Using dotnet-counters

Introduction

Nethermind can be configured to publish its metrics usingSystem.Diagnostics.Metrics . This mechanism is a native tool embedded in .NET Platform. It allows for a low overhead monitoring and reporting. Once .NET Platform metrics are enabled, they can be monitored and collected usingdotnet-counters and other tools.

Configuration

Reporting metrics asSystem.Diagnostics.Metrics is enabled by passing and additional argument--Metrics.CountersEnabled true to the Docker containers,Nethermind.Runner orNethermind.Launcher e.g../Nethermind.Runner --Metrics.CountersEnabled true .

This flag can be configured separately fromsetting-up-local-metrics-infrastracture.md as this two reporting modes are treated separately.

Metrics names

Metrics reported by a Nethermind node follow the module convention. Whenever there's a moduleX , its metrics will be reported under meterNethermind.X For example,Evm module will be repoted underNethermind.Evm and so on.

dotnet-counters

dotnet-counters is a tool provided by the .NET team to monitor and collect metrics for further analysis. The usage of it is different when used on the same machine or in the Dockerized environment. To learn more about the tool, please visit the official documentation page ofmetrics collection with dotnet-counters .

Same machine

When a node is running on the same machine,dotnet-counters , given that the .NET runtime is already installed, can be installed with the following

dotnet tool install -g dotnet-counters This will install the tool globally and will allow the user to monitor and to collect metrics from any .NET process that is run on the same machine. For further information how to monitor and collect, please refer tothe original documentation of this command .

Docker image and docker compose

When running in a Dockerized environment, the most common way is to create a separate docker image for .NET diagnostics. This can be done with the followingDockerfile

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS base

RUN dotnet tool install -g dotnet-counters; \ dotnet tool install -g dotnet-trace; \ echo 'export PATH="PATH:/root/.dotnet/tools"' >> /root/.bashrc

ENTRYPOINT ["/bin/bash"] Once it's built, asdotdiag image, it will enable runningdotnet-counters from within.

The second part is connecting the dockerized node with thedotdiag . Whether usingdocker copose or images run manually, it's important to remember thatdotnet-counters communicate over a named pipe (Windows) or an IPC socked (Linux, macOS). To make it work, volume mapping should be provided so that the two images share the directory used for the communication. Similarlypid namespace needs to be shared between them.

Let's visit an extract of adocker-compose.yaml that would provide such configuration.

version: "3.9"

services:

execution: stop_grace_period: 30s container_name: execution-client restart: unless-stopped image: IMAGE_VERSION_GOES_HERE networks: - sedge volumes: - ./dotnet-tmp:/tmp # /tmp is used to create the IPC socket, expose it as ./dotnet-tmp ports:

ports omitted as they are not changed

command:

make counters enabled so that reporting happens by setting the flag

- --Metrics.CountersEnabled=true logging: driver: "json-file" options: max-size: "10m" max-file: "10"

the created dotdiag

dotdiag: container_name: dotdiag image: dotdiag stdin_open: true # docker run -i, so that it runs tty: true # docker run -t, so that it runs volumes: - ./dotnet-tmp:/tmp # map to the same directory, to make IPC socket connection pid: "service:execution" # make pid namespaces are shared - processes are visible depends_on: - execution # make the dependency explicit Edit this page Last updatedonFeb 17, 2024 Previous Metrics Next setting-up-local-metrics-infrastracture