Connection Metrics in Private Preview

# Key Questions

## Why do I care about Connection Metric’s network data if I have Network Watcher?

* If you have a hybrid environment
* If you are diagnosing intermittent or infrequent slow app behavior where polling may miss spikes
* You are an application owner (Dev Ops) or IT Ops, and want to see latency in an application or infrastructure context

## What is the difference between Service-Level Response Time and Network Latency?

* Service-Level Response Time refers to the time needed to reply to a request from a client. This time includes both network latency and the time that processes, containers, and their dependencies need to process the request
* Network latency is the time it takes to send data across the network, and does not include compute time

## How is polling with synthetic transactions different than the data seen with an agent hooked into the TCP stack?

* Polling with synthetic transactions means sending intermittent requests across a network, either to measure round-trip latency (e.g. with a TCP or UDP ping) or to test an endpoint. This produces accurate, but infrequent data. You can miss short-lived or intermittent errors (e.g. a particular customer who sees slow response due to application issues connected to their account).
* Hooking into the TCP stack is more invasive, but it captures every TCP request.

# Example Use Scenarios

## Scenario: Alerts on Spikes in traffic

Jill has onboarded the VMs that support her e-commerce business. Along with the other alerts she created, she made OMS alerts based on the connection data found in the InboundConnection and OutboundConnection tables to alert her when the number of Inbound connections exceed a specified count or number of bytes received over a 15 minute window. She created similar alerts for outbound connections to alert based bytes sent and received over 15 minutes.

A week later one of the alerts for outbound connections was triggered for the admdemo-auto machine. She opened the map to take a look. Once she expanded the machine to view the process she saw the outbound connections from 4 different processes. She clicked on the connection for each one, and the charts made it clear that the spike in traffic was from backup.pl.

She checked the commit log for the e-commerce project and saw that backup.pl had just been updated. She rolled back the update and created a ticket to look into the issue.

## Scenario: Migration planning

Samantha is creating a migration plan for 28 applications that are running on 300 on-premises virtual machines that her company wants to move to Azure. She has installed Azure Migrate on the virtual machines and has created 28 groups to represent the applications that are running across the VMs.

In Azure Migrate she has loaded the map for a group to identify it dependencies. The maps shows 125 clients connecting into the group of machines and 23 backed servers that the group connects to. From the map she can tell that all of the clients are already monitored and part of her current migration plan. 13 of the 23 backend servers are not currently monitored, so she needs more information to determine if they need to be considered in the migration plan.

She creates a query in Log Analytics to summarize the outbound connections from the group over a 7-day period using the connection metrics data that is now available in Log Analytics. She was able to determine that only 3 of the 13 unmonitored backend servers are utilized frequently and will now be included in her migration plan.

# Technical Details

The Connection Metrics feature introduces two new tables in Log Analytics: VMConnection and VMBoundPort. These tables provide information about the connections for a machine (inbound and outbound), as well as the server ports that are open/active on them. ConnectionMetrics are also exposed via APIs that provide the means to obtain a specific metric during a time window.

Table records are generated from data reported by the Dependency Agent. Every record represents an observation over a 1-minute time interval. The TimeGenerated column indicates the start of the time interval. Each record contains information to identify the respective entity, i.e., connection or port, as well as metrics associated with that entity. At present, only network activity that occurs using TCP over IPv4 is reported.

## Common fields and conventions

The following fields and conventions apply to both VMComputer and VMBoundPort

* Computer: fully-qualified domain name of reporting machine
* AgentId: MMA/omsagent for Linux id of reporting machine
* Machine: name of the ARM resource for the machine exposed by ServiceMap. It is of the form “m-{guid}”, where guid is the same guid as AgentId
* Process: name of the ARM resource for the process exposed by ServiceMap. It is of the form “p-{hex string}”. Process is unique within a machine scope, to generate a unique process id across machines, combine Machine and Process.
* ProcessName: executable name of the reporting process
* All IP addresses are strings in IPv4 canonical format, e.g., ‘13.107.3.160’

## Connections

VMConnection contains records for both *inbound* and *outbound* connections for a given machine. A connection is classified as outbound if is initiated by the machine and process reporting it, and as inbound, otherwise. Applied to TCP, connections resulting from *accept*-ing on a listening socket are inbound, while those created by *connect*-ing to a given IP and Port are outbound. The directionality of a connection is represented by the Direction column, which can be set to either ‘inbound’ or ‘outbound’.

To control cost and complexity, connection records do not represent individual physical network connections. Multiple physical network connections are grouped into a *logical* connection, which is then reflected in the respective table, i.e., records in VMConnection represent a logical grouping and not the individual physical connections that are being observed. Physical network connection sharing the same value for the attributes listed below during a given 1 minute interval are aggregated into a single logical record in VMConnection:

* Direction – direction of the connection (‘inbound’ or ‘outbound’)
* Machine – identity of machine reporting the connection
* Process – identity of process (or groups of processes) initiating/accepting the connection
* SourceIp – IP address of the source
* DestinationIp – IP address of the destination
* DestinationPort – Port number of the destination
* Protocol – protocol used for the connection, e.g., ‘tcp’ or ‘udp’ (only ‘tcp’ supported at present)

The identity a connection is derived from the above the above 7 fields and is stored in the ConnectionId column. This column can be used to find quickly records for a specific connection across time.

To account for the impact of grouping, information about the *number* of grouped physical connections is provided as columns in the following way:

* LinksEstablished: the number of physical network connections that have been established *during* the reporting time window
* LinksTerminated: the number of physical network connections that have been terminated *during* the reporting time window
* LinksFailed: the number of physical network connections that have failed *during* the reporting time window. This information is, currently, available only for outbound connections.
* LinksLive: the number of physical network connections that were open *at the end* of the reporting time window

### Metrics

In addition to connection count metrics, information about the *volume* of data sent and received on a given logical connection or network port is also included:

* BytesSent: total number of bytes that have been sent *during* the reporting time window
* BytesReceived: total number of bytes that have been received *during* the reporting time window

The third type of data being reported is about *response time*, i.e., how long does a caller spend waiting for a request sent over a connection to be processed and responded to by the remote endpoint. The response time reported is an *estimation* of the true response time of the underlying application protocol. It is computed using heuristics based on the observation of the flow of data between the source and destination end of a physical network connection. Conceptually, it is the difference between the time the *last byte* of a request leaves the sender, and the time when the *last byte* of the response arrives back to it. These two timestamps are used to delineate request and response events on a given physical connection. The difference between them represents the response time of a single request.

While we continue to tune and improve the heuristics we use to, it should be noted that our algorithm is an approximation, that may work with varying degree of success depending on the actual application protocol used for a given network connection. For example, the current approach works well for request-response based protocols such as HTTP(S), but does not work with one-way or message queue-based protocols.

Response time-related information is conveyed by the following columns:

* Responses: the number of responses observed *during* the reporting time window. Note that the following columns only have values when the value of this column is greater than 0.
* ResponseTimeMax: the largest response time (milliseconds) observed during the reporting time window
* ResponseTimeMin: the smallest response time (milliseconds) observed during the reporting time window
* ResponseTimeSum: the sum of all response times (milliseconds) observed during the reporting time window

### Naming and Classification

For convenience, the IP address of the *remote* end of a connection is included in the RemoteIp column. For inbound connections, RemoteIp is the same as SourceIp, while for outbound connections, it is the same as DestinationIp. The RemoteDnsCanonicalNames field is a string array that includes all DNS canonical names reported by the machine for RemoteIp. The RemoteDnsQuestions field is currently not populated. Eventually, it will include an array of DNS questions that were asked about RemoteIp on the reporting machine. The RemoteClassification fields is reserved for future use.

### Gelocation

VMConnection also includes geolocation information for the remote end of each connection record (RemoteIp). RemoteCountry is the name of the country hosting RemoteIp, e.g., ‘United States’, RemoteLatitude is the geolocation latitude, e.g., 47.68, and RemoteLongitude is the geolocation longitude, e.g., -122.12.

### Malicious IP

Every RemoteIp in VMConnection is checked against a set of IPs with known malicious activity. If the RemoteIp is identified as malicious the following fields will be populated (they are empty, when the IP is not considered malicious):

* MaliciousIp – will be set to RemoteIp
* IndicatorThreadType
* Description
* TLPLevel
* Confidence
* Severity
* FirstReportedDateTime
* LastReportedDateTime
* IsActive
* ReportReferenceLink
* AdditionalInformation

## Ports

Ports on a machine that actively accept incoming traffic or could potentially accept traffic, but are idle during the reporting time window are represented in VMBoundPort.

By default, we do not write data to the VMBoundPort table. To have data written to this table, please send an email to [vminsights@microsoft.com](mailto:vminsights@microsoft.com) along with your Workspace ID and Workspace region.

Every record in VMBoundPort is identified by the following fields:

* Process – identity of process (or groups of processes) with which the port is associated
* Ip – port IP address (can be wildcard IP, ‘0.0.0.0’)
* Port – Port number
* Protocol – protocol, e.g., ‘tcp’ or ‘udp’ (only ‘tcp’ supported at present)

The identity a port is derived from the above the above 5 fields and is stored in the PortId column. This column can be used to find quickly records for a specific port across time.

### Metrics

Port records include metrics representing the connections associated with them. At present the following metrics are reported (the details for each metric are described in the previous section):

* BytesSent and BytesReceived
* LinksEstablished, LinksTerminated, LinksLive
* ResposeTime, ResponseTimeMin, ResponseTimeMax, ResponseTimeSum

### Notes

* If a process accepts connections on the same IP address but over multiple network interfaces, a separate record for each interface will be reported.
* Records with wildcard IP will contain no activity. They are included to represent the fact that a port on the machine is open to inbound traffic.
* To reduce verbosity and data volume, records with wildcard IP will be omitted when there is a matching record (for the same process, port, and protocol) with a specific IP address. When a wildcard IP record is omitted, the IsWildcardBind field on the record with the specific IP address, will be set to True, to indicate that the port is exposed over every interface of the reporting machine.
* Ports that are bound only on a specific interface have IsWildcardBind set to ‘False’

# Example Query

// the machines of interest  
let machines = datatable(m: string) ["m-82412a7a-6a32-45a9-a8d6-538354224a25"];  
// map of ip to monitored machine in the environment  
let ips=materialize(ServiceMapComputer\_CL  
| summarize ips=makeset(todynamic(Ipv4Addresses\_s)) by MonitoredMachine=ResourceName\_s  
| mvexpand ips to typeof(string));  
// all connections to/from the machines of interest  
let out=materialize(VMConnection  
| where Machine in (machines)  
| summarize arg\_max(TimeGenerated, \*) by ConnectionId);  
// connections to localhost augmented with RemoteMachine  
let local=out  
| where RemoteIp startswith "127."  
| project ConnectionId, Direction, Machine, Process, ProcessName, SourceIp, DestinationIp, DestinationPort, Protocol, RemoteIp, RemoteMachine=Machine;  
// connections not to localhost augmented with RemoteMachine  
let remote=materialize(out  
| where RemoteIp !startswith "127."  
| join kind=leftouter (ips) on $left.RemoteIp == $right.ips  
| summarize by ConnectionId, Direction, Machine, Process, ProcessName, SourceIp, DestinationIp, DestinationPort, Protocol, RemoteIp, RemoteMachine=MonitoredMachine);  
// the remote machines to/from which we have connections  
let remoteMachines = remote | summarize by RemoteMachine;  
// all augmented connections  
(local)  
| union (remote)  
//Take all outbound records but only inbound records that come from either //unmonitored machines or monitored machines not in the set for which we are computing dependencies.  
| where Direction == 'outbound' or (Direction == 'inbound' and RemoteMachine !in (machines))  
| summarize by ConnectionId, Direction, Machine, Process, ProcessName, SourceIp, DestinationIp, DestinationPort, Protocol, RemoteIp, RemoteMachine  
// identify the remote port  
| extend RemotePort=iff(Direction == 'outbound', DestinationPort, 0)  
// construct the join key we'll use to find a matching port  
| extend JoinKey=strcat\_delim(':', RemoteMachine, RemoteIp, RemotePort, Protocol)  
// find a matching port  
| join kind=leftouter (VMBoundPort   
| where Machine in (remoteMachines)   
| summarize arg\_max(TimeGenerated, \*) by PortId   
| extend JoinKey=strcat\_delim(':', Machine, Ip, Port, Protocol)) on JoinKey  
// aggregate the remote information  
| summarize Remote=makeset(iff(isempty(RemoteMachine), todynamic('{}'), pack('Machine', RemoteMachine, 'Process', Process1, 'ProcessName', ProcessName1))) by ConnectionId, Direction, Machine, Process, ProcessName, SourceIp, DestinationIp, DestinationPort, Protocol