Inspired by Aspire

Meetup, Michal Lukáč
27.2.2025

OpenTelemetry

OpenTelemetry (informally called OTEL or OTel) is an observability framework – software and tools that assist in generating and capturing telemetry data from cloud-native software.

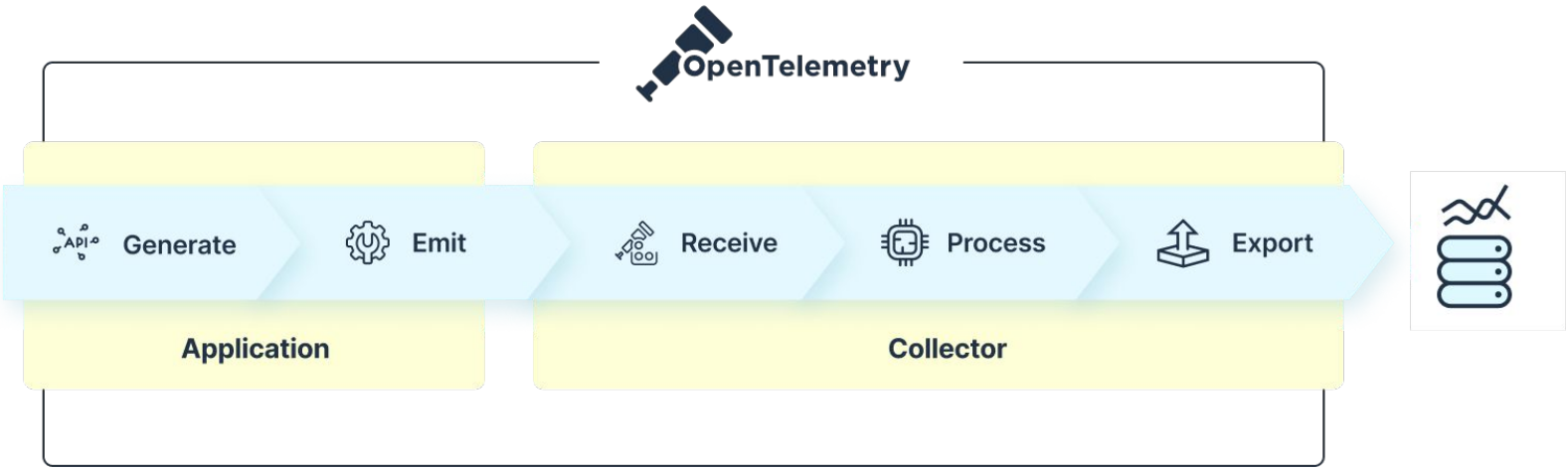
- Traces
- Metrics
- Logs

Components, most notably:

- **APIs and SDKs** per programming language for generating and emitting telemetry
- **Collector** component to receive, process and export telemetry data
- **OTLP protocol** for transmitting telemetry data

CLOUD NATIVE COMPUTING FOUNDATION

OpenTelemetry



Observability

- Built in metrics with dimensions
- DI integration for metrics
- Better Logging support (faster, can object serialization)
- Enrichment
- Redaction
- Testing fakes for Logging & Metrics

Resiliency

- New Polly based resiliency packages
- SignalR Stateful Reconnect

Scalability

- AOT (increased density)
- Performance
- Chiseled Ubuntu

Manageability

- Certificate auto-rotation support in Kestrel

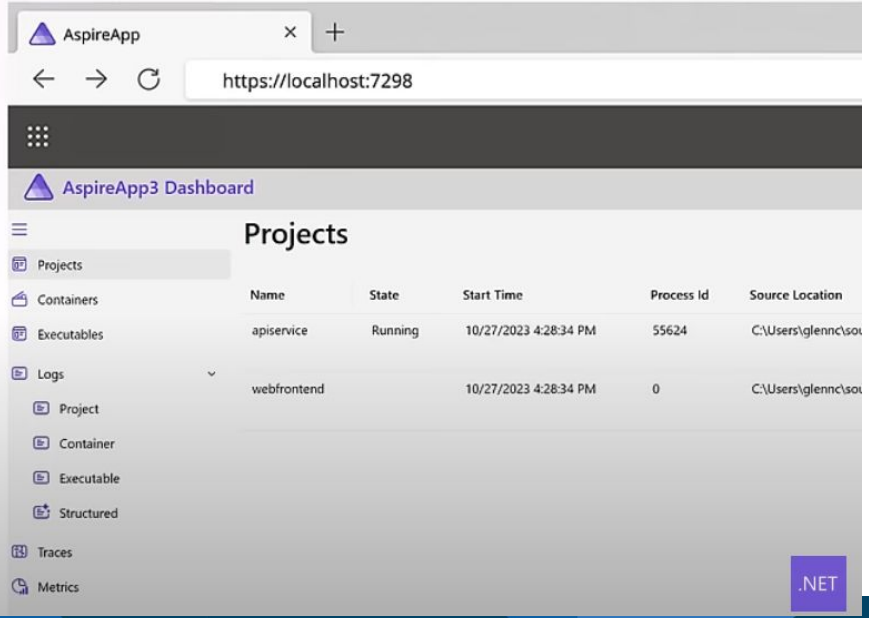
.NET Aspire

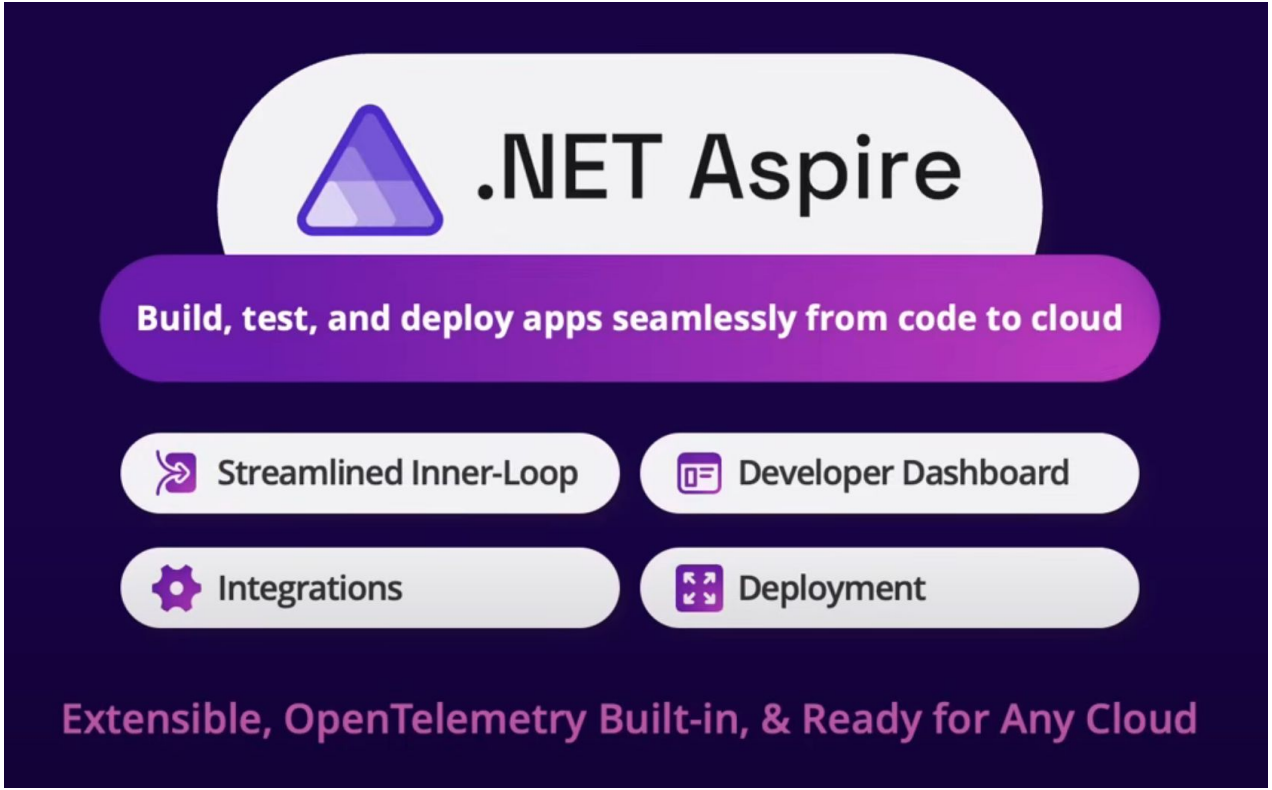
- .NET Aspire is a set of tools, templates, and packages for building observable, production ready apps.
  a. Dev-time orchestration (enhancing the local development)
     - set of abstractions that streamline the setup of service discovery, environment variables, and container configurations, eliminating the need to deal with low-level implementation details
  b. .NET Aspire integrations
  c. Project templates and tooling

.NET Aspire

Developer Dashboard

- Structured Logs
- Metrics
- Distributed Traces
- Dependencies




.NET Aspire
Build, test, and deploy apps seamlessly from code to cloud
Streamlined Inner-Loop
Developer Dashboard
Integrations
Deployment
Extensible, OpenTelemetry Built-in, & Ready for Any Cloud

How to Aspire ?

very simply

- Initiation via dotnet cli
  a. dotnet new aspire
  b. dotnet new aspire-starter

- Creates a couple of super projects, run AppHost orchestrator project
- [.NET Aspire documentation | Microsoft Learn]

Starter project

AppHost project

ServiceDefaults project

Directory

Copy

```
AspireSample
├── AspireSample.ApiService
│   ├── Properties
│   │   └── launchSettings.json
│   ├── appsettings.json
│   ├── AspireSample.ApiService.csproj
│   └── Program.cs
├── AspireSample.AppHost
│   ├── Properties
│   │   └── launchSettings.json
│   ├── appsettings.json
│   ├── AspireSample.AppHost.csproj
│   └── Program.cs
├── AspireSample.ServiceDefaults
│   ├── AspireSample.ServiceDefaults.csproj
│   └── Extensions.cs
├── AspireSample.Web
│   ├── integrations
│   │   ├── Layout
│   │   │   ├── ...
│   │   ├── Pages
│   │   │   ├── ...
│   │   ├── ...
│   ├── Properties
│   │   └── launchSettings.json
│   ├── wwwroot
│   │   ├...
│   ├── appsettings.json
│   ├── AspireSample.Web.csproj
│   ├── Program.cs
│   └── WeatherApiClient.cs
└── AspireSample.sln
```

AppHost project

- Single point of reference
- Constructing application by adding containers and projects
- Adding references to containers to projects
- No connection strings, url, ports - all resolved by Aspire orchestrating project
- Kind of replacement for docker-compose
- run AppHost to run the app
- Running AppHost will launch the orchestrator which will launch the containers and projects and Aspire Dashboard

```
<IsAspireHost>true</IsAspireHost>
```

ServiceDefaults project

- AddServiceDefaults ext method
- Called from every single application to configure
  - Telemetry
  - Healthchecks
  - Service discovery
  - Resilience
- Yours to customize

Basic telemetry setup

```
string aspireEndpoint = "http://localhost:4317"; // Default for local
// 🚀 Configure OpenTelemetry
builder.Services.AddOpenTelemetry()
    .ConfigureResource(resource => resource
        .AddService("UserManagementAPI")
    )
    .WithTracing(tracing => tracing
        .AddAspNetCoreInstrumentation() // Capture HTTP requests
        .AddEntityFrameworkCoreInstrumentation() // Capture DB Queries from EF Core
        .AddHttpClientInstrumentation() // Track outgoing HTTP requests
        .AddConsoleExporter() // Logs traces to Console (debugging)
        .AddOtlpExporter(opts =>
        {
            opts.Endpoint = new Uri(aspireEndpoint);
            opts.Protocol = OpenTelemetry.Exporter.OtlpExportProtocol.Grpc;
        }) // Export to Aspire/OpenTelemetry
    )
    .WithMetrics(metrics => metrics
        .AddAspNetCoreInstrumentation()
        .AddRuntimeInstrumentation()
        .AddConsoleExporter()
        .AddOtlpExporter(opts =>
        {
            opts.Endpoint = new Uri(aspireEndpoint);
            opts.Protocol = OpenTelemetry.Exporter.OtlpExportProtocol.Grpc;
        }) // Export Metrics
    );
```

Aspire Dashboard

Aspire Dashboard is a lightweight observability UI for .NET applications, built into the Aspire Cloud-Native Application framework from Microsoft. It provides real-time insights into your application's telemetry data, such as logs, traces, metrics, dependencies, and health checks — all in one place.

🎯 Key Purpose

Simplifies debugging & tracing in distributed applications

Provides deep insights into application behavior, performance, and errors

Works seamlessly with OpenTelemetry, Prometheus, and Jaeger

Lightweight, running locally or in container environments

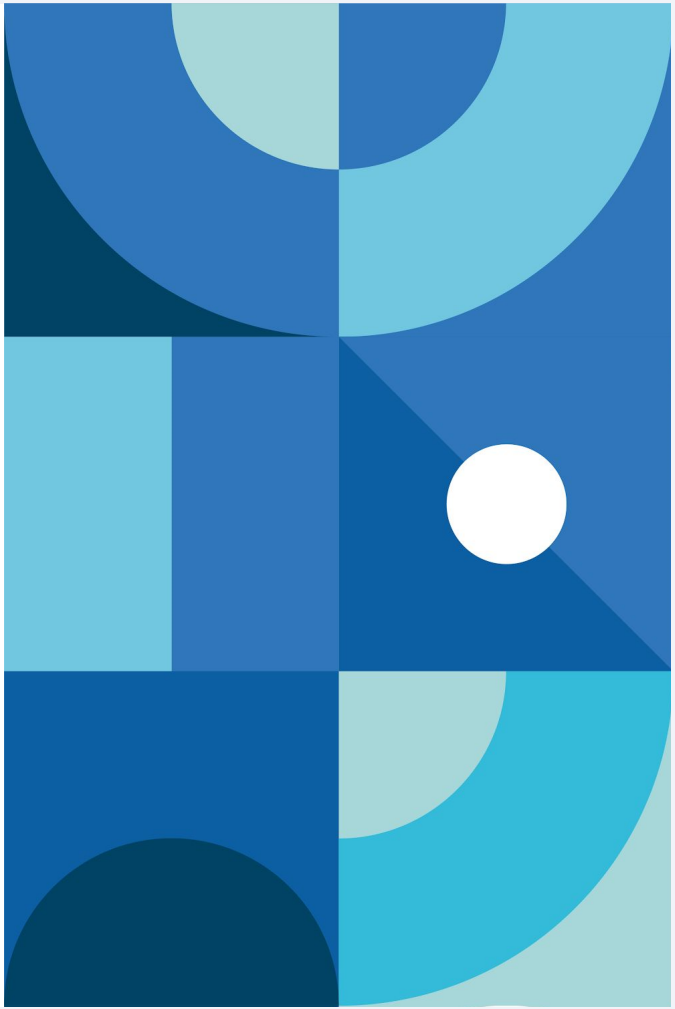
What it can do

1️⃣ Structured Logs
Logs are essential for understanding what's happening in an application.

✔ Displays structured logs that your application generates
✔ Supports OpenTelemetry logging, capturing TraceId, SpanId, and log level
✔ Logs can include custom fields & metadata, such as user ID, request IDs, etc.
✔ Supports log search and filtering for debugging easy event traces

✨ Supports structured log exporters:

Console (builder.Logging.AddConsole())
OTLP (Aspire Dashboard itself captures OTLP logs)
File Logging or Cloud Exporters (Azure Monitor, AWS CloudWatch, etc.)

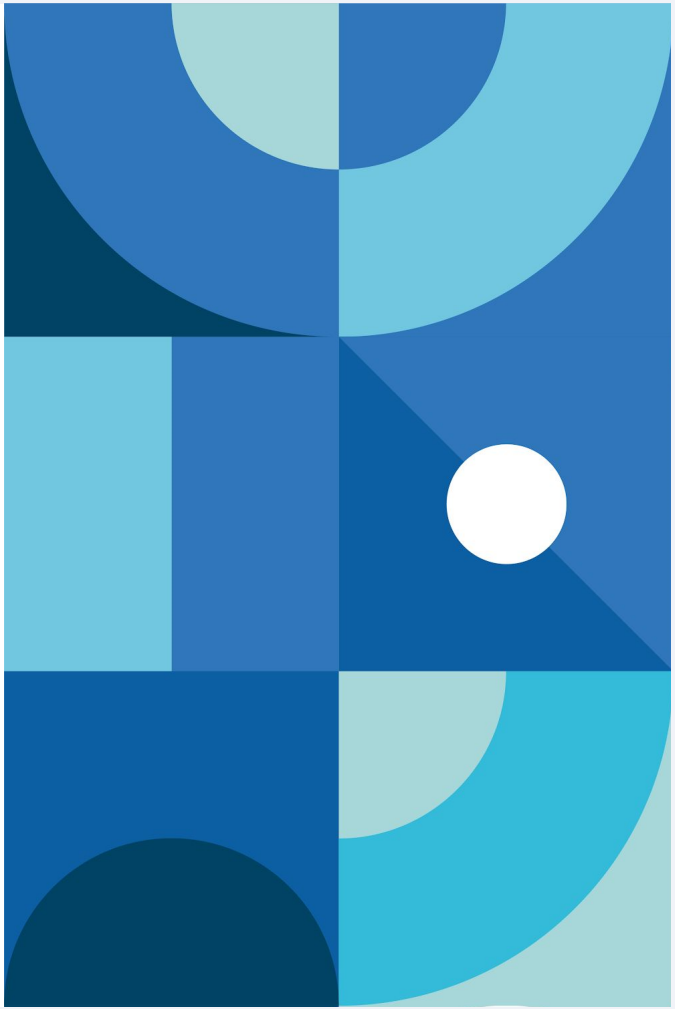


What it can do

2️⃣ Traces 🔍
Application tracing is critical for debugging distributed apps.
✔ Uses OpenTelemetry Tracing 📡
✔ Automatically captures request timelines, including API & DB queries
✔ Supports Events inside traces (e.g., "DB Query Started", "Auth Passed")
✔ Links traces to logs & errors for contextual debugging
✔ Enables Backlinks to monitoring systems such as Prometheus or Grafana

✨ Supports Tracing Exporters:

Jaeger, Zipkin, Azure Application Insights
OTLP Export (Sent to Aspire, Prometheus, or external stores)

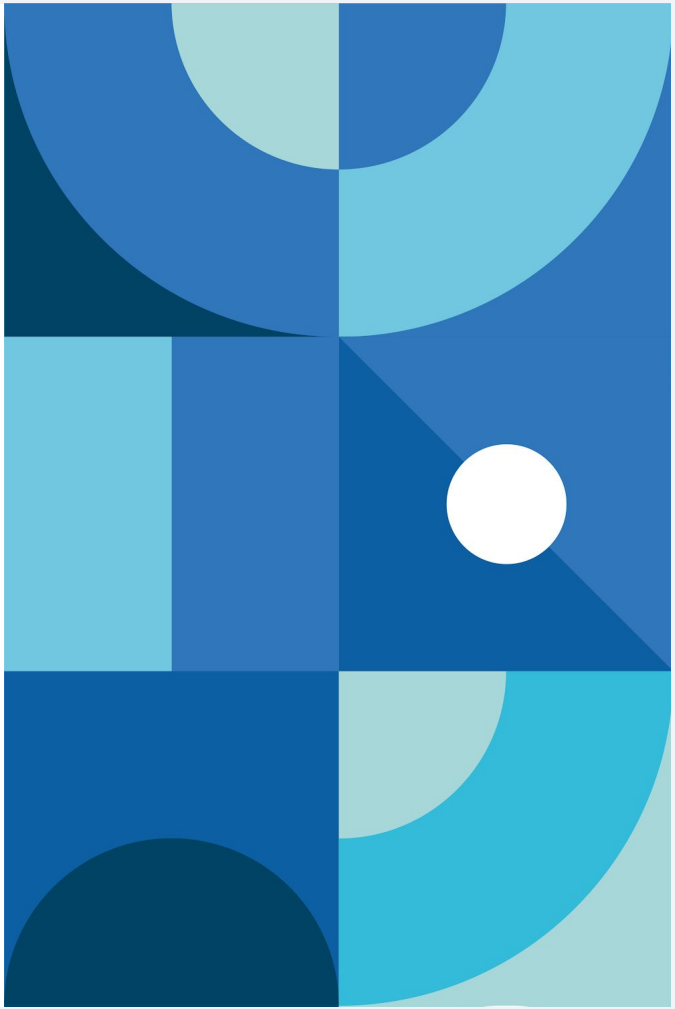


What it can do

3️⃣ Metrics 📊 (Live System Performance)
Performance metrics show how well an application is running.
✔ Captures OpenTelemetry Metrics (OTLP & Prometheus)
✔ Tracks custom metrics like users_created, api_requests_total, etc.
✔ Displays live charts & data refresh rates
✔ Allows filtering & aggregation of metrics
✔ Supports adding Prometheus/Grafana Actions

✨ Supports Metric Exporters:

Prometheus (Grafana-ready visuals)
Azure Monitor / AWS CloudWatch
Custom Visualization Dashboards

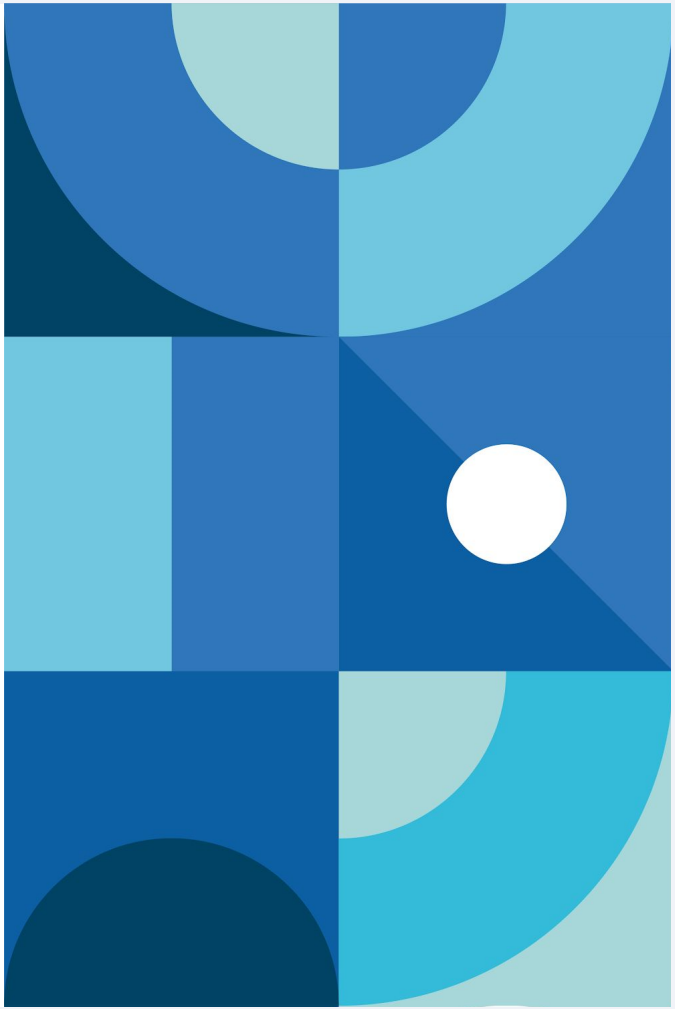


What it can do

4️⃣ Dependencies 🔗 (Service-to-Service Monitoring)
Need to monitor microservices, message queues, or API calls? 🚀
Aspire tracks service dependencies and their health!
✔ Monitors service-to-service communication (HTTP, gRPC, SQL, Kafka, Redis, etc.)
✔ Tracks external API calls & database latency metrics
✔ Helps visualize bottlenecks inside distributed systems
✔ Automatically links to traces & logs

✨ Supports Monitoring Dependencies for:

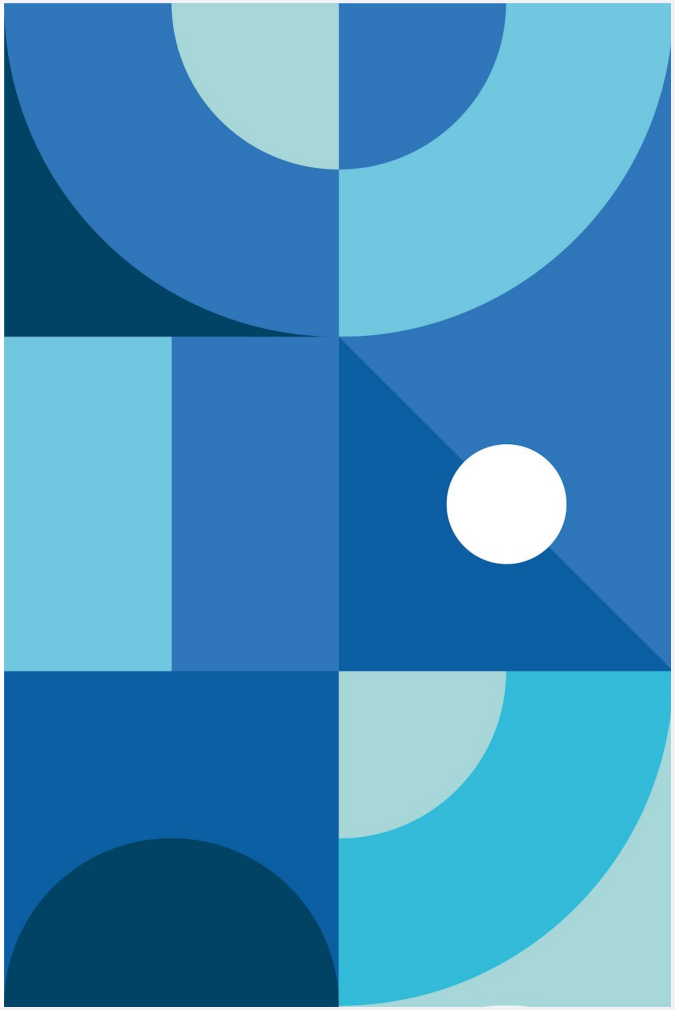
SQL Server, PostgreSQL, SQLite, MySQL
Redis, RabbitMQ
gRPC & REST APIs



What it can do

5️⃣ Health Checks 🏥
✔ Integrated Health-Check UI for /health endpoints
✔ Shows application service statuses (alive, degraded, or failing)
✔ Supports custom checks (e.g., DB connection status, queue backlogs)
✔ Works with .NET Health Checks API and Aspire's built-in health rules

✨ Aspire Desktop UI automatically tests & reports on /health APIs.

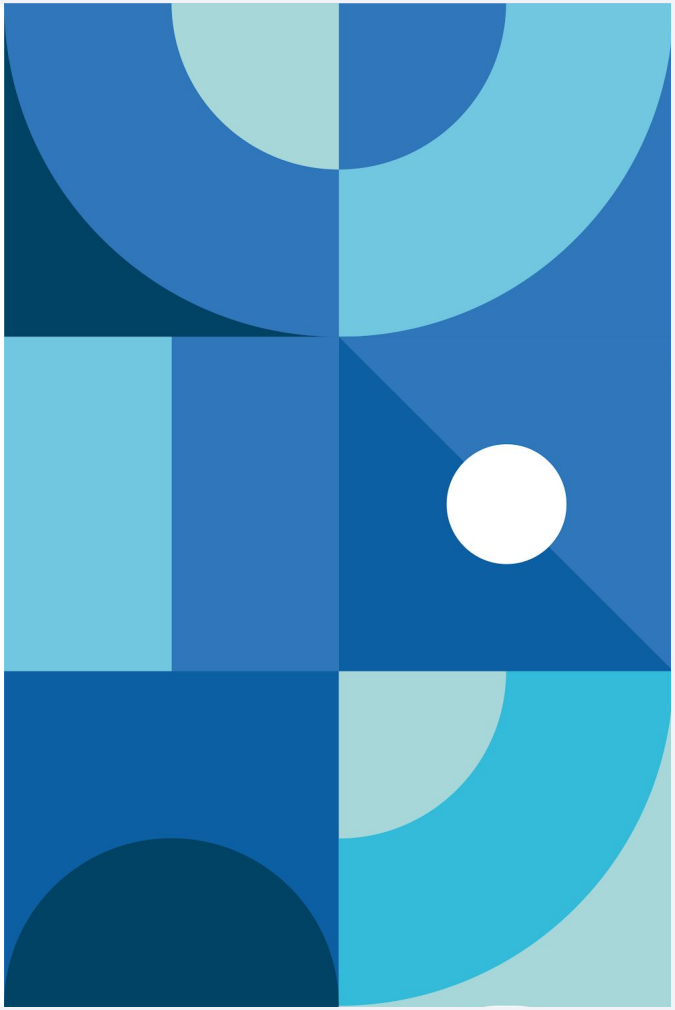


What it can do

6️⃣ Custom Actions in Aspire UI ⚡
Aspire lets you add custom actions to open external tools for deeper analysis.

✔ Open metrics in Prometheus
✔ Open traces in Jaeger
✔ Link logs to custom dashboards
✔ Open error details in Stackdriver / Kibana / Azure Monitor

✨ Aspire Desktop UI automatically tests & reports on /health APIs.



What it CANNOT do

❌ No Long-Term Storage
Aspire does not retain logs, traces, or metrics beyond its runtime session.
You must export data to Prometheus, Jaeger, or Azure Monitor for long-term analytics.
❌ Not a Production-Grade Observability Stack
It's mainly for local development & debugging 🏗️
Aspire relies on OpenTelemetry, but for large-scale analytics, you should use Grafana / DataDog / Azure Monitor
❌ No Built-in Machine Learning/Alerting
Unlike Datadog or AWS CloudWatch, Aspire doesn't support AI-driven anomaly detection.
❌ Limited UI Customization
While you can add custom actions, Aspire UI is not customizable like Grafana Dashboards.
❌ Only Works with .NET
Aspire Dashboard is tied to the Aspire Stack and OpenTelemetry for .NET
If you need cross-platform observability, better use Jaeger, Prometheus, or Grafana

Aspire

Structured

Traces

Metrics

Telemetry endpoint is unsecured Untrusted apps can send telemetry to the dashboard. More information

UserManagementAPI-9a862f71: GET /users

7f317c4

Trace detail **2/26/2025 10:06:58.303 PM** Duration **69.05ms** Resources **1** Depth **2** Total spans **2**

View logs

Name	0ms	17.26ms	34.53ms	69.05ms
- UserManage...				
UserMana...				512.8µs

UserManagementAPI-9a862f71: DATA sqlite main bed1efd

Resource **UserManagementAPI** Duration **512.8µs** Start time **66.05ms**

View logs

Filter...

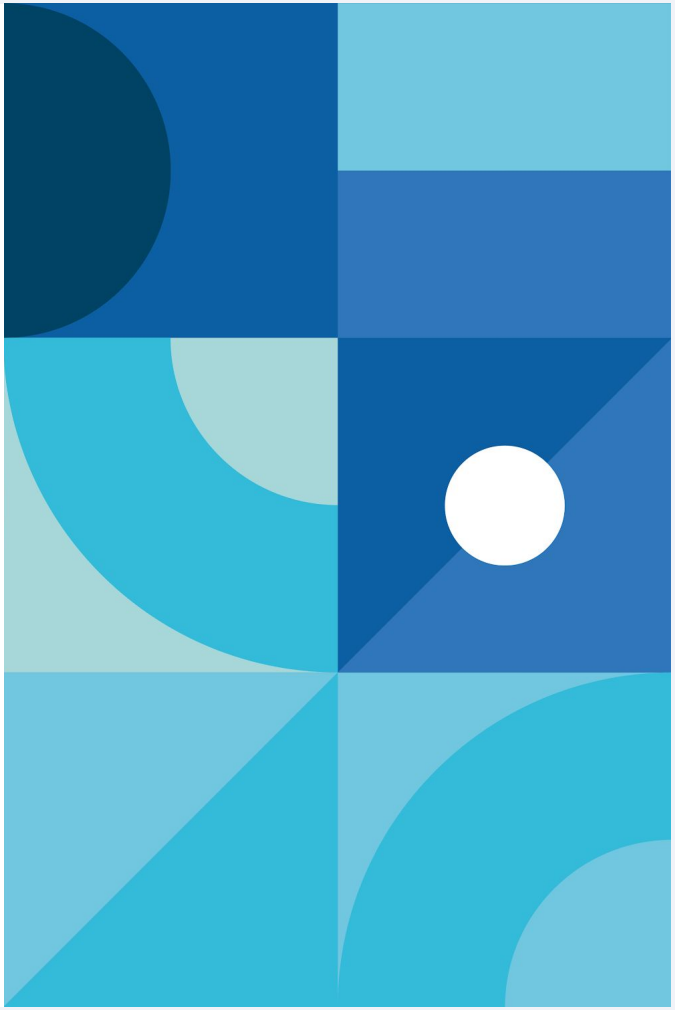
Span

6

Name	Value
SpanId	bed1efdcddc28cd8
Name	main
Kind	Client
db.name	main
db.system	sqlite

standalone Aspire Dashboard

Container image:
mcr.microsoft.com/dotnet/aspire-dashboard:9.0



“Let’s look at examples ”

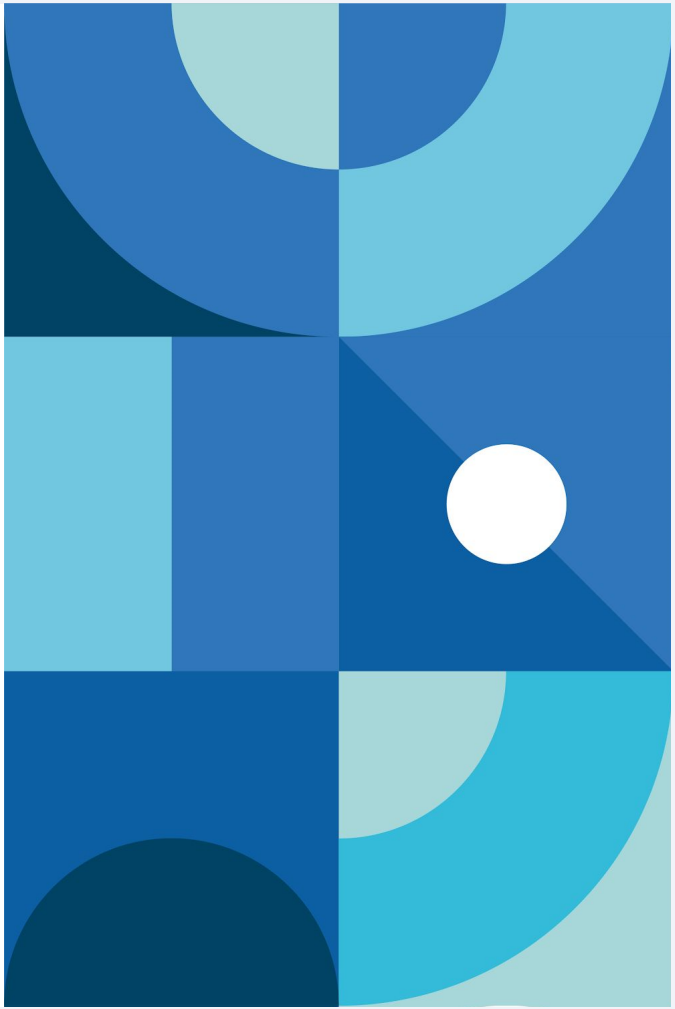
additional options

Additional context can be added using:

📍 Events → Timed logs within a trace (e.g., "User created", "Processing started")
🎯 represent important timed moments inside a trace (e.g., "DB queried", "User validated")

🔗 Links → Relate traces together (e.g., a background job connected to an API request).

🔙 Backlinks → External references pointing back to the trace ID (used in distributed tracing).



Add Events to Traces

1️⃣ Each /users request gets a trace.

2️⃣ The trace records Events:

- 🔹 User API Received Request
- ✅ User Data Validated
- 💾 User Saved to Database"

3️⃣ Now Aspire Dashboard will show "Events" inside the trace! 🎯

```
// ✅ Add Events to Traces
app.MapPost("/users", async (AppDbContext db, User user) =>
{
    using var activity = tracer.StartActivity("Create User");

    activity?.AddEvent(new ActivityEvent("🔹 User API Received Request"));

    if (string.IsNullOrWhiteSpace(user.Email))
    {
        activity?.AddEvent(new ActivityEvent("⚠️ Validation Failed: Email Missing"));
        throw new Exception("Email is required");
    }

    activity?.AddEvent(new ActivityEvent("✅ User Data Validated"));

    db.Users.Add(user);
    await db.SaveChangesAsync();

    activity?.AddEvent(new ActivityEvent("💾 User Saved to Database"));

    return Results.Created($"/users/{user.Id}", user);
});
```

Add Links Between Related Traces

🎯 Links 🔗 allow tracing operations across multiple traces and services.
If a background job processes a user request, you can link the two traces together.

✔ The background job links back to the original API request.

✔ Aspire Dashboard will now show "Links" when you click on traces! 🔗

```
app.MapPost("/process-job", async (HttpContext context) =>
{
    using var mainActivity = tracer.StartActivity("Job Triggered");
    mainActivity?.AddEvent(new ActivityEvent("⚙️ Background Job Started"));

    // ✅ Simulate receiving a trace from another request
    string? parentTraceId = context.Request.Headers["traceparent"];

    if (!string.IsNullOrEmpty(parentTraceId))
    {
        var relatedActivity = new Activity("Job Processing")
            .SetParentId(parentTraceId)
            .Start();

        relatedActivity?.AddEvent(new ActivityEvent("🔗 Linked to Original Request"));

        mainActivity?.AddLink(new ActivityLink(relatedActivity.Context)); // ✅ Add a l
    }

    return Results.Ok("Job processed.");
});
```

Add Backlinks (External References)

🎯 Backlinks 🔙 help cross-reference traces from external systems (e.g., link traces to logs or monitoring IDs).

💡 If your system integrations, such as cloud storage, log aggregators, or monitoring tools use an ID, you can link it back to OpenTelemetry.

✔ Aspire Dashboard will now show "Backlinks" in trace details

✔ Clicking the link can navigate users to an external monitoring page

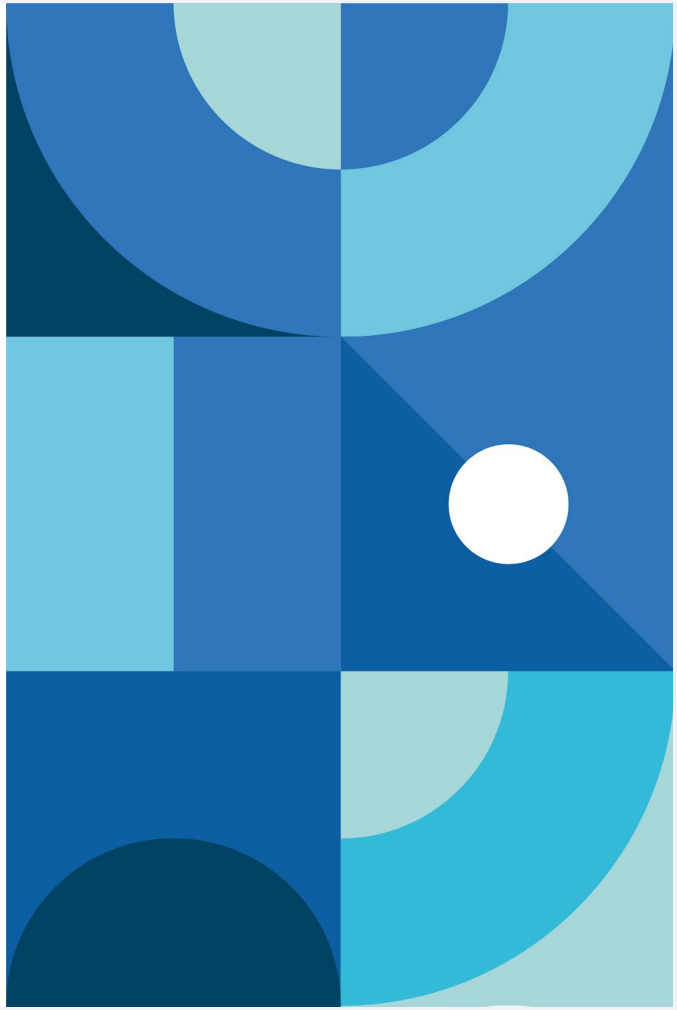
```
app.MapGet("/external-task", async (HttpContext context) =>
{
    using var activity = tracer.StartActivity("External Task");

    activity?.AddEvent(new ActivityEvent("🔹 Task Execution Started"));

    // 🔙 ✅ Add Backlink to Monitoring Tool (e.g., CloudWatch, Azure Monitor)
    string externalReferenceId = Guid.NewGuid().ToString();
    activity?.SetTag("backlink_id", externalReferenceId);
    activity?.SetTag("monitoring_url", $"https://monitoring.example.com/traces/{externa

    await Task.Delay(500); // Simulate work
    activity?.AddEvent(new ActivityEvent("✅ Task Execution Completed"));

    return Results.Ok("Task completed.");
});
```

🎯 Final Summary

✅ Events → In-trace logs (activity?.AddEvent(...))

✅ Links → Connect related traces (activity?.AddLink(...))

✅ Backlinks → External references (activity?.SetTag(...))



Add Custom Actions to Aspire Dashboard

appsettings.json

json Copy code

```
{
  "Logging": {
    "Console": {
      "IncludeScopes": true,
      "FormatterName": "json"
    }
  },
  "Aspire": {
    "Dashboard": {
      "CustomActions": [
        {
          "Name": "🔍 View in External Monitoring",
          "UrlTemplate": "https://monitoring.example.com/traces/{traceId}",
          "ApplyTo": ["Traces"]
        },
        {
          "Name": "📂 Open in Log Aggregator",
          "UrlTemplate": "https://logs.example.com/search?q={spanId}",
          "ApplyTo": ["Logs"]
        }
      ]
    }
  }
}
```

It’s evolving

.. so many samples, github repos, package names, suggested chatbot snippets & cli commands are not working :(

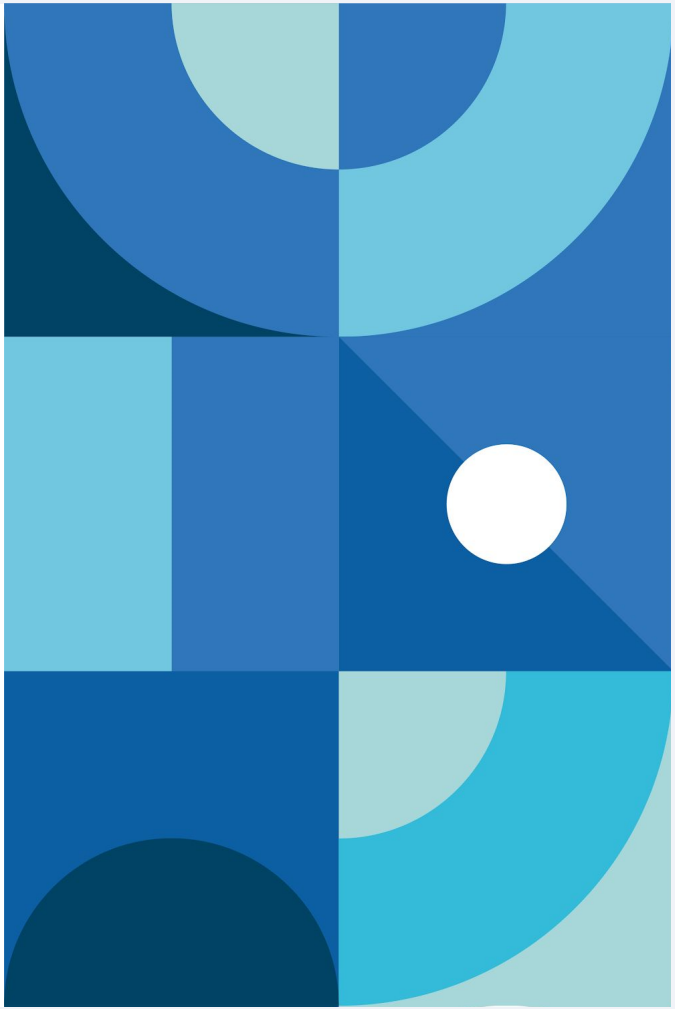
Aspire deployment to Azure

Azure CLI automatically recognizes the Aspire project and is able to handle & deploy

Using Azure CLI

- az init
- az up

Do you Aspire to try it out ? :)



resources

- .NET Aspire documentation - https://learn.microsoft.com/en-us/dotnet/aspire/

https://github.com/dotnet/aspire-samples

- .NET Aspire Workshop - https://github.com/dotnet-presentations/dotnet-aspire-workshop
- Beginner's Guide to OpenTelemetry - link
- eShop https://github.com/dotnet/eshop
- Aspire & AWS https://github.com/aws/integrations-on-dotnet-aspire-for-aws/blob/main/src/Aspire.Hosting.AWS/README.md
- Standalone Aspire dashboard

https://learn.microsoft.com/en-us/dotnet/aspire/fundamentals/dashboard/standalone?tabs=bash

https://learn.microsoft.com/en-us/samples/dotnet/aspire-samples/aspire-standalone-dashboard/

- OpenTelemetry dotnet open-telemetry/opentelemetry-dotnet: The OpenTelemetry .NET Client
- https://devblogs.microsoft.com/dotnet/announcing-dotnet-8/
- https://www.nuget.org/packages/OpenTelemetry.Exporter.OpenTelemetryProtocol
- .NET Aspire dashboard security considerations - .NET Aspire | Microsoft Learn

Aspire samples by MS https://github.com/dotnet/aspire-samples/tree/main

- https://github.com/dotnet-presentations/dotnet-aspire-workshop

THANKS

Q & A ?