

Connectivity issues to the Azure Query Editor

Last updated by | Subbu Kandhaswamy | Jun 10, 2021 at 11:00 AM PDT

Contents

- [Connectivity issues to the Azure Query Editor](#)
 - [Issue](#)
 - [Troubleshoot](#)
 - [RCA Template](#)
 - [Classification](#)

Connectivity issues to the Azure Query Editor

Issue

Customers may be reporting connectivity issues to the Azure Query Editor. This can be caused by the following categories of issues:

- Client side issues, such as browser request filtering.
- Issues in transit, such as misconfigured HTTP proxies or firewall appliances.
- Server side issues, such as performance problems, downstream connectivity issues, etc.
- It is necessary to determine if the customer's request has reached the application.

Troubleshoot

Please refer attached PS script to troubleshoot client-side connectivity issues as follows:

```

#<PowerShell Script BEGIN >

param (
    [Parameter(Mandatory=$true)]
    [string]$serverName
)

Write-Host "Azure SQL HTTPS querying troubleshooter.";
Write-Host;

if (-not $serverName.Contains('.')) {
    $serverName = "$serverName.database.windows.net";
}

$port = 1443;
$expectedIssuer = "CN=Microsoft IT TLS CA";
$sentinelQuery = "SELECT 1";
$uri = "https://${serverName}:${port}/databases/master/query?api-version=2018-08-01-preview&application=Azure+

Write-Host -NoNewline "Checking connectivity to ${serverName}:${port}... ";
$tcpsocket = $null;
$tcperror = $null;
try {
    $tcpsocket = New-Object Net.Sockets.TcpClient($serverName, $port);
} catch {
    $tcperror = $_;
}

if (-not $tcpsocket -or -not $tcpsocket.Connected) {
    Write-Host -ForegroundColor Red "Failed";
    Write-Host;
    Write-Host -ForegroundColor Red "Could not connect to $serverName over port $port. You may need to enable
    if ($tcperror) {
        Write-Host;
        Write-Host -ForegroundColor Red $tcperror;
    }
    exit 1;
}

Write-Host -ForegroundColor Green "Passed";

Write-Host -NoNewline "Checking certificate validity... ";
$sslStream = New-Object Net.Security.SslStream($tcpsocket.GetStream(), $false);
$cert = $null;
$certerror = $null;
try {
    $sslStream.AuthenticateAsClient($serverName);
    $cert = New-Object Security.Cryptography.X509Certificates.X509Certificate2($sslStream.RemoteCertificate);
} catch {
    $certerror = $_;
}

if (-not $cert) {
    Write-Host -ForegroundColor Red "Failed";
    Write-Host;
    Write-Host -ForegroundColor Red "Could not find a server certificate.";
    if ($certerror) {
        Write-Host;
        Write-Host -ForegroundColor Red $certerror;
    }
    exit 1;
} elseif (-not $cert.Issuer.StartsWith($expectedIssuer)) {
    Write-Host -ForegroundColor Yellow "Warning";
    Write-Host;
    Write-Host -ForegroundColor Yellow "Certificate issuer did not match the expected value.";
    Write-Host;
} else {
    Write-Host -ForegroundColor Green "Passed";
}

```

```

}

$tcpsocket.Close();
Write-Host -NoNewline "Running test query... ";

$credentials = Get-Credential -Message "Provide login information for $serverName";
$auth = "Basic " + [Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes("$(($credentials.UserName):$([System]::Security.Cryptography.HMACSHA256::ComputeHash($credentials.Password).ToString('x'))"));
$now = [DateTime]::UtcNow;
$notBefore = "not-before=" + $now.AddMinutes(-5).ToString("yyyy-MM-ddTHH:mm:ssZ");
$notAfter = "not-after=" + $now.AddMinutes(5).ToString("yyyy-MM-ddTHH:mm:ssZ");
$toSign = "$notBefore`r`n$notAfter`r`nauthorization: $auth`r`nhost: ${serverName}:${port}`r`n";
$hmactsha = New-Object System.Security.Cryptography.HMACSHA256
$hmactsha.Key = [Text.Encoding]::UTF8.GetBytes($auth)
$signature = $hmactsha.ComputeHash($toSign)
$csrfSignature = "$signature; $notBefore; $notAfter; signed-headers=authorization,host";

$response = $null;
$queryerror = $null;
try {
    $response = (Invoke-WebRequest -Method POST -Uri $uri -Headers @{ "Authorization" = "$auth"; "X-CSRF-Signature" = "$csrfSignature" })
} catch {
    if ($_.Exception -and $_.Exception.Response) {
        $response = $_.Exception.Response;
    } else {
        $queryerror = $_;
    }
}

if (-not $response) {
    Write-Host -ForegroundColor Red "Failed";
    Write-Host;
    Write-Host -ForegroundColor Red "Failed to invoke HTTPS request: no response was received.";
    if ($queryerror) {
        Write-Host;
        Write-Host -ForegroundColor Red $queryerror;
    }
    exit 1;
} elseif (-not ($response.StatusCode -eq 200)) {
    try {
        $details = $null;
        try {
            $details = $response.Content
        } catch {
        }

        if (-not $details) {
            try {
                $details = (New-Object System.IO.StreamReader($response.GetResponseStream())).ReadToEnd();
            } catch {
            }
        }
    } catch {
    }

    $serverDate = $null;
    try {
        $serverDate = [DateTime]::ParseExact($response.Headers['Date'], "ddd, dd MMM yyyy HH:mm:ss 'GMT'", [CultureInfo]::InvariantCulture)
    } catch {
    }

    if ($details) {
        if ($details.Contains("The Authorization header was not found.")) {
            Write-Host -ForegroundColor Red "Failed";
            Write-Host;
            Write-Host -ForegroundColor Red "The server reported that the Authorization header was not found."
        } elseif ($response.StatusCode -eq 401 -and $details.Contains("Login failed for user")) {
            Write-Host -ForegroundColor Yellow "Warning";
            Write-Host;
            Write-Host -ForegroundColor Yellow "Your login has failed. The server has properly responded with"
        }
    }
}

```

```

Write-Host;
Write-Host -ForegroundColor Yellow $details;
} elseif ($response.StatusCode -eq 403 -and $details.Contains("Client with IP address")) {
Write-Host -ForegroundColor Yellow "Warning";
Write-Host;
Write-Host -ForegroundColor Yellow "Your login has failed. The server has properly responded with";
Write-Host;
Write-Host -ForegroundColor Yellow $details;

} elseif ($serverDate -and ($serverDate -gt $now.AddMinutes(5) -or $serverDate -lt $now.AddMinutes(-5)) {
Write-Host -ForegroundColor Red "Failed";
Write-Host;
Write-Host -ForegroundColor Red "Your clock is more than 5 minutes different than the server's clo

} else {
Write-Host -ForegroundColor Red "Failed";
Write-Host;
Write-Host -ForegroundColor Red $details;
}
} else {
Write-Host -ForegroundColor Red "Failed";
Write-Host -ForegroundColor Red "Failed to execute query.";
Write-Host;
Write-Host -ForegroundColor Red $response.StatusCode;
Write-Host -ForegroundColor Red $response.StatusDescription;
}
exit 1;
}

Write-Host -ForegroundColor Green "Passed";
Write-Host;
Write-Host -ForegroundColor Green "Success";

#<PowerShell Script = END>

```

The screenshot shows a PowerShell terminal window with a dark blue background. The text is as follows:

```

cmdlet at command pipeline position 1
Supply values for the following parameters:
serverName: rg1server2.database.windows.net
Azure SQL HTTPS querying troubleshooter.

Checking connectivity to rg1server2.database.windows.net:1443... Passed
Checking certificate validity... Passed
Running test query... Passed


Success

PS C:\Users\sadev>

```

Customer ready message if customers detected that port 1443 on their side cannot be open. We should provide them with the following message:

RCA Template

Thank you for reporting this issue. As pointed in the [documentation](#) , in order to use the Query Editor functionality in the Azure portal, customers have to open ports 443 and 1443 on their local network for outbound HTTPS traffic. This is a new requirement due to recent changes of the underlying components

powering this experience to a more secure and robust implementation. Modern web browsers are extremely capable application programming platforms, but they do not today support connecting directly to an Azure SQL database. That gap used to be bridged via an intermediate service which has been deprecated on 7/2/2019, fully decommissioned on 9/15/2019 and replaced with a Web Query HTTPS API endpoint. This new API is more secure as it honors your SQL server's firewall configuration.

A very common question is why does port 1443 need to be open on the customer's network when port 443 is commonly used for HTTP/HTTPS traffic. This design choice was made in order to avoid throttling of other HTTP traffic on this port, such as any Azure SQL manageability HTTP request. This is a temporary state, as we are planning more changes to this end point that will eventually securely eliminate this requirement and only use port 443. We apologize for any inconvenience this is causing in the meantime and we appreciate your patience as we work towards providing you with rich functionality in the most reliable and secure way. The best way to work around this temporary problem for any customer who cannot open port 1443 is to use one of our client tools for querying databases, such as SSMS.

Classification

Root Cause: Azure SQL DB v2\Connectivity\Login Errors\Firewall errors and misconfigurations

How good have you found this content?

