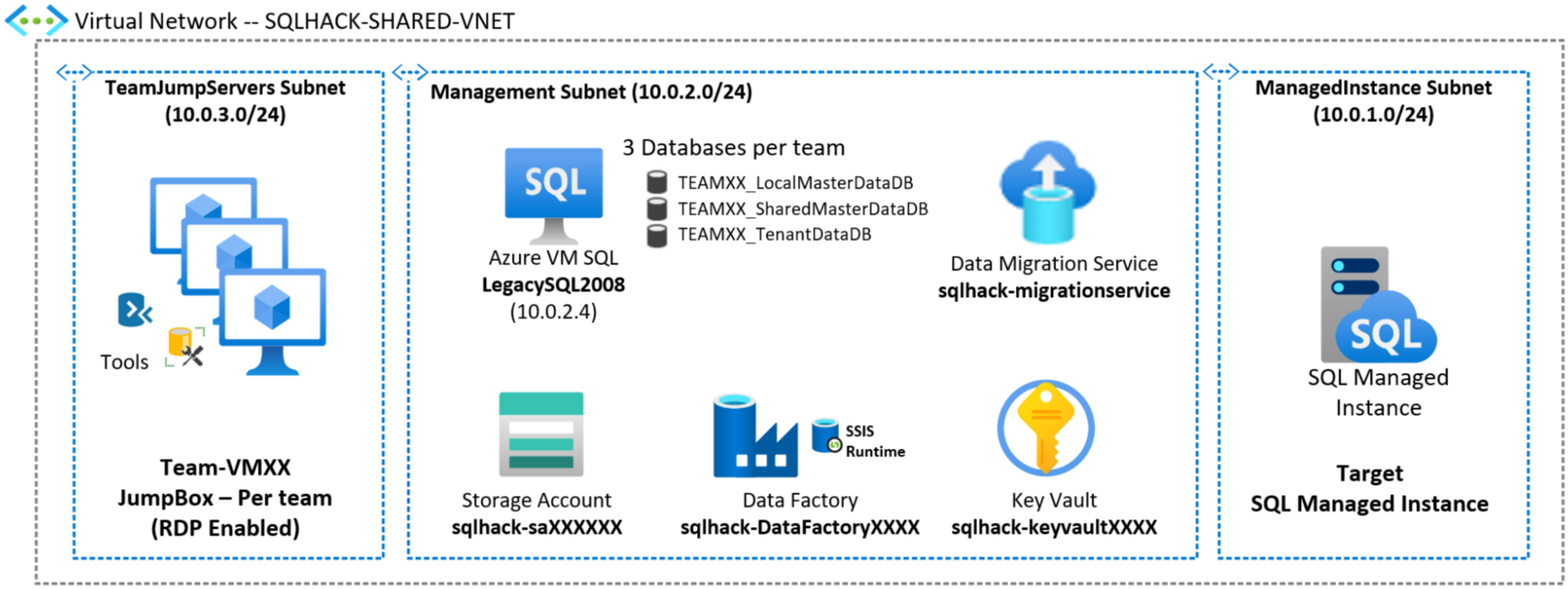
SQL Modernisation Hack
Database Administering and Monitoring Labs Step-by-step

V4.0

Contents

Migration architecture and Azure components



SQLHACK-SHARED-VNET Single Virtual Network containing all workshop resources		
TeamJumpServers Subnet Each team is assigned a Win10 VM that mimics their company desktop	**Management Subnet** Several machines and services are already deployed within a dedicated subnet within the Virtual Network	**ManagedInstance Subnet** The Azure SQL Managed Instance has been deployed into a dedicated Subnet

Lab Overview & Background

In this exercise you will explore how you can monitor and diagnose performance issues with Azure SQL Database Managed Instance using SQL Server Management Studio, DMVs (Dynamic Management Views) and Azure Portal tools.

Background

We have seen that the legacy application is a multi-tenant system with a collection of 3 databases supporting the transactional workload for each customer – these are the 3 databases we have modernised by migrating to the shared Azure SQL Managed Instance.

Apart from the tenant centric transactional databases there are 2 shared databases:

- **2008DW**: A centralised Data Warehouse database that combines data from the various tenant transactional databases and is used for aggregated reporting and analytics.
- **TenantCRM**: A centralised database that is used to manage all customer relationships and order processing.

Recently you have heard complaints from users that the TenantCRM system is running slowly. Recently changes were made to some of the stored procedures in the TenantCRM database and the App dev team believe these are the source of the problem. They've asked for your help to track down the performance issues and its root cause.

Let's follow the steps below to begin to troubleshoot this issue.

The exercises in this lab are conducted against the shared [TenantCRM] database

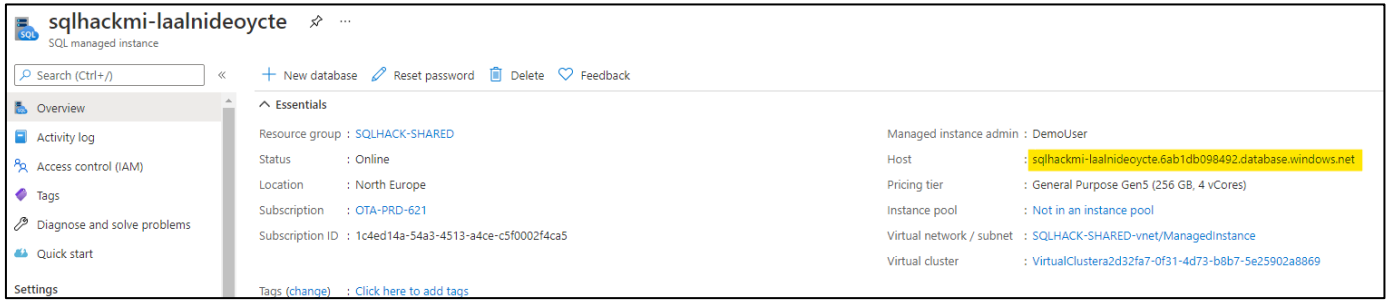
LAB 1: Identifying performance issues, their root causes and fixing the problem

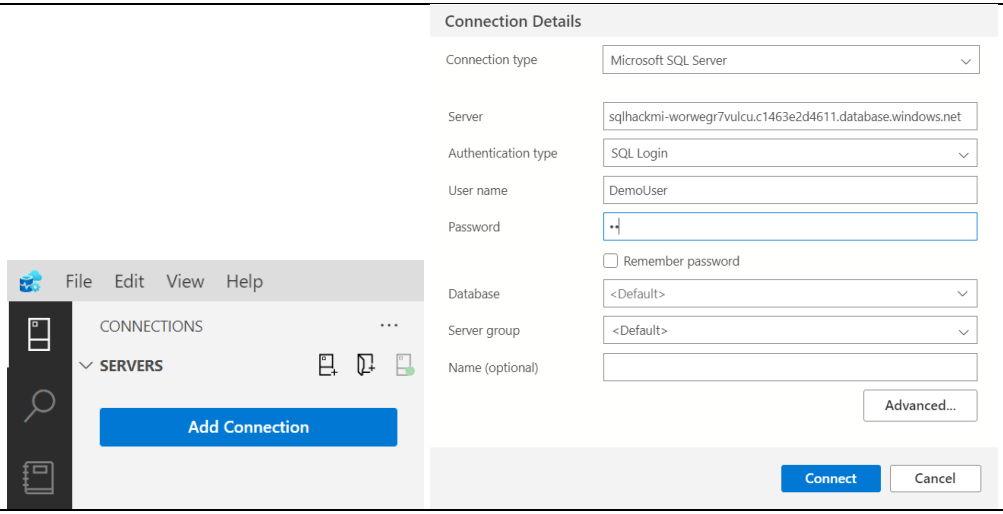
1. Using DMVs and the Query Store to identify performance bottlenecks.

Performance issues in SQL Server can be grouped into 1 of 2 high-level categories: *Running* or *Waiting*.

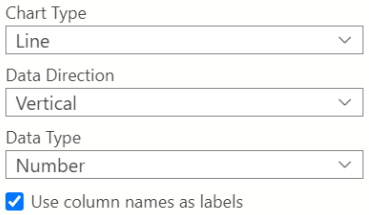
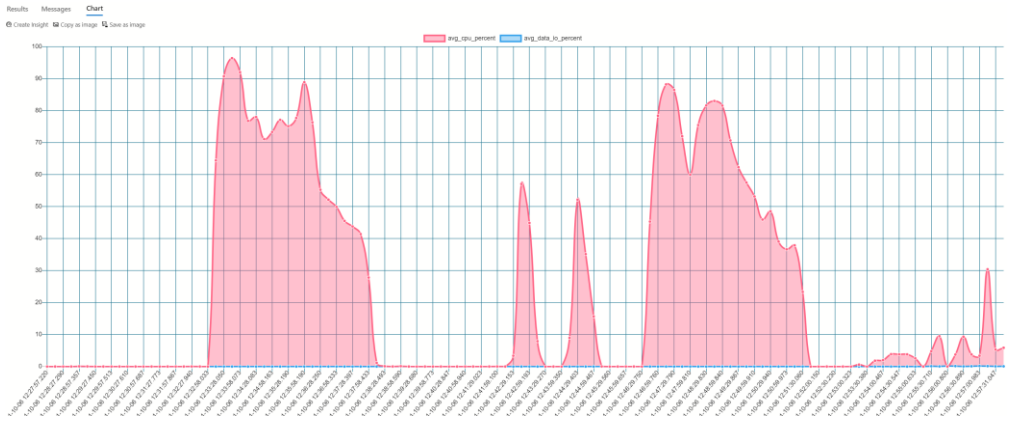
Running means a query has been allocated CPU resource and is actively processing but is taking a long time to complete. Waiting means a query is not actively being processed by the CPU but is waiting on another resource (such as memory, disk, network, etc) to pass data in so processing can continue.

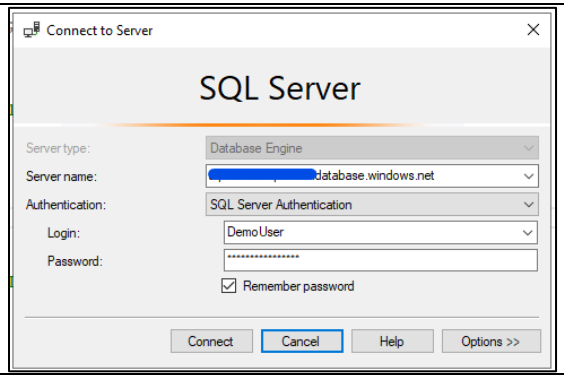
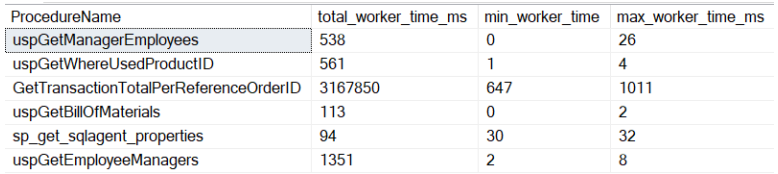
Finding the bottleneck will help us determine what category issue we have so we can progress to identifying the root cause. Let's see how to find the bottleneck using both SSMS and the Azure portal.

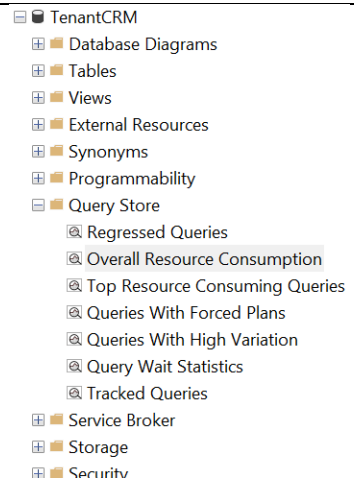
Instructions/Narrative	Screenshot	Notes
On your Win10 Team VM open the Azure Portal and navigate to the shared SQL Managed Instance. Scroll down the Overview screen copy the host name of the Managed Instance and save it somewhere handy.		

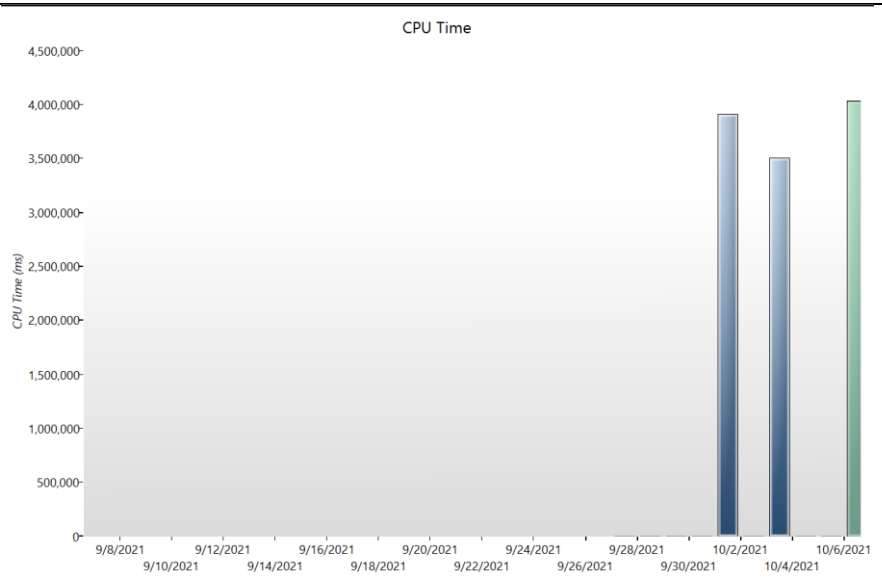
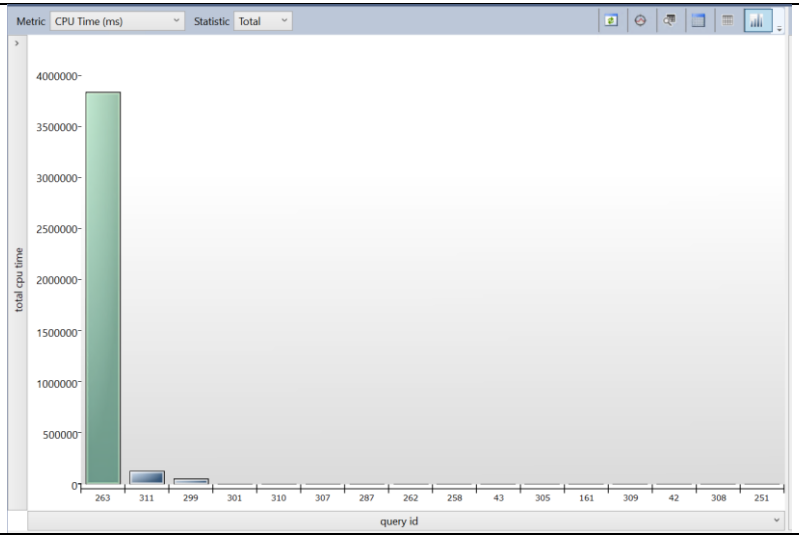
On your Team Win10 VM open **Azure Data Studio** and **Add Connection** using these details: **Server:** <just copied> **Authentication:** SQL Login **Login:** DemoUser **Pword:** Demo@pass1234567	

Narrative/Instructions	Screentshot	Notes
Open the SQL script: C:_SQLHACK_\LABS\02-Administering_Monitoring\Part_01_Monitoring_Lab_1.sql Run the **PART 1** query 1. Look at the results and note the [avg_cpu_percent] value is about 70% indicating that this database is consuming a lot of the hosts CPU resources	Results Messages 		end_time	avg_cpu_percent	avg_data_io_percent	avg_log_write_percent
---	---	---	---	---
1	2021-10-06 13:03:31.780	81.17	0.00	0.00
2	2021-10-06 13:03:16.770	77.06	0.00	0.00
3	2021-10-06 13:03:01.770	42.29	0.00	0.00
4	2021-10-06 13:02:46.747	0.00	0.00	0.00
5	2021-10-06 13:02:31.727	0.00	0.00	0.00
6	2021-10-06 13:02:16.680	0.00	0.00	0.00
7	2021-10-06 13:02:01.633	0.00	0.96	0.00
8	2021-10-06 13:01:46.600	0.00	0.00	0.00
9	2021-10-06 13:01:31.570	0.00	0.00	0.00		[sys].[dm_db_resource_stats] exposes a number of key database performance metrics. One metric every 15 seconds over the last hour

Narrative/Instructions	Screenshot	Notes
Let's try now query 2 to visualize graphically the resource consumption. **Execute** Query 2 and on the results, activate the **Chart view** On the Right  Settings 		If more than 1 hour of statistics is needed, use the DMV [sys].[server_resource_stats] up to 14 days

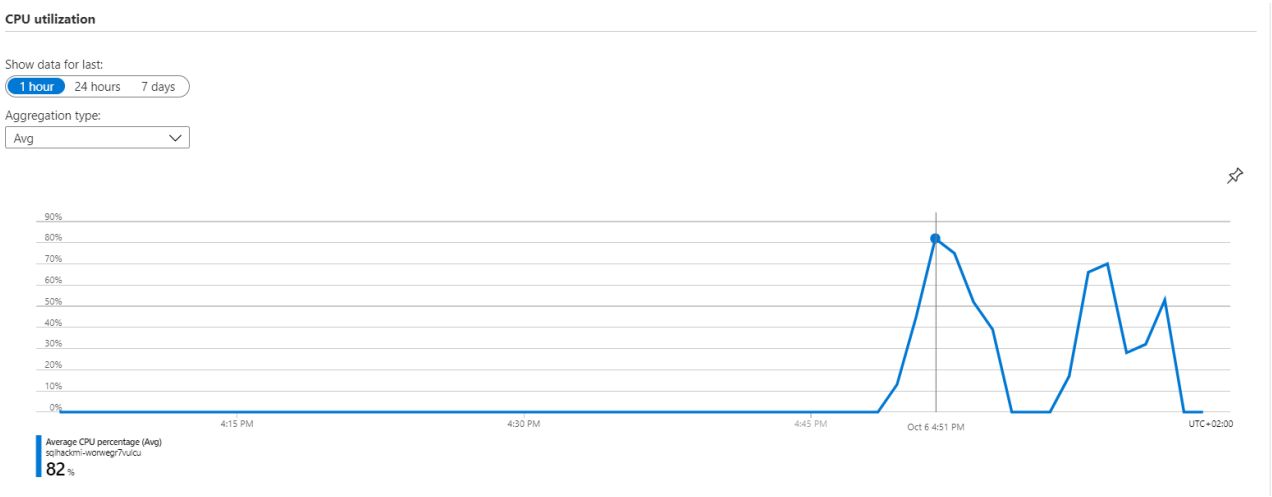
Instructions/Narrative	Screenshot	Notes
Open SQL Server Management Studio using the same details: **Server name:** SQL MI **Authentication:** SQL Server **Login:** DemoUser **Pword:** Demo@pass1234567		
Open the SQL script: C:_SQLHACK_\LABS\02-Administering_Monitoring\Part_02_Monitoring_Lab_1.sql **Run** the **PART 2** query 1. Identify the most consuming stored procedure based on the total_worker_time (CPU)		
Then to have further details on the most consuming stored procedure **Run** the **PART 2** query 2, replace the filter clause with the stored procedure name. What is the issue? How to optimise? **Tip:** Have a look to the execution plan	***CROSS APPLY and Table-valued Functions*** The PART 3 query joins the [sys].[dm_exec_requests] and [sys].[dm_exec_sql_text] DMVs to obtain long running batches and critically their offending SQL. Note that [dm_exec_sql_text] is actually a table-valued function (TVF) hence the use of the CROSS APPLY as TVFs can't be used in a normal join operation. The CROSS APPLY therefore produces the required inner-join between [dm_exec_requests] and [dm_exec_sql_text]. See APPLY documentation here: Using APPLY | Microsoft Docs and this excellent explanation on MSSQLTips: SQL Server CROSS APPLY and OUTER APPLY (mssqltips.com)	

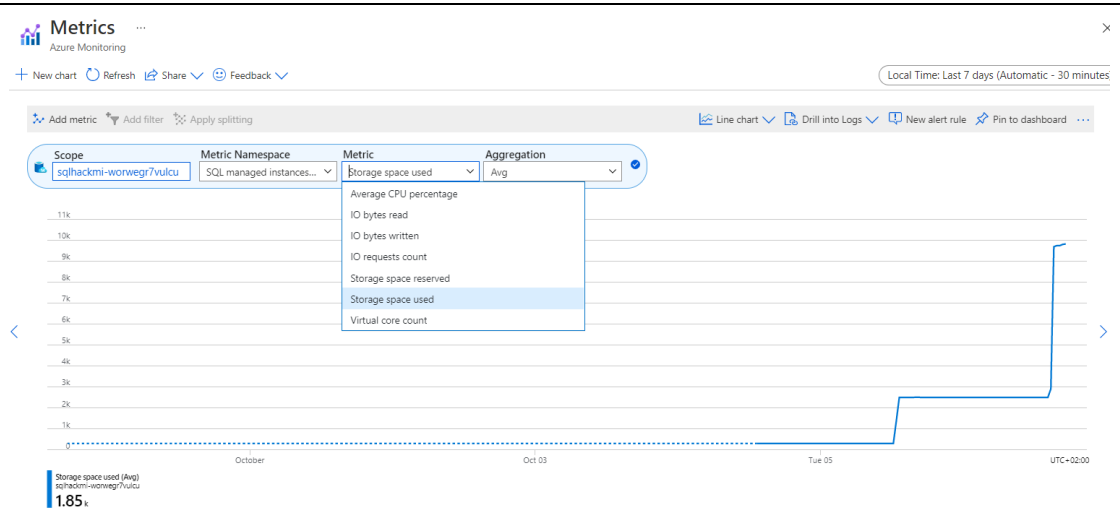
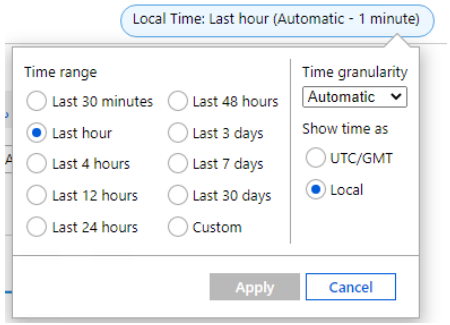
In SSMS expand the [TenantCRM] database then expand Query Store. Remember that the Query Store collects telemetry data every 15 seconds and persists it for about 1hr. Double click on **Overall Resource Consumption** report to open it.		

Narrative/Instructions	Screentshot	Notes
For further details CPU Time, double click on the current day bar		
You can identify the top CPU consuming queries. Click on bars for more details		

2. Using the Azure Portal to identify performance bottlenecks

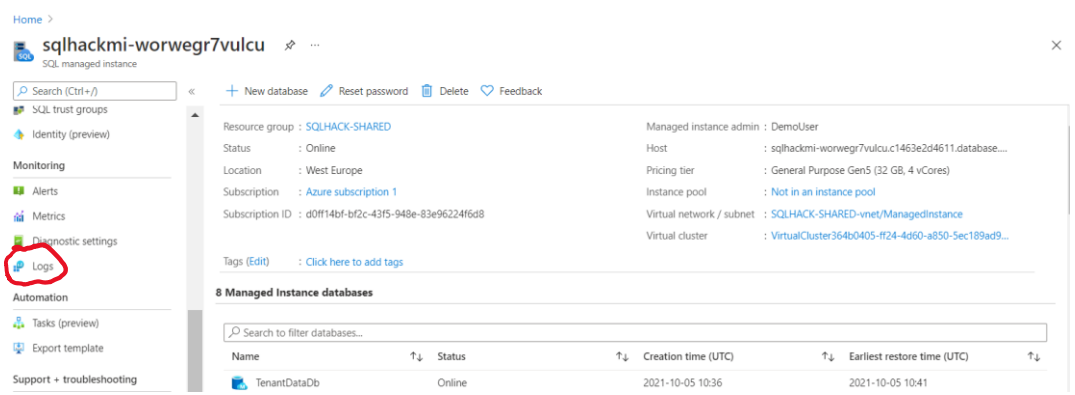
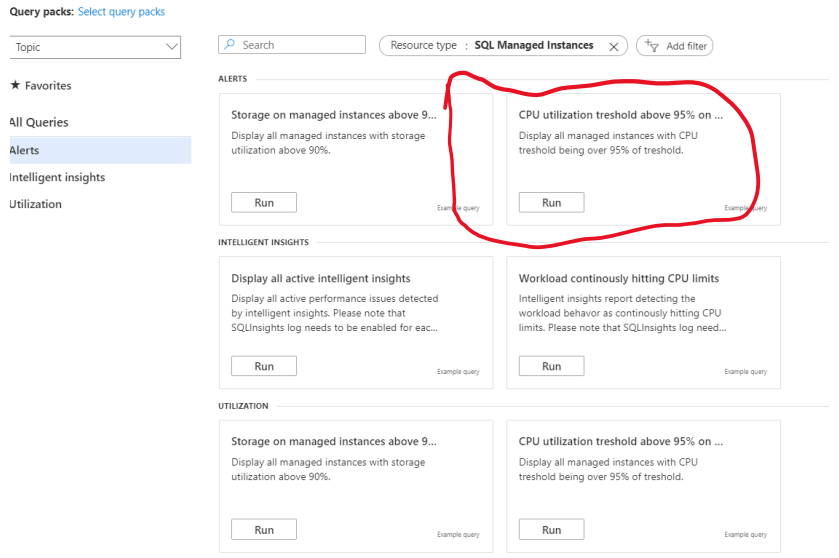
As we are using Azure SQL Managed Instance we can also use the richness of the Azure Portal to help us to troubleshoot performance issues. Let us see how this is done.

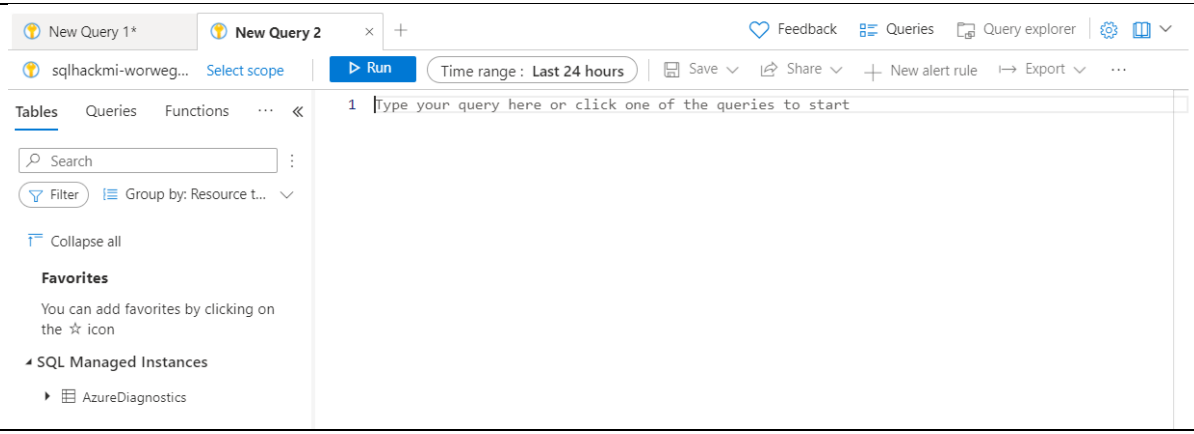
Instructions/Narrative	Screenshot	Notes
Open the Azure Portal (portal.azure.com) and open the shared SQL Managed Instance. Scroll down the **Overview** screen until you see the performance graph showing CPU utilization. Toggle the switches next to view CPU consumption over 1 hour, 24 hours and 7 days. Has the CPU usage pattern changed over time?	 CPU utilization Show data for last: 1 hour 24 hours 7 days Aggregation type: Avg 90% 80% 70% 60% 50% 40% 30% 20% 10% 0% 4:15 PM 4:30 PM 4:45 PM Oct 6 4:51 PM UTC+02:00 Average CPU percentage (Avg) 82 %	

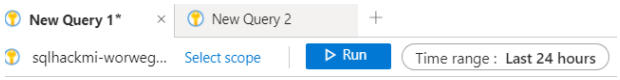
Instructions/Narrative	Screenshot	Notes
Click into the graph to go into Metrics chart screen. Here you have more flexibility over the data being reported and the format of the chart. Let's add IO Request Count the chart. Click **Add Metric** In the Metric drop down select **Storage Space Used**. .		Notice that from the chart screen you can alter the type of chart (Line, Area, Bar, Scatter to data Grid), set up alerts, and pin the chart to your Azure Portal dashboard.
Click **Local Time** on the top right of the chart to view the data for various time intervals.		

3. Using Log Analytics with KQL to visualize SQL MI and Database Stats

Now that we can use the global monitoring tool Log Analytics to identify which SQL Managed Instance .

Instructions/Narrative	Screenshot	Notes
On the Azure Portal, on the SQL Managed Instance Resource, **click on Logs** on the left Menu		
You will find several query templates You can test the **CPU utilization threshold template** And modify the parameters if needed.		

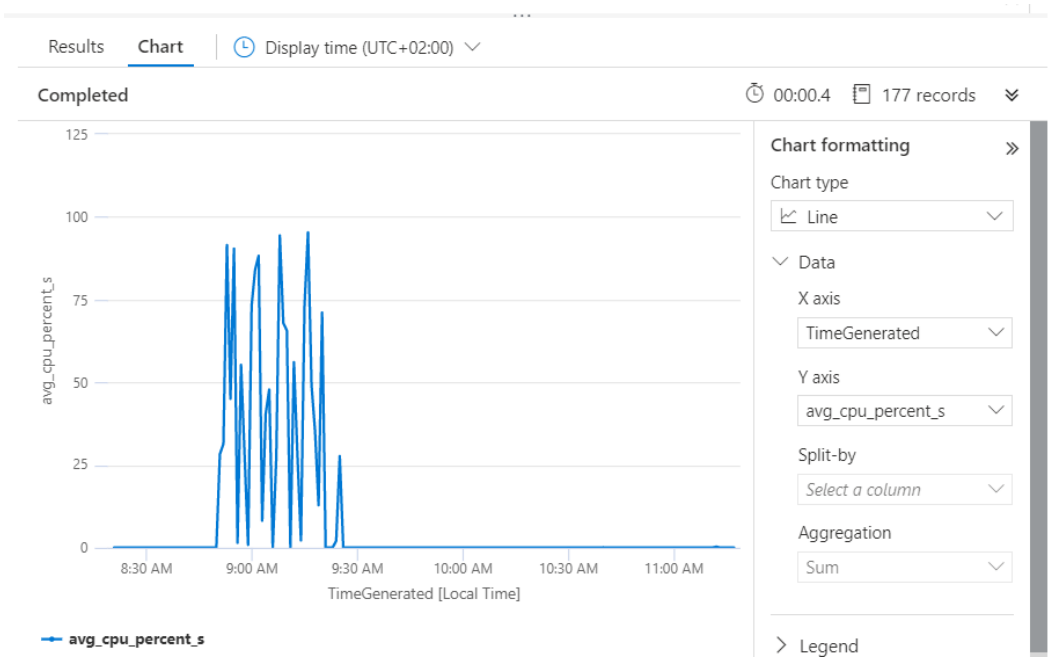
Instructions/Narrative	Screenshot	Notes
Create a new Query		

You can try the following queries. AzureDiagnostics is the table receiving the different diagnostics
 	
Modify the Time Range, By default 24hours
 
	//general queries
	// query 1 without filter
	AzureDiagnostics
	
	// query 2 SQL MI Resource usage
	AzureDiagnostics
	| where Resource == "SQLHACKMI-WORWEGR7VULCU" //replace with your SQL MI Instance Name
	| where Category == "ResourceUsageStats"
	
	// query 3 on database: waits Stats
	AzureDiagnostics
	| where Resource == "TENANTCRM"
	| where Category == "QueryStoreWaitStatistics"
	
	// query 4 on database: errors
	AzureDiagnostics
	| where Resource == "TENANTCRM"
	| where Category == "Errors"	

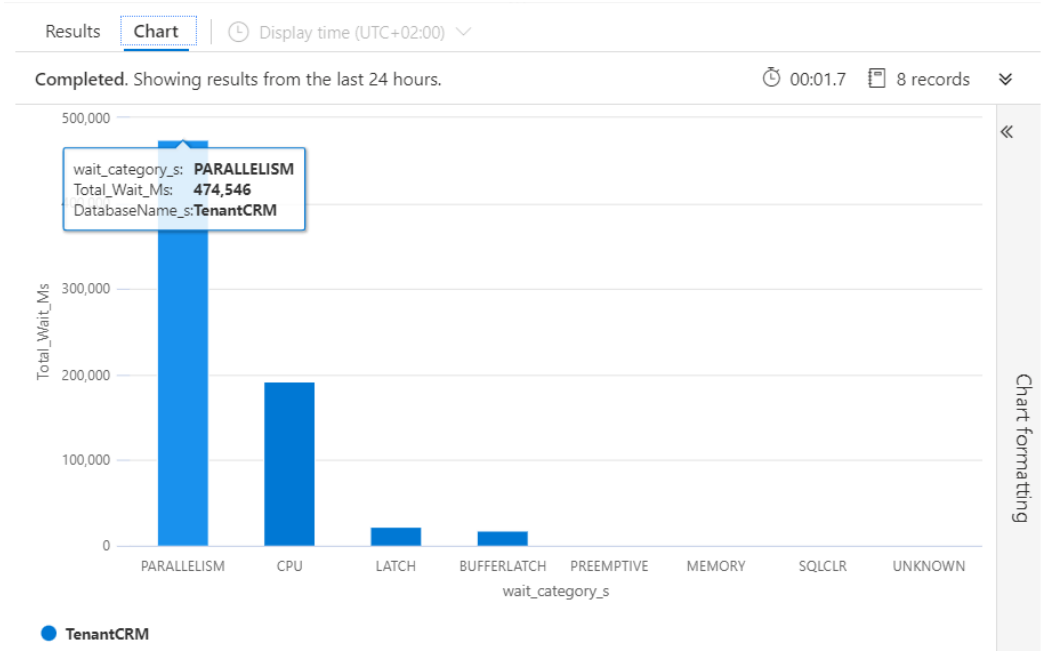
On the results for each of the queries, change the to **Chart**

The chart Formatting is on the right.

```
// more specific queries
//Query get CPU usage
AzureDiagnostics
| where Resource == "SQLHACKMI-WORWEGR7VULCU" and TimeGenerated > ago(2h) //replace with your SQL MI Instance Name
| where Category == "ResourceUsageStats"
| project TimeGenerated, todecimal(avg_cpu_percent_s)
// visualize the resulset as a chart
```



```
// Query get database Waits Stats
AzureDiagnostics
| where Category == "QueryStoreWaitStatistics"
| summarize Total_Wait_Ms = max(todouble(total_query_wait_time_ms_d)) by wait_category_s,DatabaseName_s
| order by Total_Wait_Ms desc
// visualize the resulset as a chart
```



We've now seen how the various performance monitoring and diagnosis tools – the Azure Portal, DMVs and the Query Store reports all revealed that the CPU was under pressure. The DMVs and Query Store reports also allowed us to drill-down to root cause and identify the worst offending query. In the real world you would now progress to tune the offending query to reduce the overall load on the Managed Instances CPUs and thereby improve the databases and environment performance.