AzureVPNWorkbook

Last updated by | M365 Administrator | Nov 12, 2021 at 5:56 PM GMT+1

Contents

Introduction

Azure VPN P2S is a VPN service running in Azure part of Azure Virtual WAN. To read about how to setup Azure Virtual WAN P2S VPN see https://docs.microsoft.com/azure/virtual-wan/monitor-virtual-wan

This section documents how to create an Azure Workbook that shows relevant data of for VPN clients connecting to Azure Virtual WAN P2S. Out of the box, the number of metrics supported by Azure Virtual WAN P2S is rather limited. What is available is documented here

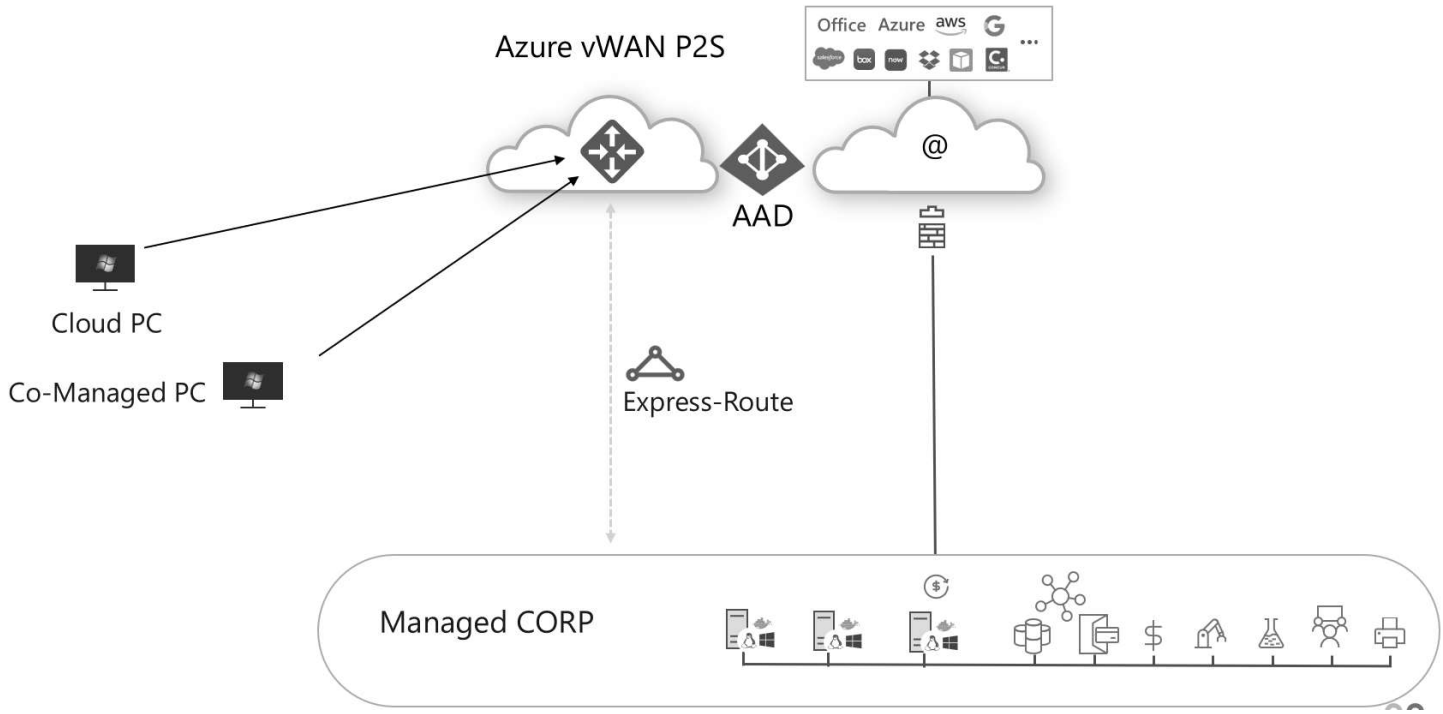
More and more enterprise customers, like <Customer>, are seeing an advantage in migrating from an on-premises VPN solution with multiple servers to manage over into using Azure Virtual WAN as a PaaS VPN solution managed by Microsoft. When combined with Microsoft always-on VPN and Azure VPN clients this works very nicely. This solution does assume that the <Customer> has a S2S VPN or Express Route to on-premises as clients wants to connect to on-premises resources.

Another assumption for this solution to be an optimal choice for the customer is that they dependant less on on-premises services as they migrate more services to Azure IaaS and/or PaaS or even SaaS solutions, thus minimizing the load on the connection to on-premises servers.

This solution follows the principle of Zero Trust where we want to migrate control and management plane to cloud services as much as possible as the assumption is that the cloud is more secure than the on-premises environment.

VPN Architecture

The architecture of the solution is shown below



The figure shows clients connecting to Azure vWAN P2S VPN as a PaaS service in Azure Cloud and from there connecting to on-premises ussing ExpressRoute. You can go to the Azure Portal and locate the P2S in a vWAN Hub, but you only get very few metrics about users connecting, so there is a need to enrich these data.

Workbook solution architecture

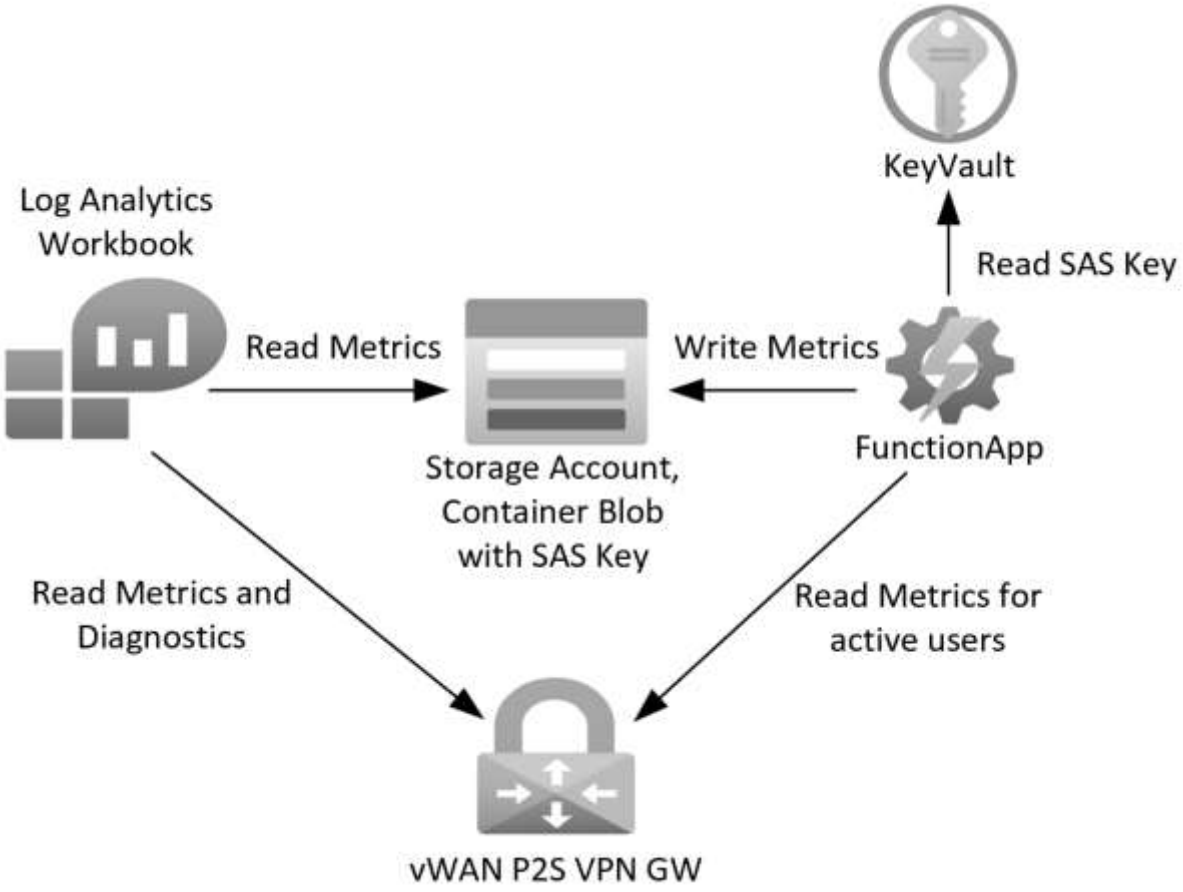
Input has been given to the Product group to get more detailed metrics about users connecting through VPN as well both active related to active sessions as well as hitoric data. This is being worked on with no ETA.

Hence we suggest a temporary solution that uses sources available today

- AzureDiagnostics: Enable P2S Debugging through Azure Monitor debug settings, GatewayDiagnosticLog, IKEDiagnosticLog, P2SDiagnosticLog, AllMetrics. Notice that some logs are very noisy and thereby rather costly in regards to Log Analytics cost, specially IKEDiagnostics
- Azure Metrics : Can be used directly from within the workbook, but the metrics available are limited
- Get-AzP2sVpnGatewayDetailedConnectionHealth which is a separate PowerShell command to get active sessions and this command only supports storing data in a storage account based on a SAS Key.

When you work with Azure Virtual WAN and look at metrics, it is most often done from within the context of an Azure workbook. You could also choose to extract data and use PowerBI which is another great tool to visualize data. In this case we choose to make a solution based on Azure Workbook and use what is already available and enrich it with more details, especially about active connections.

The figure below shows the involved components in the suggested soluion.



The VPN service is running in the Azure vWAN P2S VPN gateway and has associated metrics and debug settings that we can read and act on directly from within an Azure workbook. To get the extra information that the PowerShell command can provide, we choose to execute this command in an Azure FunctionApp and from there store the output it the required Azure storage account.

The output stored in the storage account is fetched from within the workbook by using a special function called "externaldata".

Below is a description of the various components.

Azure storage account

The Azure storage account has the following configuration settings

Storage account name: <storageaccountname>
StorageAccount Configuration:

- Secure transfer requred: Enabled

- Allow Blob public access: Disabled
- Allow storage account key access: Enabled
- Default to Azure Active Directory authorization in the Azure Portal: Enabled
- Minimum TLS version: Version 1.2
- Replication: Locally-redundant storage (LRS)
  Storage Container name: <storagecontainername>
  Authentication method: Azure AD User Account (or switch to access key, as this will also work)
  Blob in container: create/upload empty file <vpnstatfile.json> in container
  Generate SAS token and URL: (SASURI, to be saved in keyvault and used directly from within the workbook)
  Signing method: Account Key
- Key 1: Permissions
  - Read, Add, Create, Write
- Start : Choose start time
- Expiry: Choose expiry time

You can choose to create two SAS keys, one for read/write access from the Azure FunctionApp and one with read access used from the workbook, but for now, we use the same SAS key and restrict access to the workbook.

Azure FunctionApp

The FunctionApp has the following conguration settings

FunctionApp name: <functionappname>
Identity: system assigened: Status: On (we enable the managed/built-in service identity to be able to access Azure without having to use a username/password or service principal based on secrets). Note down the object id of this identity (to be used in permission assignment
)
Configuration:
Application settings

- resource group: <resourcegroupname>
- sasuri: @Microsoft.KeyVault(SecretUri=https://<keyvaultname>.vault.azure.net/secrets/sasuri/<version>) (update accordingly after keyvault is created.
- storageaccountname: <storageaccountname>
- storagecontainer: <storagecontainername>
- subscription: <subscriptionid>
- tenantname: <tenantname>
- vpngw: <vpngw> This name is something like <guid>-eastus-ps2-gw . You can get this from the vWAN HUB User VPN settings.

Function:

- FunctionName: <FunctionName>, choose for example a trigger type function to be executed each 5 minutes
- Code: see below

---

param($Timer)currentUTCtime = (Get-Date).ToUniversalTime()
if ($Timer.IsPastDue) {
Write-Host "PowerShell timer is running late!"
}
Write-Host "PowerShell timer trigger function ran! TIME: $currentUTCtime"tenantname =
$env:appsettingtenantnamesubscription = $env:appsettingsubscriptionresourceGroup =
$env:appsettingresourcegroupstorageAccountName =
$env:appsettingstorageaccountnamestorageContainer =
$env:appsettingstoragecontainervpnstatsfile= $env:appsettingvpnstatsfilevpngw =
$env:appsettingvpngwsasuri = $env:appsetting_sasuri
connect-azaccount -tenant $tenantname -identity -subscription $subscription
Get-AzP2sVpnGatewayDetailedConnectionHealth -name $vpngw -ResourceGroupName $resourceGroup -OutputBlobSasUrl $sasuri

---

For the get-AzP2sVpnGatewayDetailedConnectionHealth command to succeed, you need to have the right permissions to the information. This can be done by creating a custom Azure role. Go the Azure Portal to the resource group used and choose Access Control (IAM) and assign the folloing role/permissions
Custome Role Name: <custom role name like MicrosoftNetworkP2SGWReader>. We don't want to use built-in role like network-contributor as we only want read-access.

Type	Permissions	Description
Read	Get P2SVpnGateway	Gets a P2SVpnGateway.
Other	Gets a P2S Vpn Connection health for P2SVpnGateway	Gets a P2S Vpn Connection health for P2SVpnGateway
Other	Gets a P2S Vpn Connection health detailed for P2SVpnGateway	Gets a P2S Vpn Connection health detailed for P2SVpnGateway
Read	Gets P2S Vpn Gateway Log Definitions	Gets the events for P2S Vpn Gateway
Read	Get P2S Vpn Gateway Diagnostic Settings	Gets the P2S Vpn Gateway Diagnostic Settings
Read	Read P2S Vpn Gateway metric definitions	Gets the available metrics for P2S Vpn Gateway
Read	Get Virtual Wan P2SVpnServerConfiguration	Gets a virtual Wan P2SVpnServerConfiguration

Now assign the managed identity (objectid) this role.

Azure KeyVault

Create a KeyVault to keep the SAS key for Read/Write access to not expose it directly in the FunctionApp. Unfortunately we can't acceess this as a secret from the workbook, only from the FunctionApp.

KeyVault settings:
KeyVault Name: <keyvaultname>
Secrets:

- sasuri:
- secret value: <SASURI> (storage account SAS key)
  Enabled:Yes

New Access Policy:
Permission model: Vault access policy

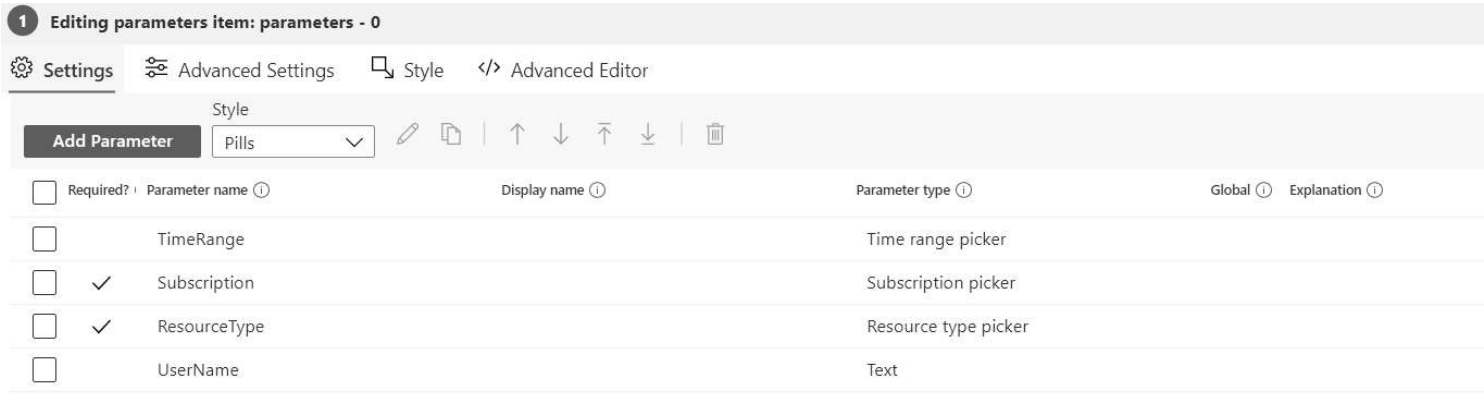
- Keys and secret management
- Key Permissions: Get, List
- Secret Permissions: Get, List
- Select Principal: <FunctionApp managed identity>

Azure workbook

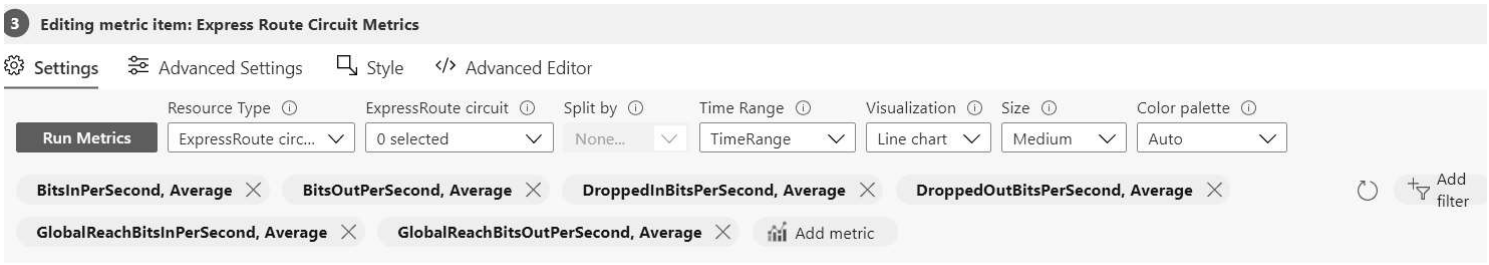
The Azure workbook is now ready to be created with a mix of built-in functionality and the addition that uses the solution above to give insights to active sessions with more details.

Create a workbook from Azure Monitor or from a Log Analytics workspace.
Give it a name: <workbook name>

P2S VPN Gateway Metrics

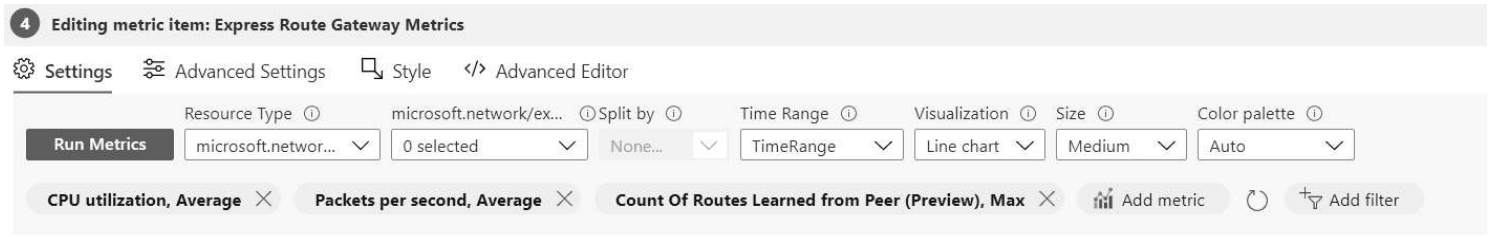


Express Route Circuit Metrics



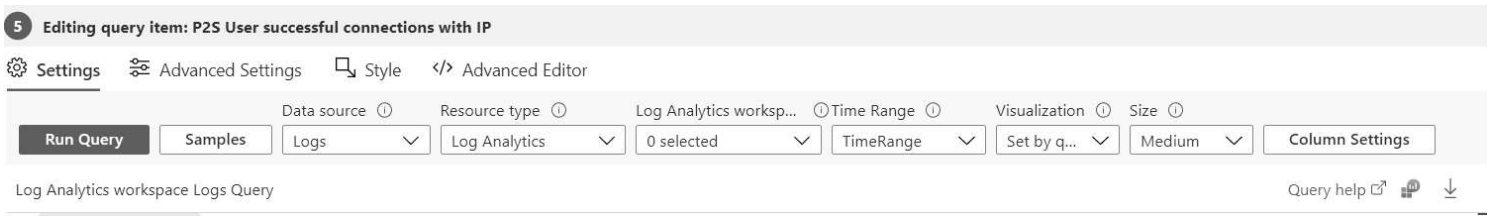
Express Route Circuit Metrics

Express Route Gateway Metrics



Express Route Gateway Metrics

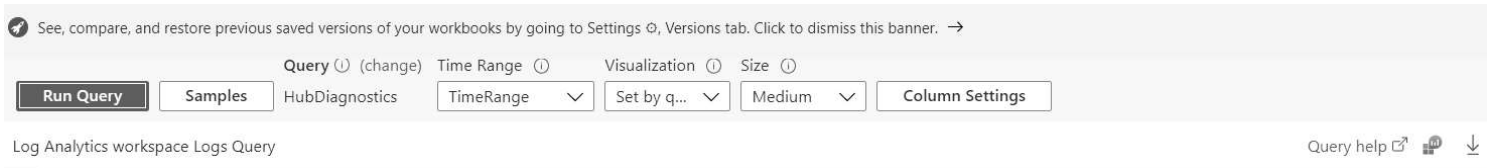
P2S User successful connections with IP



Log Query:

```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Connection successful" and Message has
"Username={UserName}"
| project splitted=split(Message, "Username=")
| mv-expand col1=splitted[0], col2=splitted[1], col3=splitted[2]
| project user=split(col2, " ")
| mv-expand username=user[0]
| project ['user']
```

EAP Authentication succeded

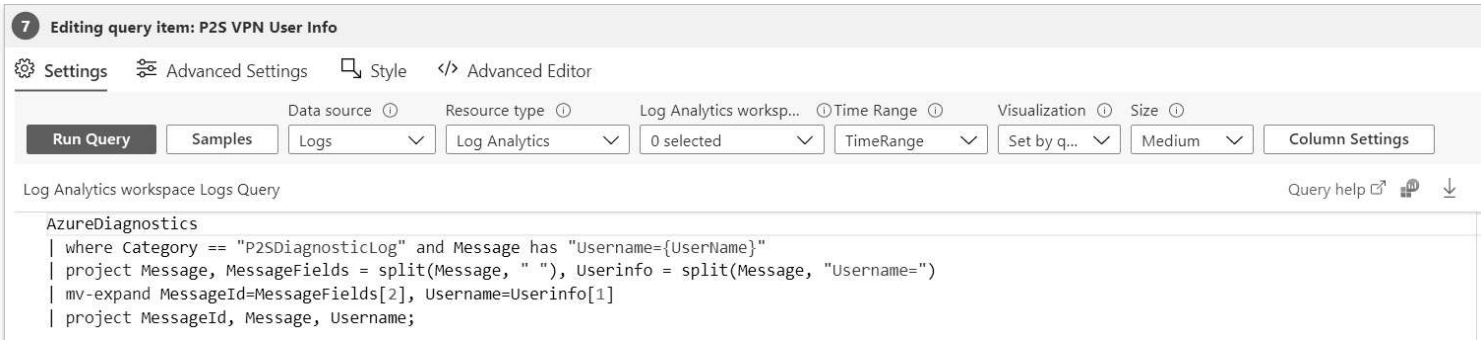



```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "EAP authentication succeeded" and Message has "Username={UserName}"
| project Message, MessageFields = split(Message, " "), Userinfo = split(Message, "Username=")
| mv-expand MessageId=MessageFields[2],user=split(Userinfo[1]," ")
| project MessageId, Message, Userinfo[1]
```

Log Query:

```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "EAP authentication succeeded" and Message has "Username={UserName}"
| project Message, MessageFields = split(Message, " "), Userinfo = split(Message, "Username=")
| mv-expand MessageId=MessageFields[2],user=split(Userinfo[1]," ")
| project MessageId, Message, Userinfo[1]
```

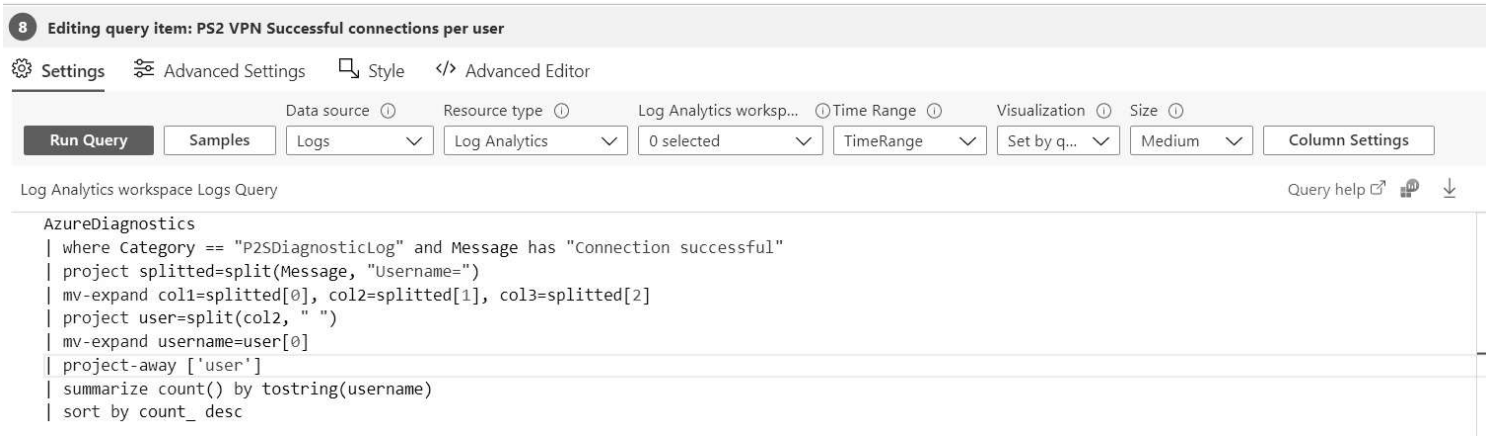
P2S VPN User Info



Log Query:

```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Username={UserName}"
| project Message, MessageFields = split(Message, " "), Userinfo = split(Message, "Username=")
| mv-expand MessageId=MessageFields[2], Username=Userinfo[1]
| project MessageId, Message, Username;
```

P2S VPN Successful connections per user

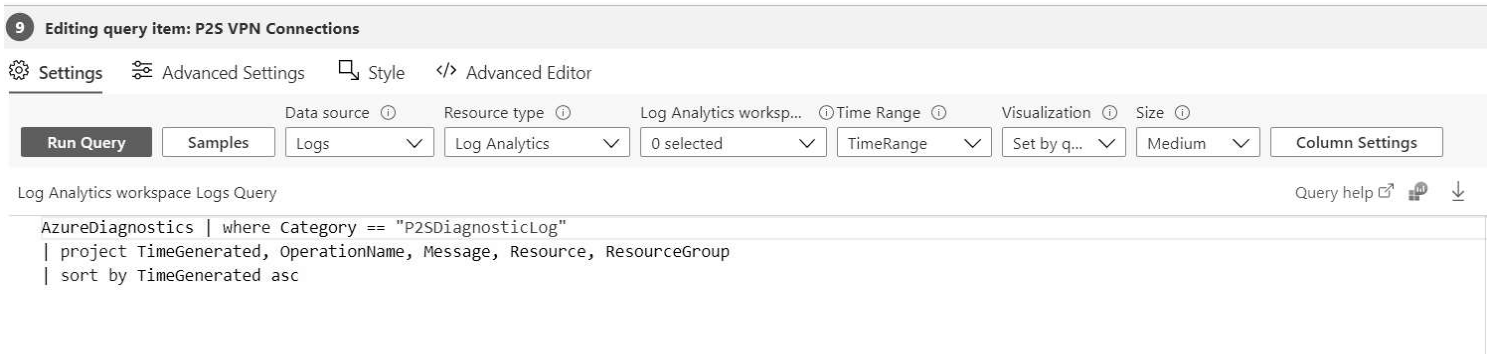


Log Query:

```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Connection successful"
| project splitted=split(Message, "Username=")
| mv-expand col1=splitted[0], col2=splitted[1], col3=splitted[2]
| project user=split(col2, " ")
| mv-expand username=user[0]
| project-away ['user']
```

| summarize count() by tostring(username)
| sort by count_ desc

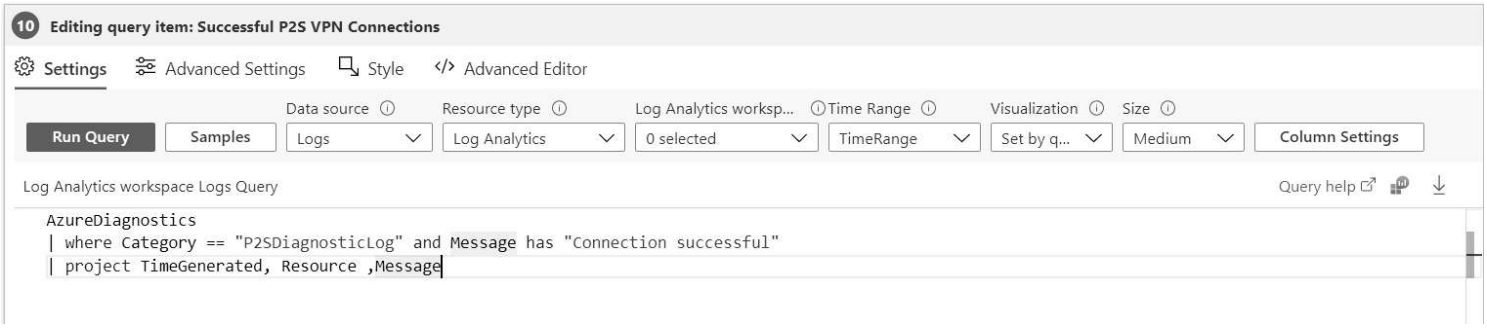
P2S VPN Connections



Log Query:

AzureDiagnostics | where Category == "P2SDiagnosticLog"
| project TimeGenerated, OperationName, Message, Resource, ResourceGroup
| sort by TimeGenerated asc

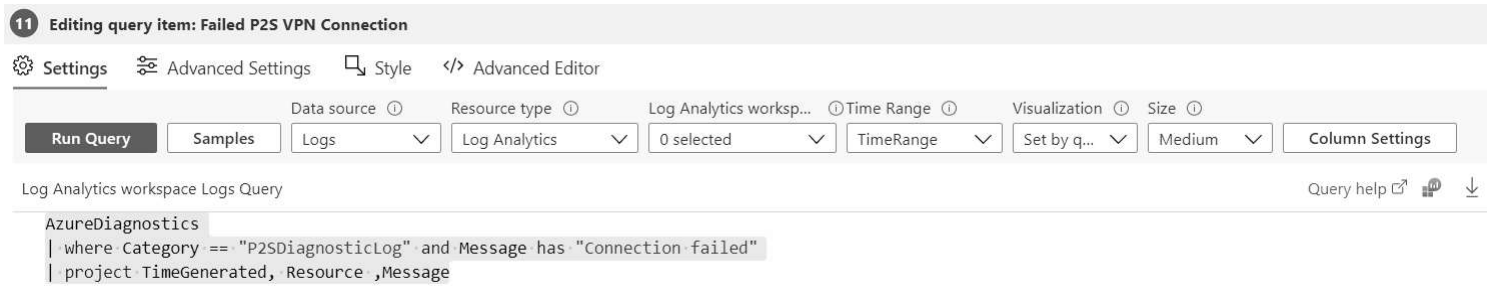
Successful P2S VPN Connections



Log Query:

AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Connection successful"
| project TimeGenerated, Resource ,Message

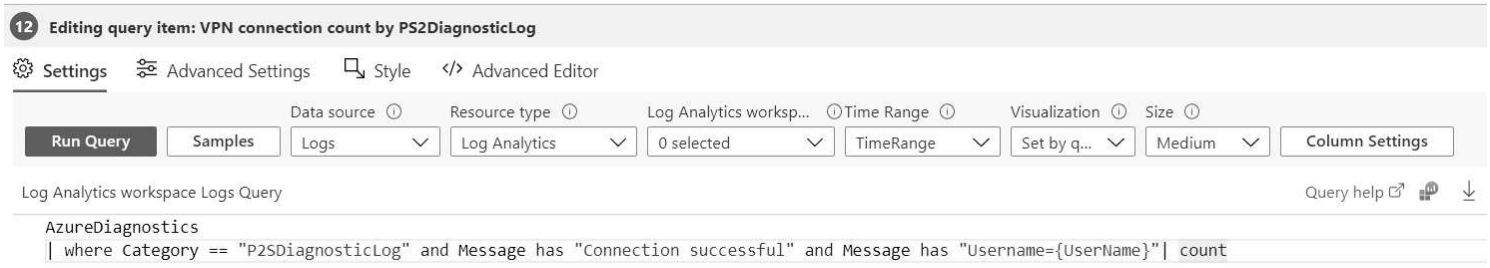
Failed P2S VPN Connections



Log Query:

```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Connection failed"
| project TimeGenerated, Resource ,Message
```

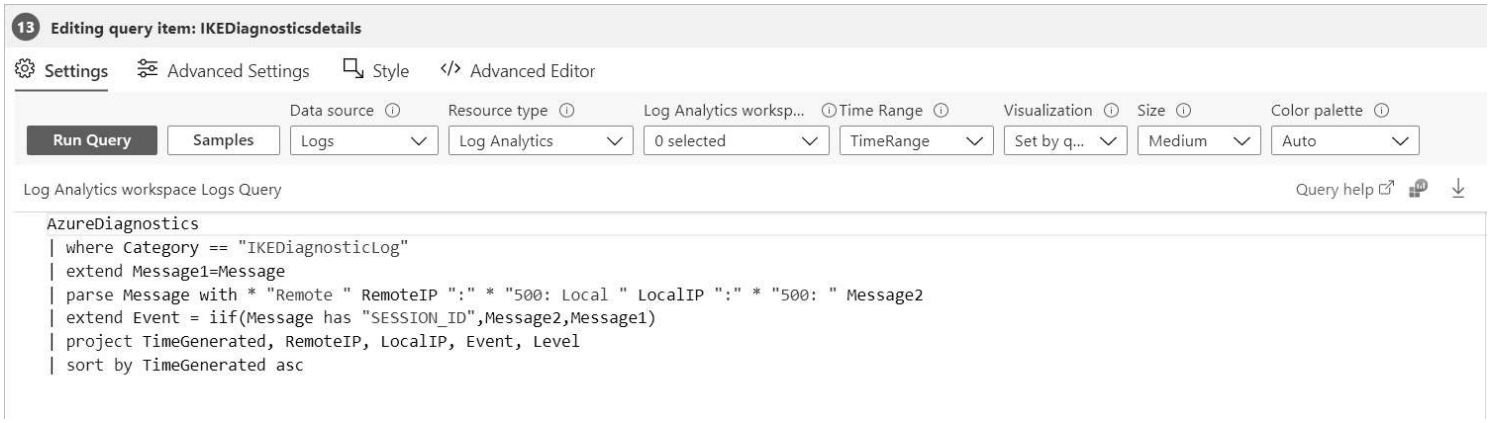
VPN Connection count by P2SDiagnosticLog



Log Query:

```
AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Connection successful" and Message has "Username={UserName}"| count
```

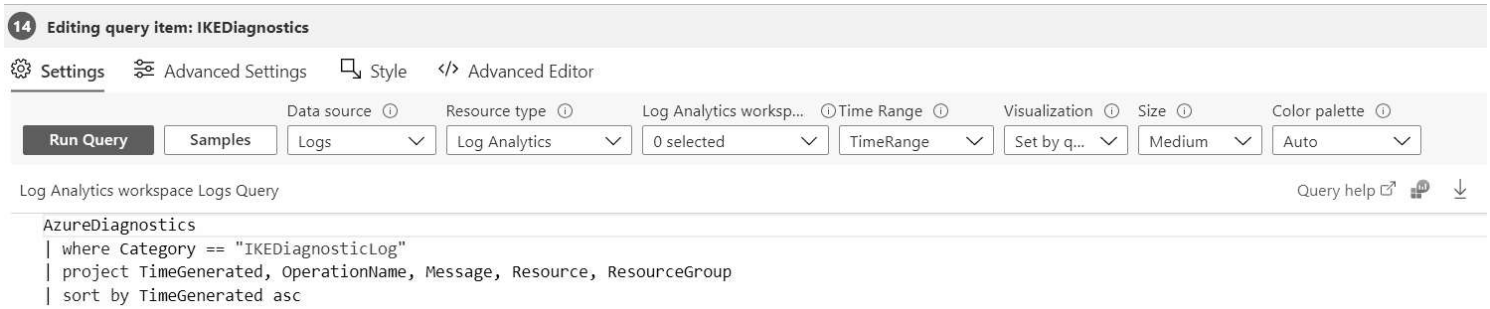
IKE Diagnosticsdetails



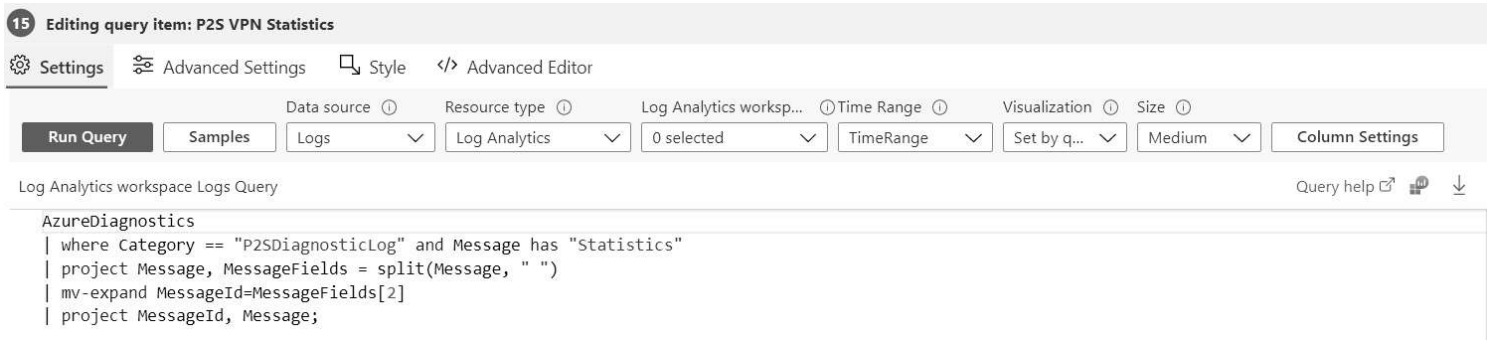
Log Query:

```
AzureDiagnostics
| where Category == "IKEDiagnosticLog"
| extend Message1=Message
| parse Message with * "Remote " RemoteIP ":" * "500: Local " LocalIP ":" * "500: " Message2
| extend Event = iif(Message has "SESSION_ID",Message2,Message1)
| project TimeGenerated, RemoteIP, LocalIP, Event, Level
| sort by TimeGenerated asc
```

IKEDiagnosticLog



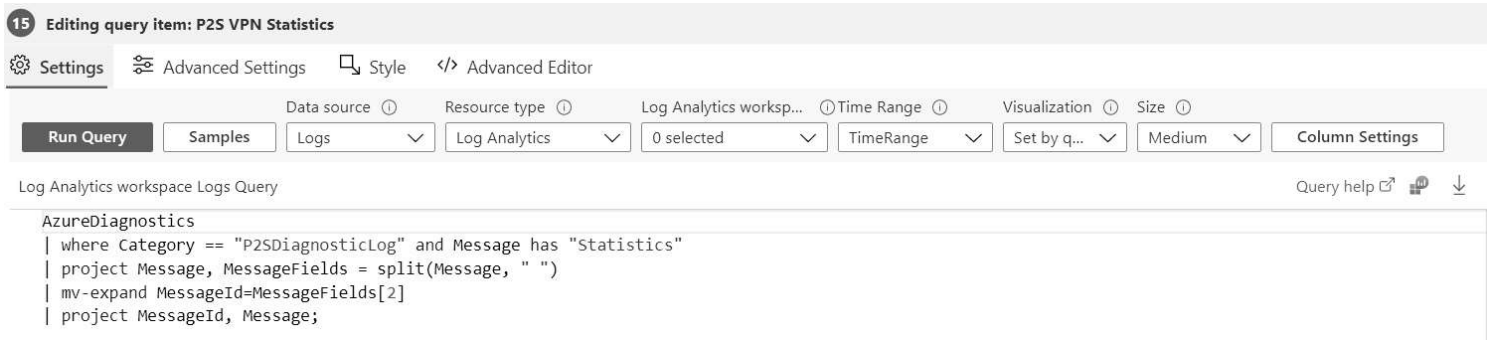
P2S VPN Statistics



Log Query:

AzureDiagnostics
| where Category == "P2SDiagnosticLog" and Message has "Statistics"
| project Message, MessageFields = split(Message, " ")
| mv-expand MessageId=MessageFields[2]
| project MessageId, Message;

P2S VPN Active Sessions Details



let P2Svpnconnections = (externaldata (resource:string, UserNameVpnConnectionHealths: dynamic) [
@"SASURI"
] with(format="multijson"));

P2Svpnconnections
| mv-expand UserNameVpnConnectionHealths
| extend Username = parse_json(UserNameVpnConnectionHealths).UserName
| extend VpnConnectionHealths =
parse_json(parse_json(UserNameVpnConnectionHealths).VpnConnectionHealths)
| mv-expand VpnConnectionHealths

```
| extend VpnConnectionId = parse_json(VpnConnectionHealths).VpnConnectionId, VpnConnectionDuration =
parse_json(VpnConnectionHealths).VpnConnectionDuration, VpnConnectionTime =
parse_json(VpnConnectionHealths).VpnConnectionTime, PublicIpAddress =
parse_json(VpnConnectionHealths).PublicIpAddress, PrivateIpAddress =
parse_json(VpnConnectionHealths).PrivateIpAddress, MaxBandwidth =
parse_json(VpnConnectionHealths).MaxBandwidth, EgressPacketsTransferred =
parse_json(VpnConnectionHealths).EgressPacketsTransferred, EgressBytesTransferred =
parse_json(VpnConnectionHealths).EgressBytesTransferred, IngressPacketsTransferred =
parse_json(VpnConnectionHealths).IngressPacketsTransferred, IngressBytesTransferred =
parse_json(VpnConnectionHealths).IngressBytesTransferred, MaxPacketsPerSecond =
parse_json(VpnConnectionHealths).MaxPacketsPerSecond
| extend PubIp = tostring(split(PublicIpAddress, "😊.[0])
| project Username, VpnConnectionId, VpnConnectionDuration, VpnConnectionTime, PubIp, PublicIpAddress,
PrivateIpAddress, MaxBandwidth, EgressPacketsTransferred, EgressBytesTransferred, IngressPacketsTransferred,
IngressBytesTransferred, MaxPacketsPerSecond;
```