Connect via VNet-local endpoint

Last updated by | Vitor Tomaz | Mar 14, 2023 at 1:04 PM PDT

Contents

Self-help content presented in Azure Portal

(This content was shown to the customer during case submission. It's also visible on 'Diagnose and solve problems' blade.)

Learn how to connect an application to a Managed Instance via VNet-local endpoint

The default, VNet-local endpoint deployed with each Azure SQL Managed Instance behaves as if a computer running the service were physically attached to your virtual network. It allows near-complete traffic control via route tables, network security groups, DNS resolution, firewalls, and similar mechanisms. You can also use this endpoint to involve your instance in scenarios requiring connectivity on ports other than 1433, such as auto-failover groups, distributed transactions, and MI Link.

The VNet-local endpoint is the default means to connect to SQL Managed Instance. It is a domain name of the form <mi_name>.<dns_zone>.database.windows.net that resolves to an IP address from the subnet's address pool; hence "VNet-local", or an endpoint that is local to the virtual network. The VNet-local endpoint can be used to connect a SQL Managed Instance in all standard connectivity scenarios.

There are multiple choices when deciding how and where you host your application. You may choose to host an application in the cloud by using Azure App Service or with one of Azure's virtual network integrated options like Azure App Service Environment, Azure Virtual Machines, and virtual machine scale sets. You could also take a hybrid cloud approach and keep your applications on-premises. Whichever method is chosen, you can connect it to Azure SQL Managed Instance via VNet-local endpoint.

Scan the following headings, and select one or more to learn more about how to connect an application to a Managed Instance from inside the virtual network.

Connect inside the same VNet

Connecting an application inside the same virtual network as SQL Managed Instance is the simplest scenario. Virtual machines inside the virtual network can connect to each other directly even if they are inside different subnets. That means that all you need to connect an application inside App Service Environment, or a virtual machine is to set the connection string appropriately.

Sample connection strings can be seen with the **Connection strings** option in the left menu on your SQL Managed Instances page in Azure Portal.

Connect from a different VNet

To connect from a different VNet, the application needs to be able to access the virtual network where SQL Managed Instance is deployed. To create access to the VNet, make a connection between the application virtual network and the SQL Managed Instance virtual network. The virtual networks don't have to be in the same subscription for this scenario to work.

Azure VNet peering is preferable because it uses the Microsoft backbone network. In regard to connectivity, there is no noticeable difference in latency between virtual machines in a peered virtual network and in the same virtual network.

See more details about other options with this article, Connect inside a different VNet.

Connect from on-premises

You can also connect your on-premises application to SQL Managed Instance via virtual network (private IP address). To access it from on-premises, you need to make a site-to-site connection between the application and the SQL Managed Instance virtual network.

There are two options for how to connect on-premises to an Azure virtual network:

- Site-to-site VPN connection (Azure portal, PowerShell, Azure CLI)
- Azure ExpressRoute connection
- Point-to-site connection to a virtual network

If you've established an on-premises to Azure connection successfully and you can't establish a connection to SQL Managed Instance, check if your firewall has an open outbound connection on SQL port 1433 as well as the 11000-11999 range of ports for redirection.

Connect from Power BI

To connect Power BI to Managed Instance through it's private IP address, install an on premise data gateway on a virtual machine that has virtual network connectivity to the Managed Instance.

Connect from Azure Data Factory

To access the SQL Managed Instance private endpoint, set up a self-hosted integration runtime that can access the database.

- If you provision the self-hosted integration runtime in the same virtual network as your Managed Instance, make sure that your integration runtime machine is in a different subnet than your Managed Instance.
- If you provision your self-hosted integration runtime in a different virtual network than your Managed Instance, you can use either a virtual network peering or a virtual network to virtual network connection.
- For more information, see Connect your application to SQL Managed Instance.

Connect from Azure App Services

To connect an application that's hosted by Azure App Service to Managed Instance through its private IP address, you first need to make a connection between the application and the SQL Managed Instance virtual network.

- For more information, see Integrate your app with an Azure virtual network.
- For troubleshooting tips, see Troubleshooting virtual networks and applications.
- If a connection cannot be established, try syncing the networking configuration.

Connecting Azure App Service to SQL Managed Instance When you integrate Azure App Service to a network peered to a SQL Managed Instance virtual network (they don't belong to the same VNet), the following configuration is required to be set up:

- SQL Managed Instance virtual network must not have a gateway.
- SQL Managed Instance virtual network must have the `Use remote gateways` option set.
- Peered virtual network must have the `Allow gateway transit` option set.

Connect to an SMTP for Database Mail

If you are trying to reach the mail server from a Managed Instance using an SQL Agent Job that tests the network connection:

- Make sure that you enabled the port that communicates with the email server. Add the Port in the Outbound security rules of the Network Security Group that controls access to your Managed Instance.
- Create a SQL Agent job that has one PowerShell task that runs a command like `tnc <your_email_server> -Port <your_email_server_port>`. Run the job and check the job output in the job history. Confirm the DNS resolution is correct and that the port is reachable.

Learn more about firewalls

Azure SQL Managed Instance does not have IP-based firewall like Azure SQL Database. The access to SQL Managed Instance is controlled via Network Security Groups (NSG). See allow public endpoint traffic on the network security group for an example on how to add rules to an NSG.

Troubleshooting connectivity issues

Common issues and solutions

- If you are unable to connect to SQL Managed Instance from an Azure virtual machine within the same virtual network but a different subnet, check if you have a Network Security Group set on VM subnet that might be blocking access. Additionally, open outbound connection on SQL port 1433 as well as ports in the range 11000-11999, since those are needed for connecting via redirection inside the Azure boundary.

- Ensure that BGP Propagation is set to Enabled for the route table associated with the virtual network.
- If using P2S VPN, check the configuration in the Azure portal to see if you see Ingress/Egress numbers. Non-zero numbers indicate that Azure is routing traffic to/from on-premises.
- If you're using virtual network peering, ensure that you have followed the instructions for setting Allow Gateway Transit and Use Remote Gateways.
- If you're using virtual network peering to connect an Azure App Service hosted application, and the SQL Managed Instance virtual network has a public IP address range, make sure that your hosted application settings allow your outbound traffic to be routed to public IP networks. Follow the instructions in the article, Regional virtual network integration.

Test connectivity using Azure SQL Connectivity Checker

The Azure SQL Connectivity Checker tool is a PowerShell script, run from the client machine, which automates a series of checks for the most common configuration problems. Most issues it detects will come with recommendations for how to resolve them.

Run the following PowerShell script from the Windows client computers where the error is occurring.

- To run the tests from Linux, from machines without internet access, or from a containerized environment, see SQL Connectivity Checker on GitHub.

1. Open Windows PowerShell ISE (in **Administrator mode** if possible).
   Note: To collect a network trace along with the tests (`CollectNetworkTrace` parameter), you must run PowerShell as an administrator.
2. Open a **New Script** window.
3. Paste the following in the script window:

```
$parameters = @{
    # Supports Single, Elastic Pools and Managed Instance (provide FQDN, MI public endpoint is supported)
    # Supports Azure Synapse / Azure SQL Data Warehouse (*.sql.azuresynapse.net / *.database.windows.net)
    # Supports Public Cloud (*.database.windows.net), Azure China (*.database.chinacloudapi.cn), Azure Ge
    Server = '.database.windows.net' # or any other supported FQDN
    Database = ''  # Set the name of the database you wish to test, 'master' will be used by default if n
    User = ''  # Set the login username you wish to use, 'AzSQLConnCheckerUser' will be used by default i
    Password = ''  # Set the login password you wish to use, 'AzSQLConnCheckerPassword' will be used by d

    ## Optional parameters (default values will be used if omitted)
    SendAnonymousUsageData = $true  # Set as $true (default) or $false
    RunAdvancedConnectivityPolicyTests = $true  # Set as $true (default) or $false, this will load the li
    ConnectionAttempts = 1 # Number of connection attempts while running advanced connectivity tests
    DelayBetweenConnections = 1 # Number of seconds to wait between connection attempts while running adv
    CollectNetworkTrace = $true  # Set as $true (default) or $false
    #EncryptionProtocol = '' # Supported values: 'Tls 1.0', 'Tls 1.1', 'Tls 1.2'; Without this parameter
}

$ProgressPreference = "SilentlyContinue";
if ("AzureKudu" -eq $env:DOTNET_CLI_TELEMETRY_PROFILE) {
    $scriptFile = '/ReducedSQLConnectivityChecker.ps1'
} else {
    $scriptFile = '/AzureSQLConnectivityChecker.ps1'
}
$scriptUrlBase = 'http://raw.githubusercontent.com/Azure/SQL-Connectivity-Checker/master'
cls
Write-Host 'Trying to download the script file from GitHub (https://github.com/Azure/SQL-Connectivity-Che
try {
    [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12 -bor [Net.SecurityPro
    Invoke-Command -ScriptBlock ([Scriptblock]::Create((Invoke-WebRequest ($scriptUrlBase + $scriptFile)
    }
catch {
    Write-Host 'ERROR: The script file could not be downloaded:' -ForegroundColor Red
    $_.Exception
    Write-Host 'Confirm this machine can access https://github.com/Azure/SQL-Connectivity-Checker/' -Fore
    Write-Host 'or use a machine with Internet access to see how to run this from machines without Intern
}
#end
```

4. Set the parameters on the script. You must set the server name and database name. Setting the user and password is best practice but optional.

5. Run the script. The results are displayed in the output window. If the user has permissions to create folders, a folder with the resulting log file is created along with a ZIP file (`AllFiles.zip`). When running on Windows, the folder opens automatically after the script completes.

6. Examine the output for any issues detected and any recommended steps to resolve the issue.

7. If the issue can't be resolved, send `AllFiles.zip` using the **File upload** option in the **Details** step of creating your support case.

Resources

- Connectivity architecture for Azure SQL Managed Instance
- Connect your application to Azure SQL Managed Instance
- Configure connection types