AppDynamics

- Setup

Setup

create docker image

docker run -d --name windows_vm -p 8006:8006 -p 3389:3389/tcp -p 3389:3389/udp --device=/dev/kvm --cap-add=NET_ADMIN -e VERSION="11" -e RAM_SIZE="4G" -e CPU_CORES="2" --stop-timeout 120 dockurr/windows

RDP into the virtual machine and use powershell

Invoke-WebRequest -Uri "https://builds.dotnet.microsoft.com/dotnet/Sdk/10.0.100/dotnet-sdk-10.0.100-win-x64.exe" -OutFile "$env:TEMP\dotnet-sdk.exe"

Start-Process "$env:TEMP\dotnet-sdk.exe" -ArgumentList "/install /quiet /norestart" -Wait

```
mkdir C:\metricpoc
cd C:\metricpoc
dotnet new console
```

Replace program.cs

```
using System.Net.Http.Json;

using var http = new HttpClient
{
    BaseAddress = new Uri("http://localhost:8293")
```

```
};

while (true)
{
    var metricName = "Custom Metrics|WindowsVmPoc|CSharpLoop";
    var value = Random.Shared.Next(1, 100);

    var payload = new[]
    {
        new
        {
            metricName,
            aggregatorType = "OBSERVATION",
            value
        }
    };

    Console.WriteLine($"{DateTime.Now:HH:mm:ss} Sending {metricName} = {value}");

    using var response = await http.PostAsJsonAsync("/api/v1/metrics", payload);

    Console.WriteLine($"HTTP {(int)response.StatusCode} {response.StatusCode}");
    response.EnsureSuccessStatusCode();




    var metricName2 =
        "Server|Component:SdkConsole|Custom Metrics|WindowsVmPoc|CSharpLoop";

    var value2 = Random.Shared.Next(1, 100);

var payload2 = new[]
{
    new
    {
        metricName = metricName2,
        aggregatorType = "OBSERVATION",
        value = value2
    }
```

```
};
    Console.WriteLine($"{DateTime.Now:HH:mm:ss} Sending {metricName2} = {value2}");

    using var response2 = await http.PostAsJsonAsync("/api/v1/metrics", payload2);

    Console.WriteLine($"HTTP {(int)response.StatusCode} {response.StatusCode}");
    response2.EnsureSuccessStatusCode();



    await Task.Delay(TimeSpan.FromSeconds(60));
}
```

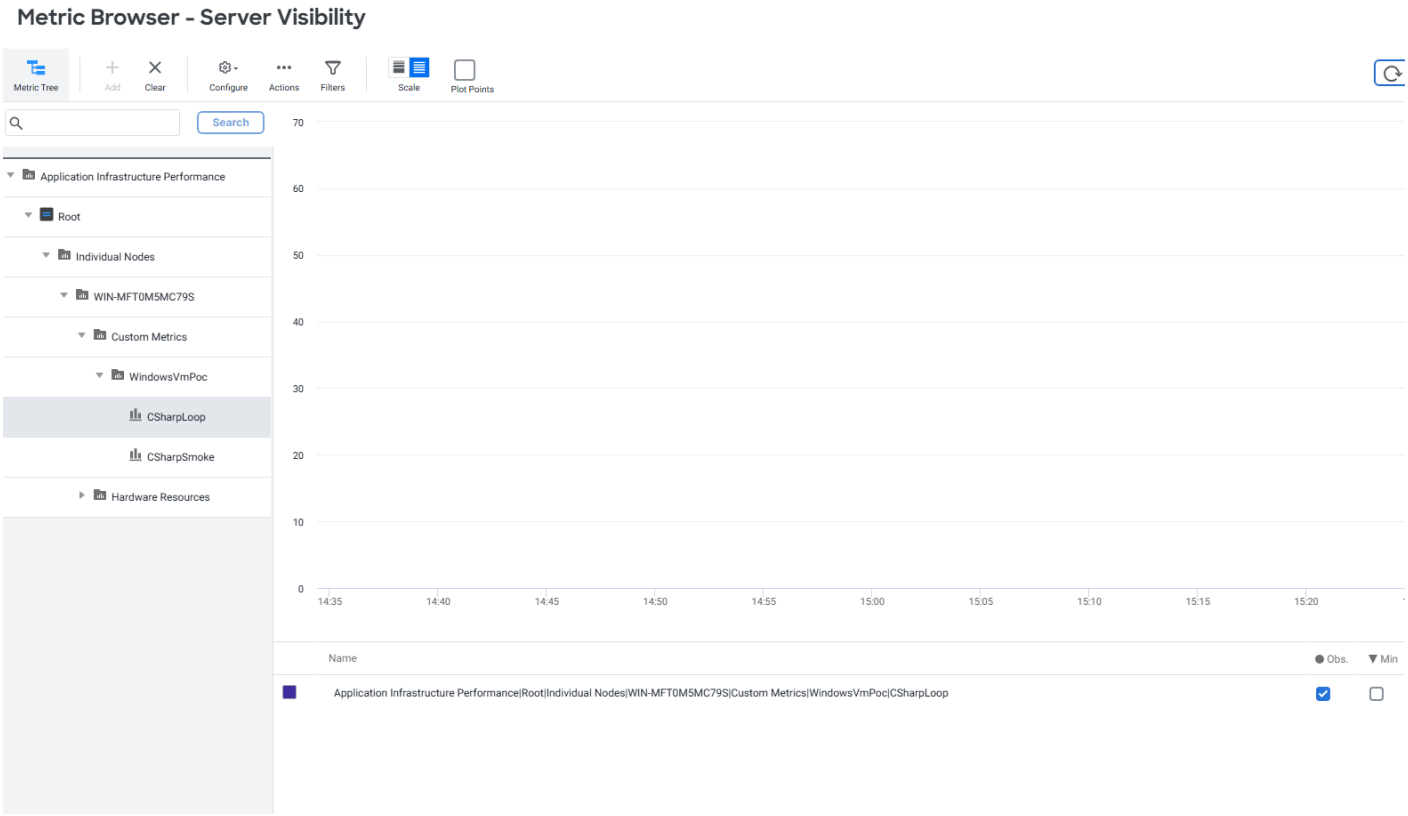
Turn off firwall

notepad C:\AppDynamics\bin\MachineAgentService.vmoptions add these at the end

```
-Dmetric.http.listener=true
-Dmetric.http.listener.port=8293
-Dmetric.http.listener.host=0.0.0.0
```

disable or configure irewall

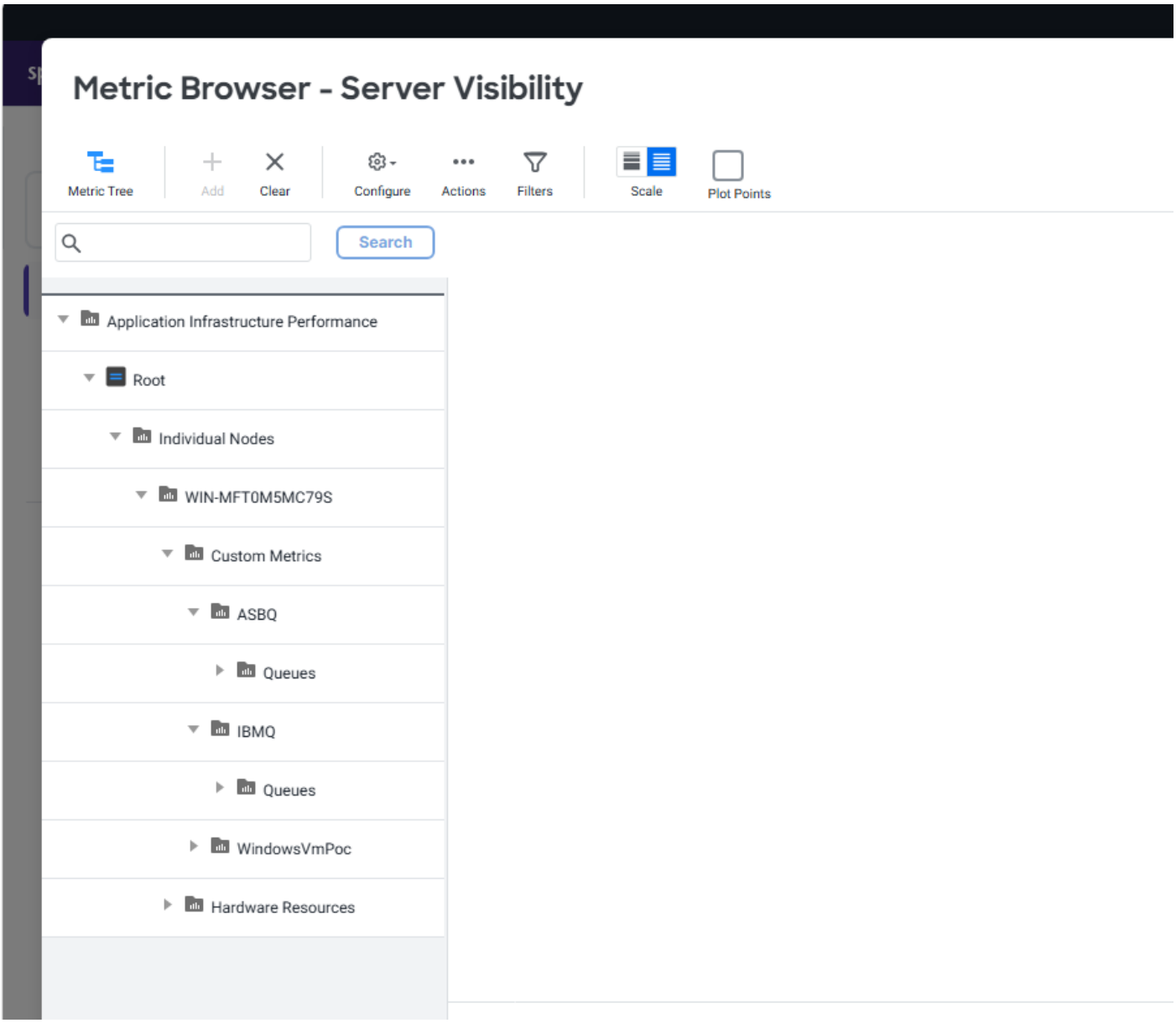
```
dotnet run
```




```
using System.Net.Http.Json;

using var http = new HttpClient
{
    BaseAddress = new Uri("http://localhost:8293")
};

var ibmQueues = new[]
{
    "ORDER.IN",
    "ORDER.OUT",
    "PAYMENT.IN",
    "PAYMENT.DEADLETTER",
    "WORKFLOW.EVENTS"
};

var asbQueues = new[]
{
    "workflow-command",
    "workflow-event",
    "deadletter-retry",
    "notification-out",
    "audit-events"
};

Console.WriteLine("Starting AppDynamics fictional queue metric simulator.");
Console.WriteLine("Target: http://localhost:8293/api/v1/metrics");
```

```
Console.WriteLine();

while (true)
{
    var metrics = new List();

    AddIbmQueueMetrics(metrics, ibmQueues);
    AddAzureServiceBusQueueMetrics(metrics, asbQueues);

    Console.WriteLine($"{DateTime.Now:HH:mm:ss} Sending {metrics.Count} metrics...");

    using var response = await http.PostAsJsonAsync("/api/v1/metrics", metrics);

    var body = await response.Content.ReadAsStringAsync();

    Console.WriteLine($"HTTP {(int)response.StatusCode} {response.StatusCode}");

    if (!string.IsNullOrWhiteSpace(body))
        Console.WriteLine(body);

    response.EnsureSuccessStatusCode();

    await Task.Delay(TimeSpan.FromSeconds(60));
}

static void AddIbmQueueMetrics(List metrics, string[] queues)
{
    foreach (var queue in queues)
    {
        var basePath = $"Custom Metrics|IBMQ|Queues|{queue}";

        var depth = Random.Shared.Next(0, 500);
        var oldestAge = depth == 0 ? 0 : Random.Shared.Next(5, 3600);
        var inRate = Random.Shared.Next(0, 120);
        var outRate = Math.Max(0, inRate + Random.Shared.Next(-20, 25));

        metrics.Add(Observation($"{basePath}|Depth", depth));
        metrics.Add(Observation($"{basePath}|OldestMessageAgeSeconds", oldestAge));
        metrics.Add(Observation($"{basePath}|MessagesInPerMinute", inRate));
        metrics.Add(Observation($"{basePath}|MessagesOutPerMinute", outRate));
    }
}

static void AddAzureServiceBusQueueMetrics(List metrics, string[] queues)
{
    foreach (var queue in queues)
    {
        var basePath = $"Custom Metrics|ASBQ|Queues|{queue}";

        var active = Random.Shared.Next(0, 1000);
        var deadLetter = Random.Shared.Next(0, 25);
        var scheduled = Random.Shared.Next(0, 100);
        var transfer = Random.Shared.Next(0, 10);
```

```
        metrics.Add(Observation($"{basePath}|ActiveMessages", active));
        metrics.Add(Observation($"{basePath}|DeadLetterMessages", deadLetter));
        metrics.Add(Observation($"{basePath}|ScheduledMessages", scheduled));
        metrics.Add(Observation($"{basePath}|TransferMessages", transfer));
    }
}

static AppDynamicsMetric Observation(string name, long value)
{
    return new AppDynamicsMetric
    {
        MetricName = name,
        AggregatorType = "OBSERVATION",
        Value = value
    };
}

public sealed class AppDynamicsMetric
{
    public required string MetricName { get; init; }
    public required string AggregatorType { get; init; }
    public long Value { get; init; }
}
```



Remote access:

```
docker run -d --name windows_vm \
  -p 8006:8006 -p 3389:3389/tcp -p 3389:3389/udp -p 8293:8293 \
  --device=/dev/kvm --cap-add=NET_ADMIN \
  -e VERSION="11" -e RAM_SIZE="4G" -e CPU_CORES="2" -e DISK_SIZE="64G" \
  -e HOST_PORTS="8293" \
  -v 3206d0feb37030d1a381c5a2e97c4fc80de74de27297a4ba19b267acc6f21179:/storage \
  --stop-timeout 120 dockurr/windows
```

```
New-NetFirewallRule -DisplayName "AppDynamics Agent Listener" -Direction Inbound -LocalPort 8293 -Protocol TCP -Action Allow
```