDbDizeGb = max_db_max_size_in_mb/1024
| extend MaxLogKbps = primary_max_log_rate/1024.0
//| distinct slo_name,  MaxCPUCores,MaxMemoryGB, MaxIOPS, MaxLogKbps,MaxDbDizeGb, WorkersLimit,
SessionsLimit // , LogicalServerName
| summarize StartTime=min(TIMESTAMP), EndTime=max(TIMESTAMP)
    by LogicalServerName, AppName, slo_name,  MaxCPUCores,MaxMemoryGB, MaxIOPS,
MaxLogKbps,MaxDbDizeGb, WorkersLimit, SessionsLimit
| order by StartTime asc
| join kind= leftouter (
   MonDmRealTimeResourceStats
| where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}" and
AppName =~ "{AppName}"
| filter replica_type == 0
| summarize arg_min(TIMESTAMP, *) by AppName, database_name
) on AppName
| project StartTime, EndTime, LogicalServerName, AppName, database_name, slo_name,
MaxCPUCores,MaxMemoryGB, MaxIOPS, MaxLogKbps,MaxDbDizeGb, WorkersLimit, SessionsLimit,
cpu_cap_in_sec


// C.04
// SLO Change history
MonAnalyticsDBSnapshot
| where logical_server_name =~ "{LogicalServerName}"  and   logical_database_name  =~ "
{LogicalDatabaseName}"
| summarize min(start_utc_date), max(end_utc_date) by sql_instance_name, AppName, service_level_objective ,
physical_database_id, fabric_partition_id, logical_database_id
| project min_start_utc_date, max_end_utc_date, sql_instance_name, service_level_objective,
physical_database_id,logical_database_id,fabric_partition_id
| order by min_start_utc_date asc nulls last

// C.05.A
// Health of replicas
MonDmDbHadrReplicaStates
| where LogicalServerName =~ "{LogicalServerName}"  and AppName =~ "{AppName}"  //and
logical_database_name  =~ "{LogicalDatabaseName}"
| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where synchronization_state_desc == "NOT SYNCHRONIZING" or synchronization_health_desc ==
```

```
"NOT_HEALTHY"
| project TIMESTAMP, NodeName , synchronization_state_desc , synchronization_health_desc
| order by TIMESTAMP asc nulls last
// C.05
// Failover history by WinFab
MonFabricApi
| filter LogicalServerName =~ "{LogicalServerName}" and AppName =~ "{AppName}"
| filter event in ("hadr_fabric_api_replicator_begin_change_role", "hadr_fabric_api_replica_end_change_role")
| where TIMESTAMP >datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| filter new_role_desc == "PRIMARY"
| project originalEventTimestamp,AppName, event , NodeName , hotpatch_data_package_version,
current_state_desc ,new_role , new_role_desc, logical_database_id, physical_database_id, partition_id,
code_package_version, result_desc
| order by originalEventTimestamp asc nulls last

// C.06
// Failover reasons from XTS
// examples:       AppMemoryUsageMB, memory usage PLB
WinFabLogs
|where EventType == "Operation" and TaskName == "CRM" | where Id == tolower("{PartitionID}")
| extend decisionId = extract("DecisionId: ([a-z0-9-]+)", 1, Text)
| project decisionId
| summarize count() by decisionId
| join kind = inner (
    WinFabLogs | where EventType == "Decision" and TaskName == "CRM"  and TIMESTAMP
>datetime({StartTime}) and TIMESTAMP < datetime({EndTime}) //  and ClusterName =="tr6.centralus1-
a.worker.database.windows.net"
    | extend decisionId = Id
    | summarize min(ETWTimestamp), min(Text) by decisionId
) on decisionId
| project PreciseTimeStamp = min_ETWTimestamp, decisionId , Text = min_Text
| order by PreciseTimeStamp asc nulls last
// C.07 MAXDOP
// MAXDOP, 0 means MAXDOP is not set. MAXDOP=# processors
MonDatabaseMetadata
| where LogicalServerName=~ "{LogicalServerName}" and AppName=~"{AppName}" //logical_db_name =~ "
{LogicalDatabaseName}"
| where table_name== "sysobjvalues" and valclass ==7 and objid == 1032 //DB_CONFIG_MAX_DOP
| project TIMESTAMP, MAXDOP=value


// C.08 Database Options
//database options
MonDatabaseMetadata
//| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}"   and AppName =~ "{AppName}" and logical_db_name
=~ "{LogicalDatabaseName}"
| where table_name== "sysdbreg"
| extend is_parameterization_forced =  (binary_and(status2, 0x08000000) == 0x08000000)
```

```
| extend is_auto_create_stats_on = (binary_and(status2, 0x1000000) == 0x1000000)
| extend is_auto_create_stats_incremental_on = (binary_and(status2, 0x00400000) == 0x00400000)
| extend is_auto_update_stats_async_on = (binary_and(status2, 0x80000000) == 0x80000000)
| extend is_auto_update_stats_async_on = (binary_and(status2, 0x80000000) == 0x80000000)
| extend is_query_store_on=  (binary_and(status2, 0x00000010) == 0x00000010)   //DBR_QDSENABLED
| extend is_auto_update_stats_on= iif((status2/1073741824) % 2 == 0, false, true)
| project TIMESTAMP,  LogicalServerName, logical_db_name,  cmptlevel, is_query_store_on,
is_auto_create_stats_on, is_auto_update_stats_on, is_parameterization_forced, is_auto_update_stats_async_on,
is_auto_create_stats_incremental_on
| top 1 by TIMESTAMP desc nulls last

// C.09
 //table size
let myAppName="{AppName}";
let PartitionStats=materialize(MonWiDmDbPartitionStats
| where AppName contains myAppName and logical_database_name !='master' and index_id in (0,1)
| summarize used_page_count=max(used_page_count), row_count=max(row_count) by database_id,
logical_database_name, object_id, index_id, partition_number, data_date=bin(TIMESTAMP, 1d));
let FilteredResults=materialize(MonDatabaseMetadata
| where AppName contains myAppName and logical_db_name != 'master'
| where (table_name=='sysclsobjs' and class==50) or (table_name=='sysschobjs' and ['type']=='U ') or
(table_name=='sysidxstats' and indid in (0,1))
| project TIMESTAMP, table_name, class, ['type'], id, name, nsid, indid);
let schemas=FilteredResults
| where (table_name=='sysclsobjs' and class==50)
| summarize by schema_id=id, schema_name=tolower(name);
let tables=FilteredResults
| where (table_name=='sysschobjs' and ['type']=='U ')
| summarize by schema_id=nsid, object_id=id, table_name=name;
let indexes=FilteredResults
| where (table_name=='sysidxstats' and indid in (0,1))
| extend index_type_desc=iff(['type']==0, 'HEAP', iff(['type']==1, 'CLUSTERED', iff(['type']==5, 'CCI',
tostring(['type']))))
| summarize by object_id=id,index_id=indid,index_type=type,index_type_desc;
tables
| join kind=inner (schemas) on schema_id
| join kind=inner (indexes) on object_id
| join kind=inner (PartitionStats) on object_id
| project database_id, schema_id,  object_id,  table_type=index_type_desc, partition_number, used_page_count,
row_count, size_kb=used_page_count*8, data_date
//| project database_id, logical_database_name, schema_id, schema_name, object_id, table_name,
table_type=index_type_desc, partition_number, used_page_count, row_count, size_kb=used_page_count*8,
data_date
| summarize argmax(data_date, *) by object_id,partition_number
| top 100 by max_data_date_row_count desc
| project object_id, partition_number, data_date=max_data_date,row_count=max_data_date_row_count,
table_type=max_data_date_table_type
```

```
// C.10
// trace flags
MonNodeTraceETW
| where Message contains 'trace'
| where Message contains "{AppName}"
| where TIMESTAMP > ago(1d)

//*****************************************************************************************************************
// Node wide info
//*****************************************************************************************************************
// N.01
// Perfmon Counter oneminute for CPU and DISK
// TOP N in each category
// watch out
//    1. total CPU %
//    2. individual CPU % pegged to 100% is not good either
MonCounterOneMinute
| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where NodeName =~ "{NodeName}" and ClusterName =~ "{ClusterName}"
| extend Category = case(CounterName =~  @"\Memory\Available MBytes", "Memory",
   strlen(extract(@"\\Processor\([0-9]{1,2}\)\\% (Privileged Time|Processor Time)", 0, CounterName,
typeof(string))) >0, "Processor",
   strlen(extract(@"\\Processor\(_Total\)\\% (Processor Time|Privileged Time)", 0, CounterName, typeof(string)))
>0, "Total Processor",
   strlen(extract(@"\\LogicalDisk\([A-Z]:\)\\Avg. Disk sec/(Read|Write)", 0, CounterName, typeof(string))) >0,
"Disk", "Other")
| where Category != "Other"
| top-nested of bin(TIMESTAMP, 5min) by avg(CounterValue), top-nested of Category by avg(CounterValue),
top-nested 5 of CounterName by Avg_CounterValue=avg(CounterValue) desc
| sort by TIMESTAMP asc nulls last
| project TIMESTAMP, CounterName , Avg_CounterValue
| render timechart


// N.02
// Perfmon Counter oneminute for available MB memory
MonCounterOneMinute
| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where NodeName =~ "{NodeName}" and ClusterName =~ "{ClusterName}"
| where CounterName =~  @"\Memory\Available MBytes"
| summarize avg(CounterValue) by bin(TIMESTAMP, 5min),  CounterName
| render timechart


// N.03
// Node System Event log error summary
MonSystemEventLogErrors
| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where NodeName =~ "{NodeName}" and ClusterName =~ "{ClusterName}"
```

```
| summarize Count=count(), min(TIMESTAMP), max(TIMESTAMP),   min(EventDescription)by EventID
| order by Count desc nulls last


// N.04
// Node Application log error summary
MonAppEventLogErrors
| where TIMESTAMP > datetime({StartTime}) and TIMESTAMP < datetime({EndTime})
| where NodeName =~ "{NodeName}" and ClusterName =~ "{ClusterName}"
| summarize Count=count(), min(TIMESTAMP), max(TIMESTAMP),   min(EventDescription)by EventID
| order by Count desc nulls last
```

**How good have you found this content?**