Docker Performance Stats Monitoring Process

What does it do?

In a basic overview, this Monitoring process gathers all of the docker container stats to be looked at later. These include: Access Date, Container ID, Container Name, CPU percent, Memory Usage in MB, Net input/output in MB, Block input/output in MB. Next, it places the stats into a txt file and then moves it over to the d1vdbopsdb7700 SQL server where the data can be queried when needed. Additionally, it aggregates the data older than 1 week and places it into an aggregate table while clearing out the main table. All of these processes are scheduled through windows task scheduler and the SQL server agent

How is this done?

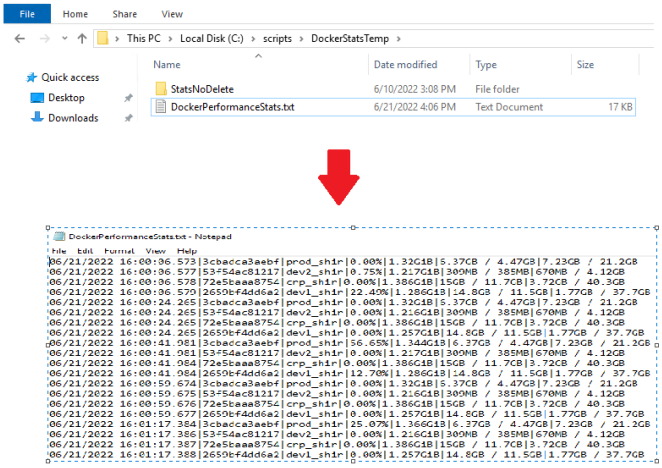
PowerShell

1. Script1: "C:\scripts\removeandadddockerstats.ps1"
    a. checks to see if a text file named "DockerPerformanceStats" already exists in the path C:\scripts\DockerStatsTemp\DockerPerformanceStats.txt
        i. ^^if so it deletes it

1. **Delete DockerPerformanceStats**

```
$FileName = "C:\scripts\DockerStatsTemp\DockerPerformanceStats.txt"
if (Test-Path $FileName) {
   Remove-Item $FileName -verbose
}
```

ii. ^^ if not, reads the statistics to a new txt document "DockerPerformanceStats" and adds a new 4 lines of data every 15 seconds for 14 minutes. This will be read by the SQL proc when the bulk insert is performed. This file can be found on the C drive of the D1VAZINT3000 computer in C:\scripts\DockerStatsTemp. It does it for 14 minutes to give it time to load into the table before this proc is executed again and the file gets deleted.



1.

2. **Create and populate DockerPerformanceStats**

```
$timeout = new-timespan -minutes 14
$sw = [diagnostics.stopwatch]::StartNew()

while ($sw.elapsed -lt $timeout)
{
    docker stats --no-stream --format "{{.Container}}|{{.Name}}|{{.CPUPerc}}|{{.
MemUsage}}|{{.NetIO}}|{{.BlockIO}}" | %{ "{0:MM/dd/yyyy HH:mm:ss.fff}|{1}" -f (Get-
Date), $_} >> C:\scripts\DockerStatsTemp\DockerPerformanceStats.txt
    Start-Sleep -Seconds 15

}
```

iii. Next a copy of the file is made with the date and time added onto the end of the file name and sent to an archive folder. This is stored on the shared folder location: \\d1vdbopsdb7700\DockerPerformance\Archive\

1. The archived txt files over 7 days old will be deleted. It is just there to give us enough time to find and fix an error when trying to run the task.

iv. Finally, the txt file is copied over to the SQLbox, that way the bulk load can be performed

SQL - "dockerperf" database

1. Two main tables being utilized
    a. [dbo].[PerformanceStats]
        i. Table that DockerPerformanceStats.txt gets loaded into
    b. [dbo].[AggrPerformanceStats]
        i. Aggregated data from the PerformanceStats table that are older than 7 days old
2. Stored procs being used
    a. [dbo].[LoadPerformanceStats]
        i. creates a temp table called #massagetable
        ii. preforms the bulk insert from DockerPerformanceStats.txt into the massage table
            1. after the data is in the massage table, it is taken and manipulated in a way where the data is easier to read, and converted to datatypes we can do math with.
            2. The insert statement gets tweaked whenever this is an error running the scheduled insert. You just go look at the archive folder, find the txt document from this insert, and then look through it to find where it would get caught up.

a. **Bulk Insert**

```
insert into PerformanceStats
(
        AccessDate,
        ContainerID,
        ContainerName,
        CPUPercent,
        MemoryMB,
        NetIO,
        NetInputMB,
        NetOutputMB,
        BlockIO,
        BlockInputMB,
        BlockOutputMB
)
select
        AccessDate,
        ContainerID,
        ContainerName,
        [CpuPercent] = cast(replace(CPU, '%', '') as decimal(6,2)),
        [MemoryMB] =
                case
                        when patindex('%Gib%', Memory) > 0
                                then cast(cast(replace(Memory, 'Gib', '') as
decimal(10,3)) * 1024 as decimal(10,3))
                        when patindex('%Mib%', Memory) > 0
                                then cast(replace(Memory, 'Mib', '') as decimal
(10,3))
                        else
                                cast(cast(replace(Memory, 'B', '') as decimal
(10,3)) / 1024 as decimal(10,3))
                end,
        NetIO,
        [NetInputMB] =
                case
                        when PATINDEX( '%Gb%', SUBSTRING(NetIO, 1, patindex('%
/ %', NetIO)-1)) > 0
                        then (cast(SUBSTRING(NetIO, 1, patindex('% / %', NetIO)
-3)as decimal(10,2) ) * 1024)
                        else SUBSTRING(NetIO, 1, patindex('% / %', NetIO)-3)
                end,
        [NetOutputMB] =
                case
                        when PATINDEX('%GB%', substring(NetIO, (patindex('% /
%', NetIO)+3), (len(NetIO)) - (patindex('% / %', NetIO)))) > 0
                        then cast(substring(NetIo, PATINDEX('% / %', NetIO)+2,
len(NetIO) - (PATINDEX('% / %', NetIO)+2) - 1) as decimal(10,2)) * 1024
                        else cast(substring(NetIO, patindex('% / %', NetIO)+2,
len(NetIO) - (patindex('% / %', NetIO)+2) - 1) as decimal(10,2))
                end,
        BlockIO,
        [BlockInputMB] =
                case
                        when PATINDEX( '%Gb', SUBSTRING(BlockIO, 1, patindex
('% / %', NetIO)-1)) > 0
                        then (cast(SUBSTRING(BlockIO, 1, patindex('% / %',
BlockIO)-3)as decimal(10,2) ) * 1024)
                        else SUBSTRING(BlockIO, 1, patindex('% / %', BlockIO)
-3)
                end,
        [BlockOutputMB] =
                case
                        when PATINDEX('%GB%', substring(BlockIO, (patindex('%
/ %', BlockIO)+3), (len(BlockIO)) - (patindex('% / %', BlockIO)))) > 0
                        then cast(substring(BlockIO, PATINDEX('% / %', BlockIO)
+2, len(BlockIO) - (PATINDEX('% / %', BlockIO)+2) - 1) as decimal(10,2)) * 1024
                        else cast(substring(BlockIO, patindex('% / %', BlockIO)
+2, len(BlockIO) - (patindex('% / %', BlockIO)+2) - 1) as decimal(10,2))
                end
from #MassageTable
```

b. [dbo].[AggrPerformanceStatsLoad]
    i. After the stats have been loaded into the performance stats table. At some point we have to clean it up and minimize the size of the table that way it isn't growing forever. This is accomplished by taking the data that is older than 7 days and aggregating it all. This means taking the avg, min, max, sum, and count of all of these values by hour, per container, and putting them in another container.
    ii. the proc runs on a try catch with a transaction inside of it, it tries to aggregate and insert all of the data older than 7 days, and if there is an error then it moves into the catch and that transaction gets rolled back.

1. **Aggregate insert**

```
begin try
begin transaction
insert into AggrPerformanceStats
select
        concat(substring(convert(varchar(100), accessdate, 121), 1, 13), ':00'),
        ContainerID,
        ContainerName,
        count(AccessDate),
        avg(CPUPercent),
        Min(CPUPercent),
        Max(CPUPercent),
        Avg(MemoryMB),
        Min(MemoryMB),
        Max(MemoryMB),
        Avg(NetInputMB),
        Min(NetInputMB),
        Max(netInputMB),
        Sum(NetInputMB),
        Avg(NetOutputMB),
        Min(NetOutputMB),
        Max(NetOutputMB),
        Sum(NetOutputMB),
        Avg(BlockInputMB),
        Min(BlockInputMB),
        Max(BlockInputMB),
        sum(BlockInputMB),
        Avg(BlockOutputMB),
        Min(BlockOutputMB),
        Max(BlockoutputMB),
        Sum(BlockOutputMB)
from PerformanceStats
where accessdate < @cutoffdate
group by
        concat(substring(convert(varchar(100), accessdate, 121), 1, 13), ':00') ,
        containerID,
        containerName
```

Scheduled Tasks and Jobs

The 3 jobs\tasks that make all of the things in SQL work, run the above scripts and procedures to make sure the Docker Stats are being monitored and recorded throughout the entire day without having to manually go run all of these things

1. DockerPerformanceTask - Task Scheduler
    a. executes the PowerShell script: removeandadddocekerstats.ps1
    b. runs every 15 min starting at 12:00 am
    c. owner is AppOpsAutomation
2. DBOps Docker Performance Stats Load - SQL Server Agent
    a. executes the stored procedure : LoadPerformanceStats
    b. occurs every 15 minutes starting at 12:05 am
    c. owner is sa
3. DBOps Docker Performance Stats Aggregation - SQL Server Agent
    a. executes the stored procedure: AggrPerformanceStatsLoad
    b. occurs once a day at 12:10 am
    c. owner is sa