.NET

云原生开发挑战赛

暨.NET20周年特别活动-51Aspx

Upgrade to Dotnet6.0

赛事流程&奖项

大赛时间：2022.3.15——2022.6.15

报名：即可获得500积分+微软亲签的证书

扫描二维码进群:直播讲师课后入群答疑+直播讲师课后入群答疑

赛事奖项：

一等奖，1名：51Aspx平台.NET20周年奖杯、10000元奖金。
二等奖，3名：51Aspx平台.NET20周年奖杯、4000元奖金。
三等奖，5名：51Aspx平台.NET20周年奖杯、2000元奖金。
特别奖，1名：51Aspx平台创新基金支持+5000元奖金。
参与奖，报名参赛成功前100名用户可获得参与奖，奖励51Aspx平台一年VIP会员。

云原生开发挑战赛

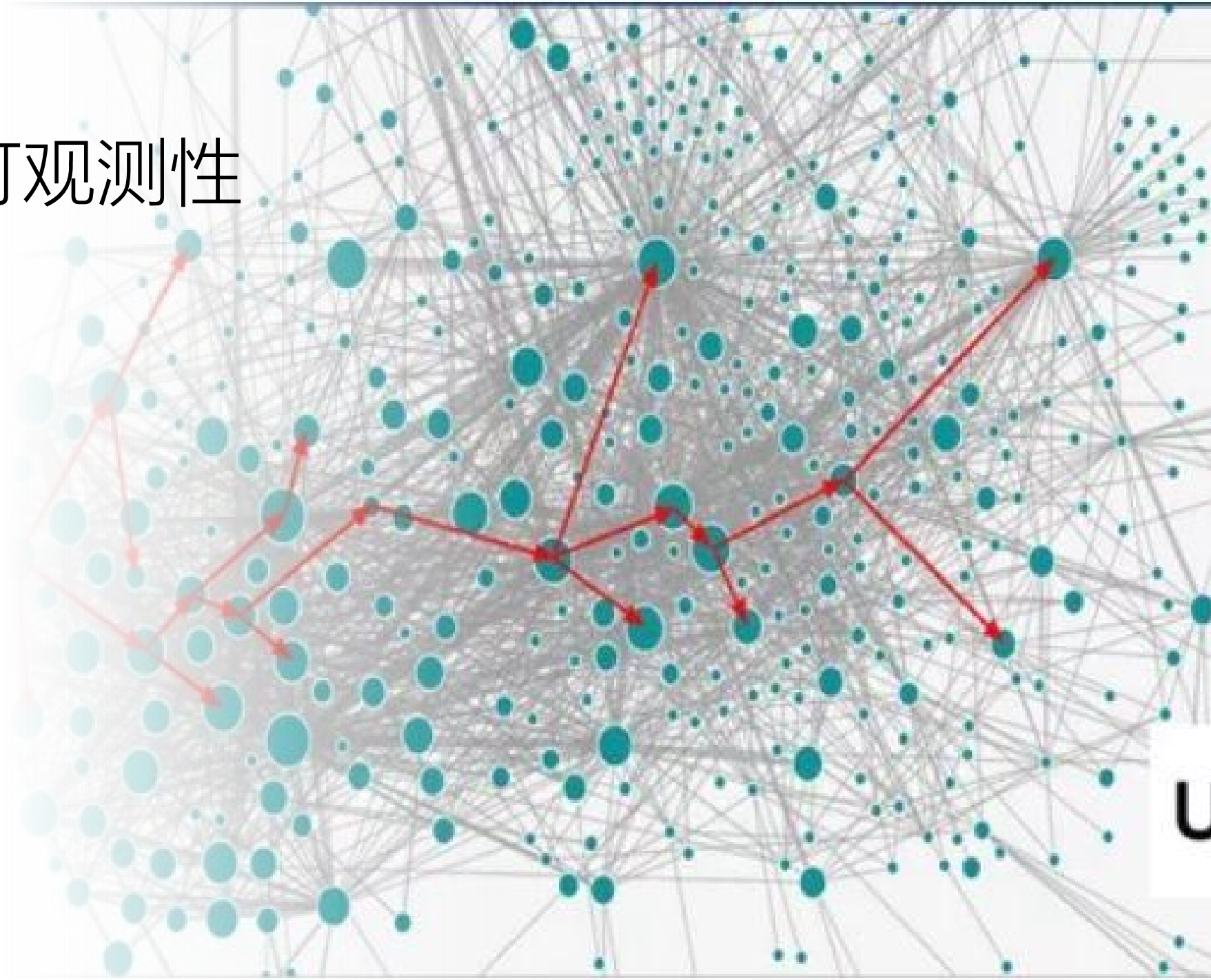
- 微软最有价值专家 郝冠军

背景

- 云原生架构的特性
  - Modularity 模块化
  - Observability 可观察性
  - Deployability 可部署性
  - Testability 可测试性
  - Disposability 可处理性
  - Replaceability 可替换性

云原生开发挑战赛

为什么需要可观测性



Observability vs Monitoring

- Observability
  - Focus on Application
  - Developer

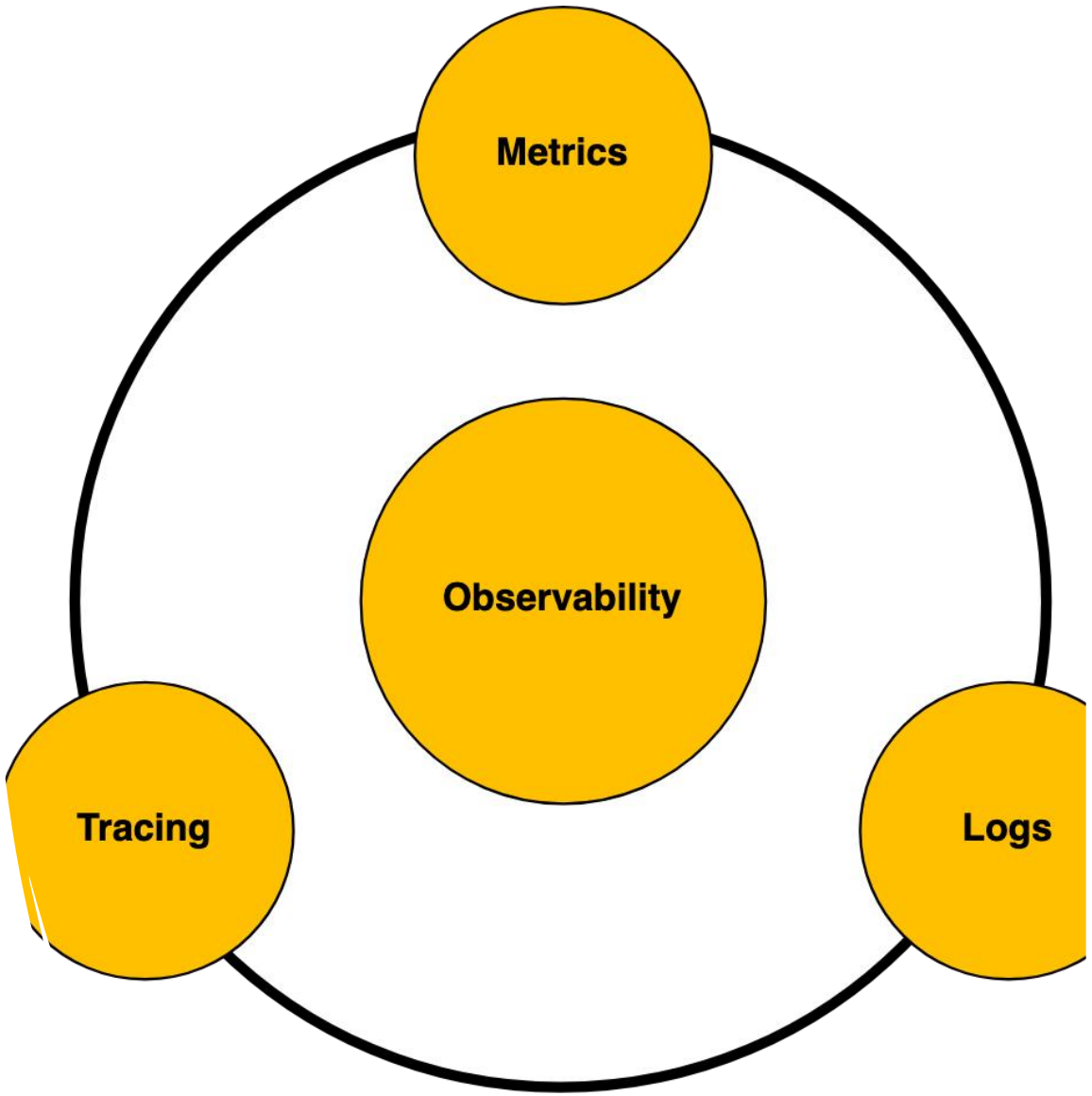
- Monitoring
  - Focus infrastructure
  - IT

- [Observability vs Monitoring: What's The Difference - THCBin Tech Blog]

云原生开发挑战赛

可观测性的 3 个维度

- Tracing
- Metrics
- Logging



Metrics, logs, 与 traces

- Metrics
  - Aggregated summary statistics.
- Logs
  - Detailed debugging information emitted by processes.
- Distributed Tracing
  - Provides insights into the full lifecycles, aka traces of requests to a system, allowing you to pinpoint failures and performance issues.

Structured data can be transmuted into any of these!

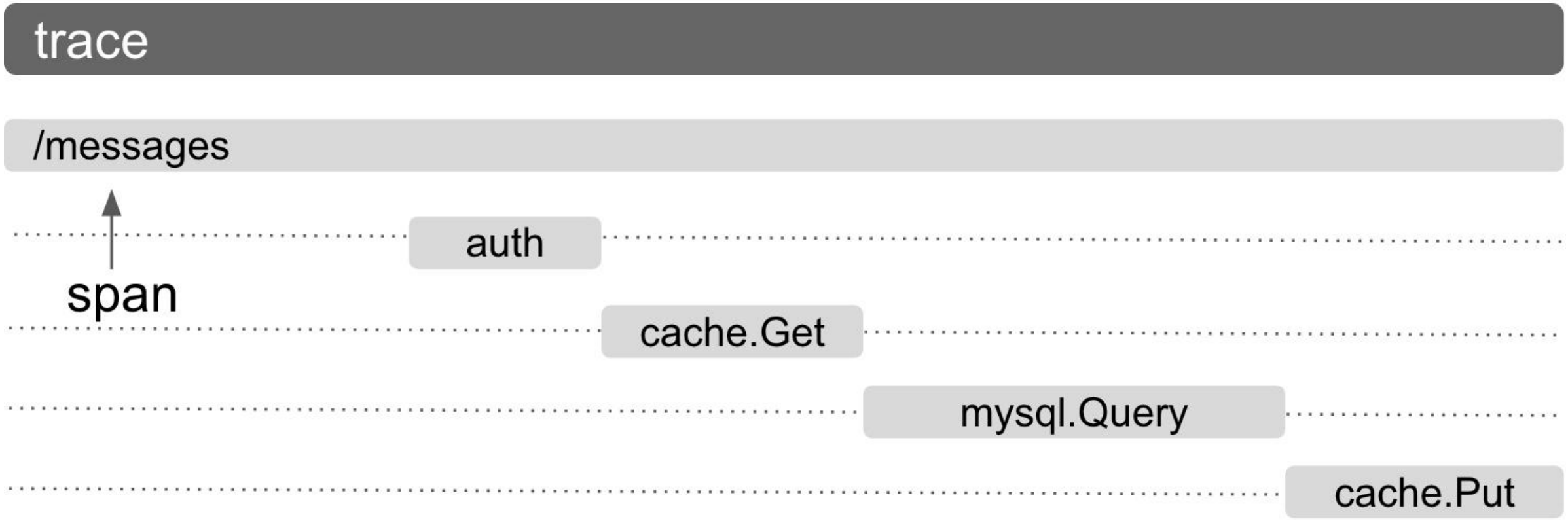
Tracing 概念

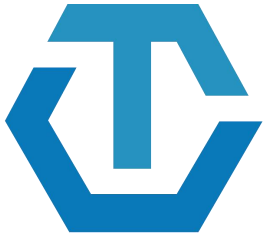
- Span
  - Represents a single unit of work in a system.
  - Typically encapsulates: operation name, a start and finish timestamp, the parent span identifier, the span identifier, and context items.

- Trace
  - Defined implicitly by its spans. A trace can be thought of as a directed acyclic graph of spans where the edges between spans are defined as parent/child relationships.

- DistributedContext
  - Contains the tracing identifiers, tags, and options that are propagated from parent to child spans

Tracing concepts





OPENTRACING

- 370 Contributors
- 5360 stars on GitHub
- 100s of supported Integrations with OSS libraries and frameworks

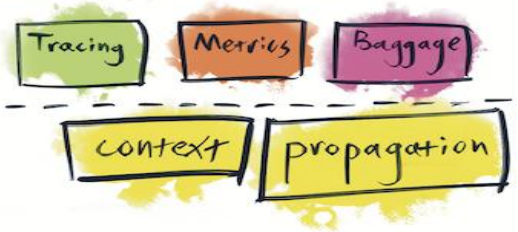
- 390 Contributors
- 3392 stars on GitHub
- Backed by Google, Microsoft, Omnition, Postmates, Dynatrace

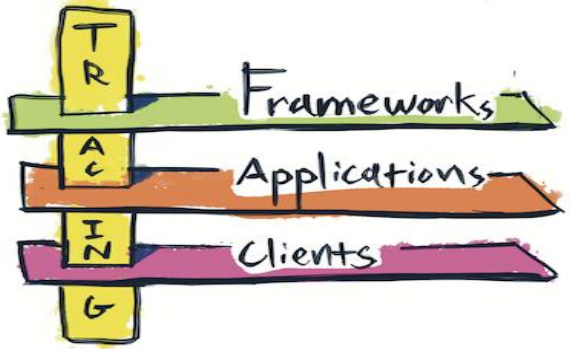
OpenTracing and OpenCensus have merged form OpenTelemery!

OpenTelemetry

OpenTelemetry Architecture

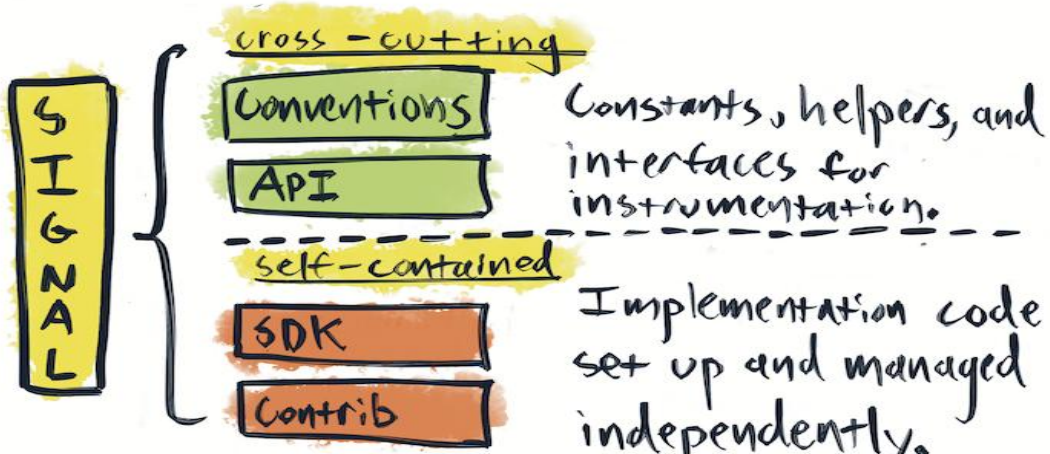
OpenTelemetry is designed as a set of independent observability tools, called Signals, which are built on top of a shared mechanism for context propagation.



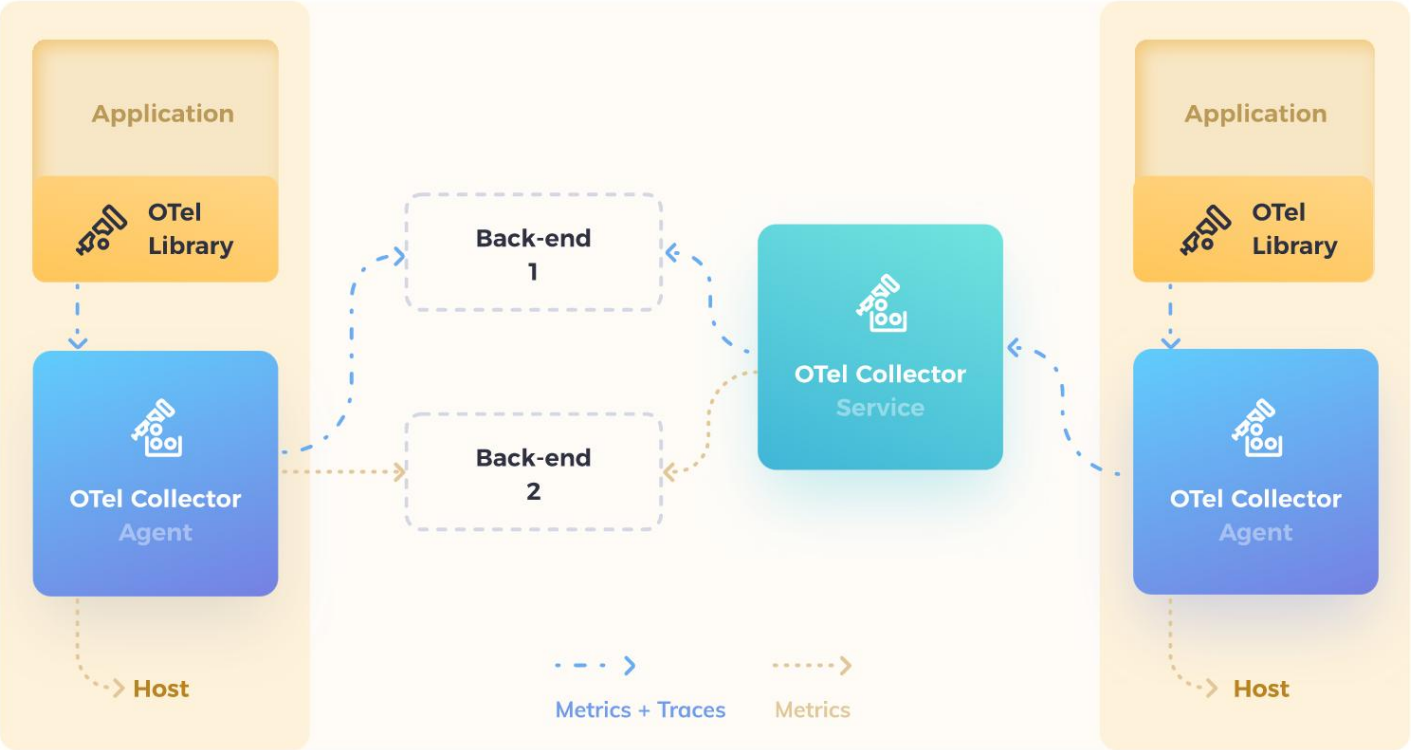


Signals work as cross-cutting concerns, mixed into many libraries.

The portion of each signal which is imported as a cross-cutting concern is kept seperate from the portion which can be managed independently by the application owner.



OpenTelemetry



REFERENCE ARCHITECTURE

OpenTelemetry .NET Status and Releases

- OpenTelemetry .NET API
  - *The API only surfaces necessary abstractions (抽象) to instrument an application/library*
- OpenTelemetry .NET SDK
  - *OpenTelemetry SDK is a reference implementation (实现) of the OpenTelemetry API.*
- 状态
  - Traces：稳定
  - Metrics： 稳定
  - Logging：
    - Ilogger 稳定

See: https://opentelemetry.io/docs/net/

云原生开发挑战赛

Tracing

- Attribute
  - Available in current span.
- Event
  - As a log in span
- Baggage 传播
  - *Baggage can be propagated out of proc (传播出进程之外) using Propagators.*
  - *OpenTelemetry SDK ships a BaggagePropagator and enables it by default.*
  - https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/src/OpenTelemetry.Api/README.md#baggage-api

OpenTelemetry.API vs .NET Activity API

- System.DiagnosticSource NuGet Package
- Span => Activity
- Attribute => Tag
  - Tags in Activity represents the OpenTelemetry Span Attributes.
- Event => Event
- Baggage => Baggage

See: https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/src/OpenTelemetry.Api/README.md#

入门

```
using System.Diagnostics;

using OpenTelemetry;
using OpenTelemetry.Trace;
using OpenTelemetry.Resources;

// 定义重要的用于初始化 Trace 的常量
var serviceName = "MyCompany.MyProduct.MyService";
var serviceVersion = "1.0.0";

// 配置重要的 OpenTelemetry 设置，以及输出到控制台
using var tracerProvider = Sdk.CreateTracerProviderBuilder()
    .AddSource(serviceName)
    .SetResourceBuilder(
        ResourceBuilder.CreateDefault()
            .AddService(serviceName: serviceName, serviceVersion: serviceVersion))
    .AddConsoleExporter()
    .Build();

var MyActivitySource = new ActivitySource(serviceName);

using var activity = MyActivitySource.StartActivity("SayHello");
activity?.SetTag("foo", 1);
activity?.SetTag("bar", "Hello, World!");
activity?.SetTag("baz", new int[] { 1, 2, 3 });
```

云原生开发挑战赛

自动遥测 (Instrumentation)

- 使用预定义的遥测库 (*Instrumentation library*) 完成遥测

```
// 配置重要的 OpenTelemetry 设置，以及输出到控制台，
// 配置自动遥测
builder.Services.AddOpenTelemetryTracing(b =>
{
    b
    .AddConsoleExporter()
    .AddSource(serviceName)
    .SetResourceBuilder(
        ResourceBuilder.CreateDefault()
            .AddService(serviceName: serviceName, serviceVersion: serviceVersion))
    .AddHttpClientInstrumentation()
    .AddAspNetCoreInstrumentation();
});
```

云原生开发挑战赛

遥测库（Instrumentations）

- OpenTelemetry.Instrumentation.AspNet
  - OpenTelemetry.Instrumentation.AspNet.TelemetryHttpModule
- OpenTelemetry.Instrumentation.AspNetCore
- OpenTelemetry.Instrumentation.Http
- OpenTelemetry.Instrumentation.GrpcNetClient
- OpenTelemetry.Instrumentation.SqlClient
- OpenTelemetry.Instrumentation.StackExchangeRedis

手动遥测

```
// 为应用命名服务名称.
// 可以通过配置文件、常量等等.
public static readonly ActivitySource MyActivitySource = new(TelemetryConstants.ServiceName);

public static void DoWork()
{
    using var parentActivity = MyActivitySource.StartActivity("ParentActivity");

    // 遥测父 activity
    parentActivity?.SetTag("operation.value", 1);
    parentActivity?.SetTag("operation.name", "Saying hello!");
    parentActivity?.SetTag("operation.other-stuff", new int[] { 1, 2, 3 });

    using (var childActivity = MyActivitySource.StartActivity("ChildActivity"))
    {
        // 同一函数中的子 activity
    }

    // 重新回到父 activity
}
```

Exporters

- OpenTelemetry.Exporter.Console
- OpenTelemetry.Exporter.InMemory
- OpenTelemetry.Exporter.Jaeger
- OpenTelemetry.Exporter.OpenTelemetryProtocol.Logs
- OpenTelemetry.Exporter.OpenTelemetryProtocol
- OpenTelemetry.Exporter.Prometheus
- OpenTelemetry.Exporter.Zpages
- OpenTelemetry.Exporter.Zipkin

传播

- DistributedContextPropagator

Demo

云原生开发挑战赛

End