The State of Observability in .NET

Tracing .NET applications in the Cloud

Gergely Kalapos
Elastic

Gergely Kalapos

Web: http://kalapos.net
Twitter: @gregkalapos

- Works on observability and performance monitoring tools at Elastic
- Loves OpenSource, works in a distributed team
- Born in Hungary, lives in Austria

Slides and sample code

https://bit.ly/2ksf65E

Agenda

- Observability overview
- DiagnosticSource
- W3C TraceContext
- OpenTelemetry
- Elastic APM, Application Insights, Jaeger for .NET developers

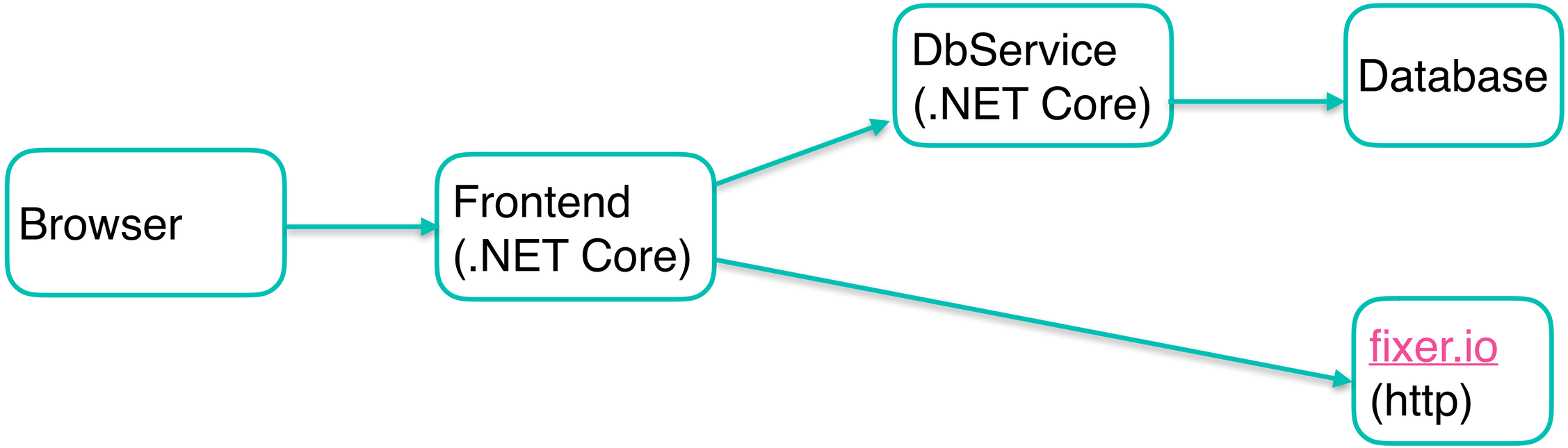
Goals

- Overview about the observability landscape in .NET for developers
- Standards
- Open Source libraries
- Be able to research and decide what you need in order to have observability for your production system

Non-Goals

- Overview about a single product
- Giving a tutorial-style talk about a single product

Demo application

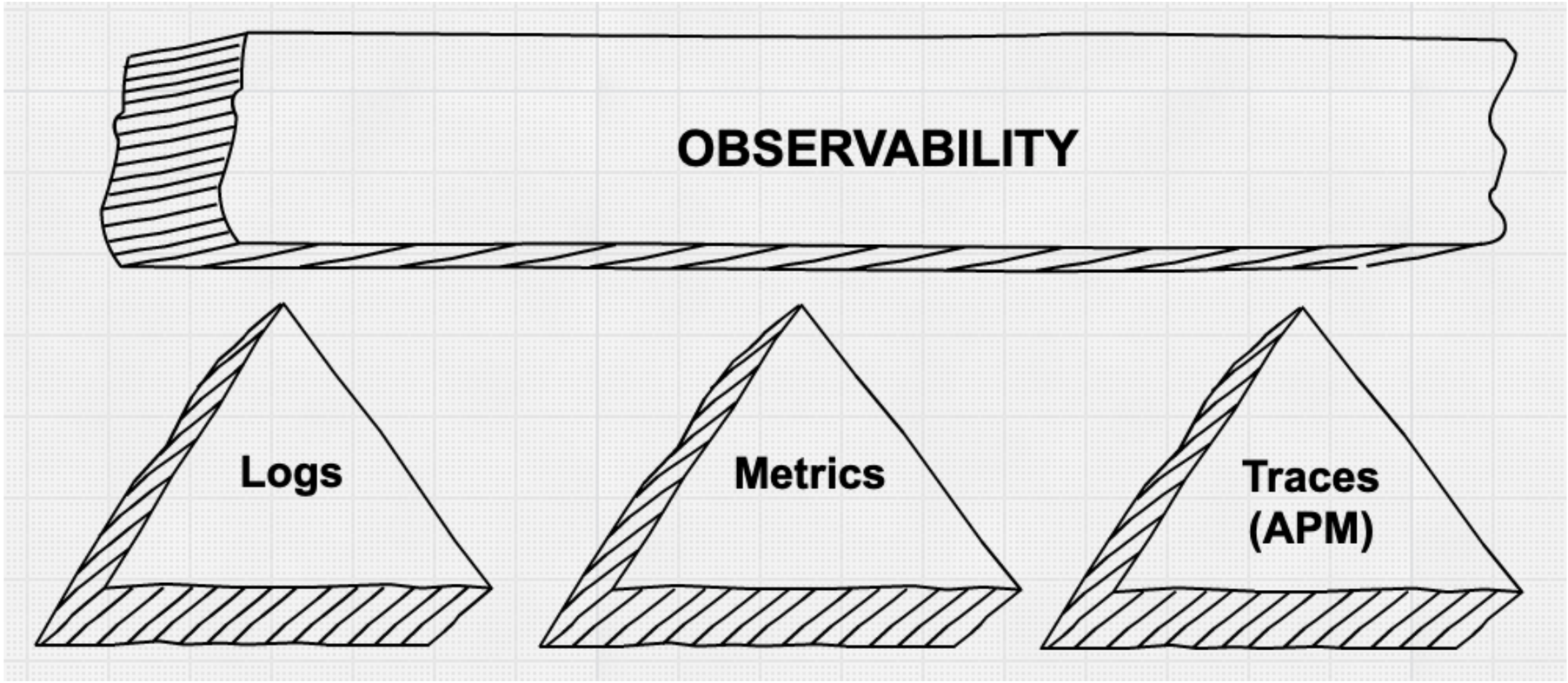


Demo

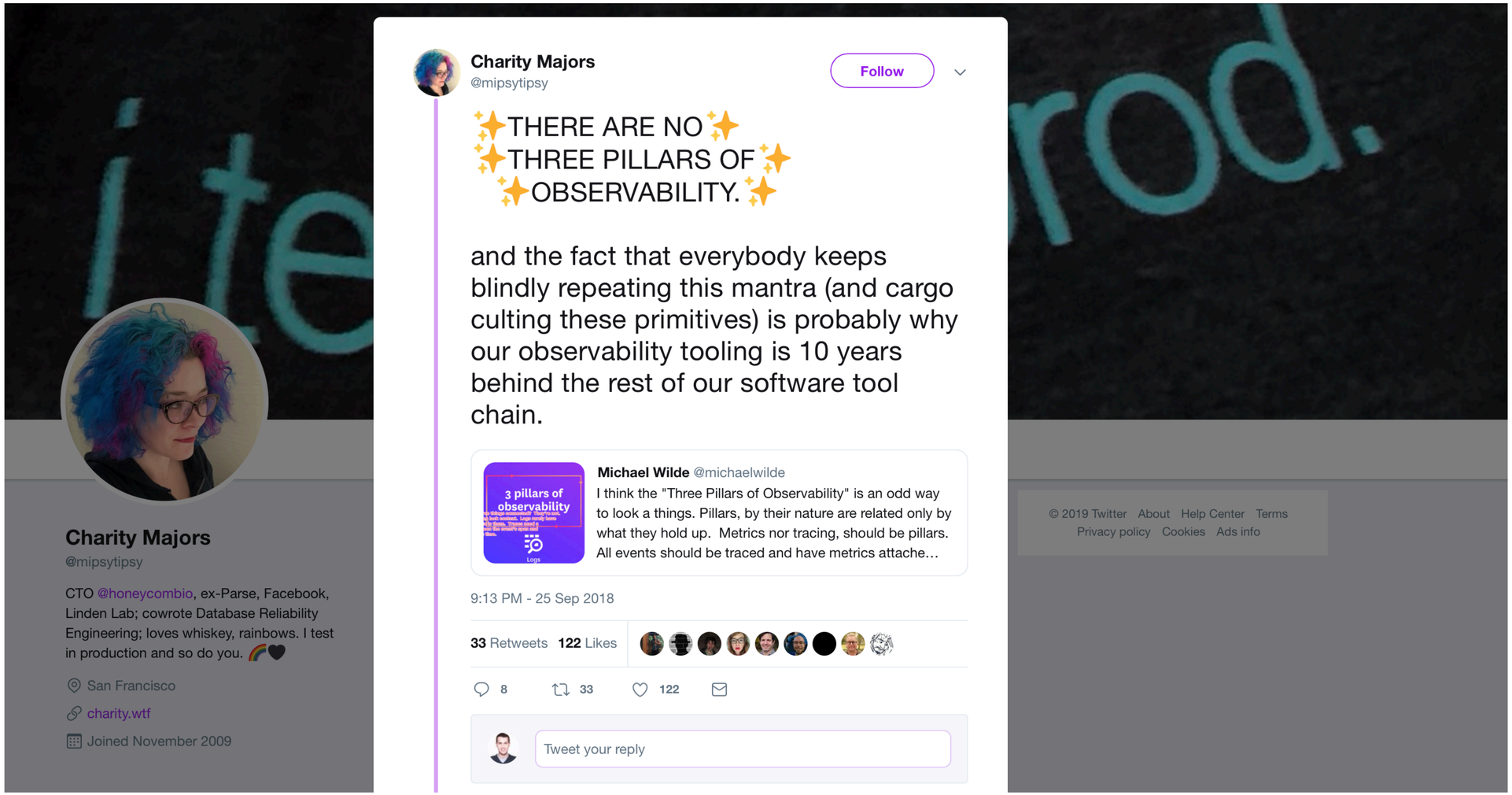
A system is observable when you can ask arbitrary questions about it and receive meaningful answers without having to write new code or command line tools.

It lets you discover unknown-unknowns and debug in production.

3 Pillars of Observability



3 Pillars of Observability



Source: https://twitter.com/mipsytipsy/status/1044666259898593282?lang=en

Pillar or not, I honestly don’t care, just show me all logs and metrics to a specific trace!

Instrumentation with DiagnosticSource & Activity

Diagnostic Source

- Part of the .NET (Core) Framework (System.Diagnostics.DiagnosticSource)
- Instrument code for production time logging of rich data payloads
- 2 Capabilities:
  - Write events to it
  - Listen for events and use the data
- Works with objects -> structured data
- Create metrics based on it, observability for your system

Part 1

Instrument with DiagnosticSource

```
class MyLibrary
{
    private readonly static DiagnosticSource _diagnosticSource = new DiagnosticListener("MyLibrary");

    public double CalculateAverage(IEnumerable<double> items)
    {
        if (_diagnosticSource.IsEnabled("MyLibrary")
            _diagnosticSource.Write("CalculateAverageStarted", new { Items = items });


        var avgValue = items.Average();


        if (_diagnosticSource.IsEnabled("MyLibrary")
            _diagnosticSource.Write("CalculateAverageFinished", new { Items = items, Result = avgValue });

        return avgValue;
    }
}
```

Part 2

Receiver

```
DiagnosticListener.AllListeners.Subscribe(new Subscriber());


class Subscriber : IObserver<DiagnosticListener>
{
    public void OnNext(DiagnosticListener listener)
    {
        if(listener.Name == "MyLibrary")
            listener.Subscribe(new MyLibraryObserver());
    }
}
```

```
class MyLibraryObserver : IObserver<KeyValuePair<string, object>>
{
    public void OnNext(KeyValuePair<string, object> kv)
    {
        switch (kv.Key)
        {
            case "CalculateAverageStarted":
                if (kv.Value.GetType().GetTypeInfo().GetDeclaredProperty("Items").GetValue(kv.Value) is IEnumerable<double> items)
                {
                    Console.WriteLine("CalculateAverageFinished - items:");
                    foreach (var item in items)
                        Console.WriteLine(item);
                }
                break;

            case "CalculateAverageFinished":
                if (kv.Value.GetType().GetTypeInfo().GetDeclaredProperty("Result").GetValue(kv.Value) is double result)
                    Console.WriteLine($"CalculateAverageFinished - result: {result}");
                break;
        }
    }
}
```

Demo

DiagnosticSource in HttpClient

Demo

HttpClient - DiagnosticSource

- Sends events when the HTTP Request starts and stops
- System.Net.HttpRequest/System.Net.HttpResponse: Deprecated
- System.Net.Http.HttpRequest.Start/
  System.Net.Http.HttpRequest.Stop: Based on Activity, use these

DiagnosticSource Used by

- HttpClient
- ASP.NET Core: Incoming HTTP requests, etc.
- MVC: Routing, Controller, Action, etc.
- Entity Framework Core: DB Request started, finished, etc.
- Lots of Azure libraries

Activity

- Store and access diagnostic context
- Integrated with DiagnosticSource
- Has an ID
- Hierarchical - non-root activates have a parent ID
- Tags: represents context needed for logging - local to the given activity
- Baggage: represents information to be logged with the activity and passed to its children

Activity

```
var activity = new Activity(“MyActivity");

_diagnostics.StartActivity(activity, new  { Property1 = prop1, Property2 = prop2 });

//Do work – normal application/library code

activity.AddBaggage("MyBaggageId", value);
activity.AddTag("MyTagId", value);

_logger.Log(activity.Id);

_logger.Log(activity.ParentId);


_diagnostics.StopActivity(activity,id);
```

Activity.Current

- Points to the currently active activity which flows with async calls and is available during request processing

- DiagnosticSource.StartActivity(myActivity): sets myActivity to Activity.Current

- DiagnosticSource.StopActivity(myActivity): clears Activity.Current

```
class MyLibraryListener : IObserver<KeyValuePair<string, object>>
{
    public void OnNext(KeyValuePair<string, object> receivedEvent)
    {
        var currentActivity = Activity.Current;

        var tags = currentActivity.Tags;

        var id = currentActivity.Id;
        var parentId = currentActivity.ParentId;
        //…
    }
}
```

Demo

DiagnosticSource & Activity
How to put the data to AI/Elastic APM or any other tool?

- Application Insights:
  - TelemetryClient
    - TrackDependency(…)
    - TrackEvent(…)
    - TrackException(…)
    - TrackPageView(…)
    - TrackRequest(…)
    - TrackTrace(…)

```
class MyLibraryObserver : IObserver<KeyValuePair<string, object>>
{
        private TelemetryClient _telemetryClient;
        public MyLibraryObserver(TelemetryClient telemetryClient) => _telemetryClient = telemetryClient;
        public void OnNext(KeyValuePair<string, object> kv)
        {
            switch (kv.Key)
            {
                case "CalculateAverage.Start":
                    _telemetryClient.TrackEvent("CalculateAverageAsync.Start");
                    break;
                case "CalculateAverage.Stop":
                    var properties = new Dictionary<string, string>();
                    if (Activity.Current != null)
                    {
                            foreach (var tag in Activity.Current.Tags)  properties.Add(tag.Key, tag.Value);
                      }
                    if (kv.Value.GetType().GetTypeInfo().GetDeclaredProperty("Result").GetValue(kv.Value) is double result)
                        properties.Add("Result", result.ToString());
                    _telemetryClient.TrackEvent("CalculateAverageAsync.Stop", properties);
                    break;
                default:
                    break;
            }
        }
}
```

DiagnosticSource & Activity
How to put the data to AI/Elastic APM or any other tool?

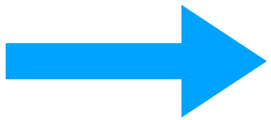
- Elastic APM:
  - Public Agent API: https://www.elastic.co/guide/en/apm/agent/dotnet/current/public-api.html
  - Agent.Tracer.StartTransaction()
  - Agent.Tracer.CurrentTransaction.StartSpan()

```
class MyLibraryObserver : IObserver<KeyValuePair<string, object>>
{
    readonly ConcurrentDictionary<string, ISpan> _concurrentDictionary = new ConcurrentDictionary<string, ISpan>();

    public void OnNext(KeyValuePair<string, object> kv)
    {
        switch (kv.Key)
        {
            case "CalculateAverage.Start":
                var transaction = Agent.Tracer.CurrentTransaction;
                if (transaction != null && Activity.Current.Id != null)
                {
                    var span = transaction.StartSpan("CalculateAverage", "MyLib");
                    _concurrentDictionary.TryAdd(Activity.Current.Id, span);
                }

                break;

            case "CalculateAverage.Stop":

                if (Activity.Current != null &&
                    _concurrentDictionary.Remove(Activity.Current.Id, out ISpan currentSpan))
                {
                    if (kv.Value.GetType().GetTypeInfo().GetDeclaredProperty("Result").GetValue(kv.Value) is double result)

                        currentSpan.Labels["Result"] = result.ToString();

                    foreach (var tag in Activity.Current.Tags)
                        currentSpan.Labels[$"tag - {tag.Key}"] = tag.Value;

                    currentSpan.End();
                }

                break;
            default:
                break;
        }
    }
}
```

Demo

Activity vs. direct usage of the library

- Shouldn't we just do this?

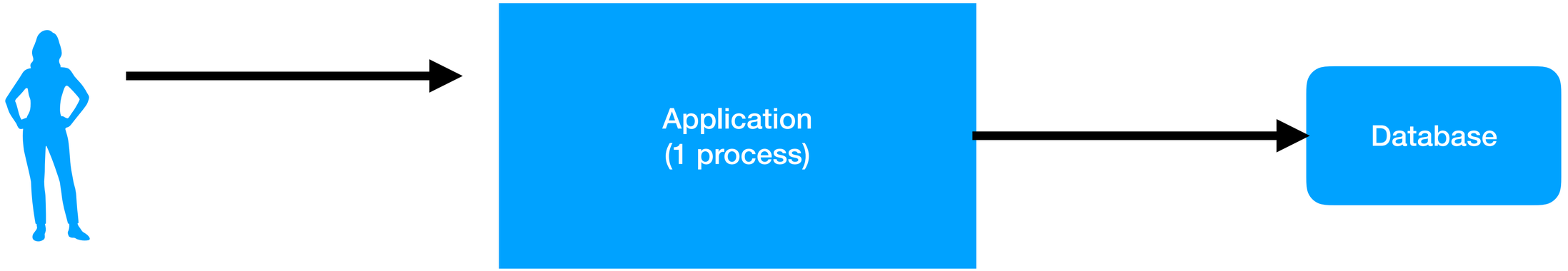


```
class MyLibrary
{
    public double CalculateAverage(IEnumerable<double> items)
    {
     var span = Agent.Tracer.CurrentTransaction.StartSpan(nameof(CalculateAverage), "CustomSpan");
     try
     {
            var avgValue = items.Average();
            span.Labels["result"] = avgValue;
            return avgValue;
     }
     catch (Exception e) {  span.CaptureException(e); throw;   }
     finally { span.End();  }
    }
}
```

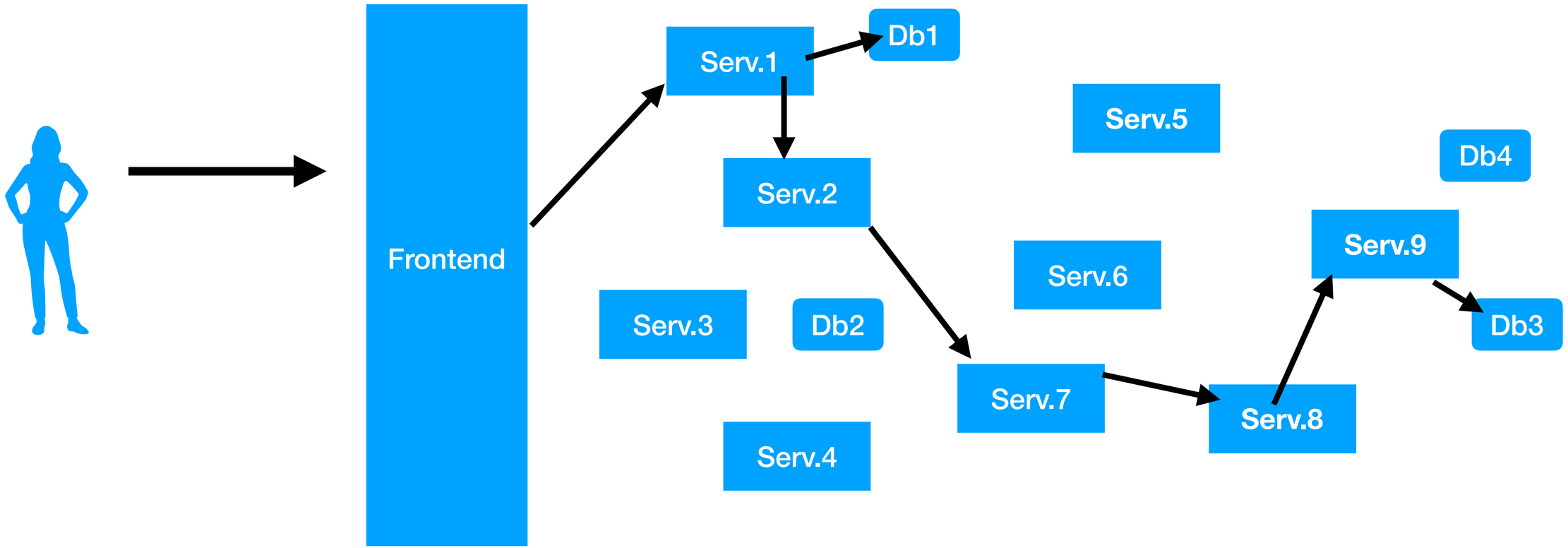
Activity vs. direct usage of AI/OT etc.

- It depends!
- Using DiagnosticSource&Activity and bridging - **Pros**:
  - Part of the framework - no extra dependency needed
  - Designed and maintained by Microsoft
  - Very light-weight and small
  - No vendor lock-in
  - DiagnosticSource is there since .NET Core 1.0 and with high probability it won't go away (other tools sometimes don't even reach version 1.0)
- Using DiagnosticSource&Activity and bridging - **Cons**:
  - Additional overhead of the bridging

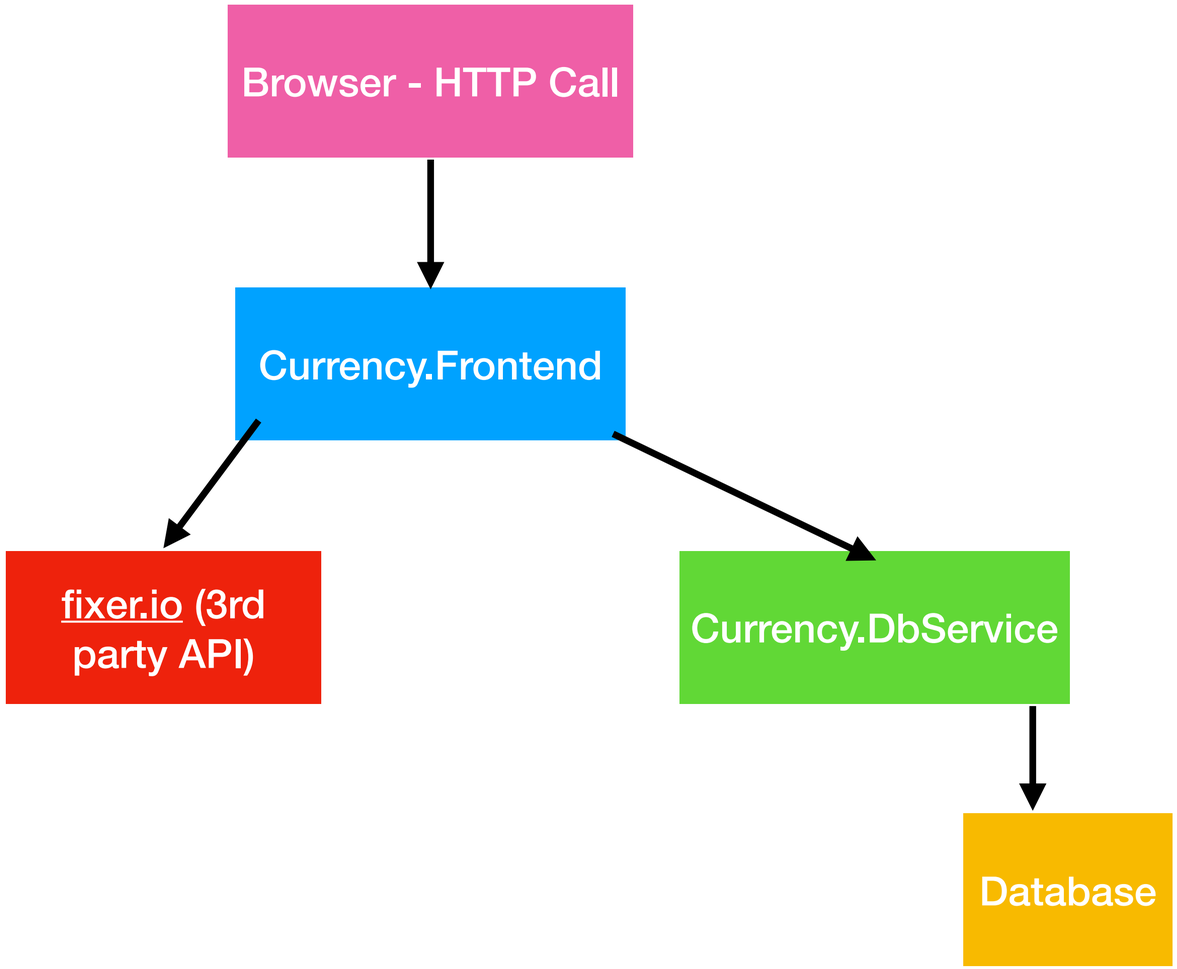
Distributed tracing
Monolith



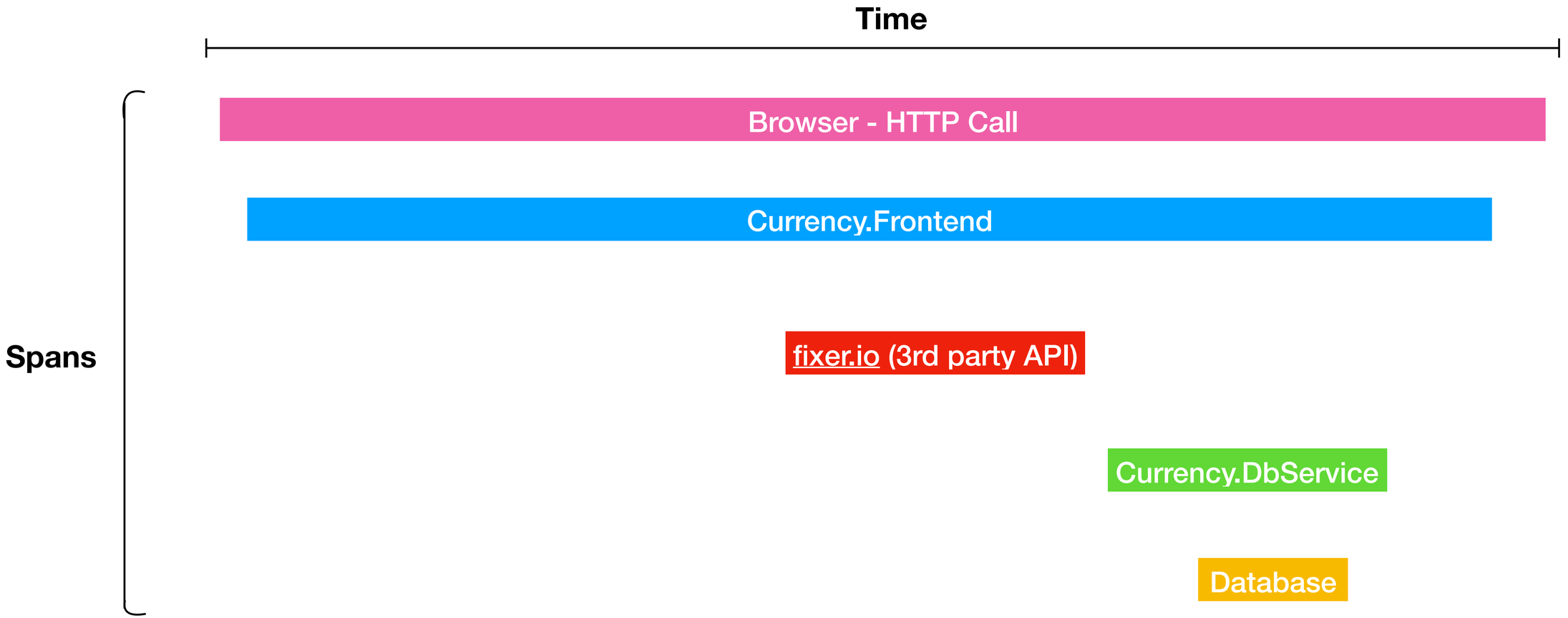
Distributed tracing
Micro-services



Distributed tracing

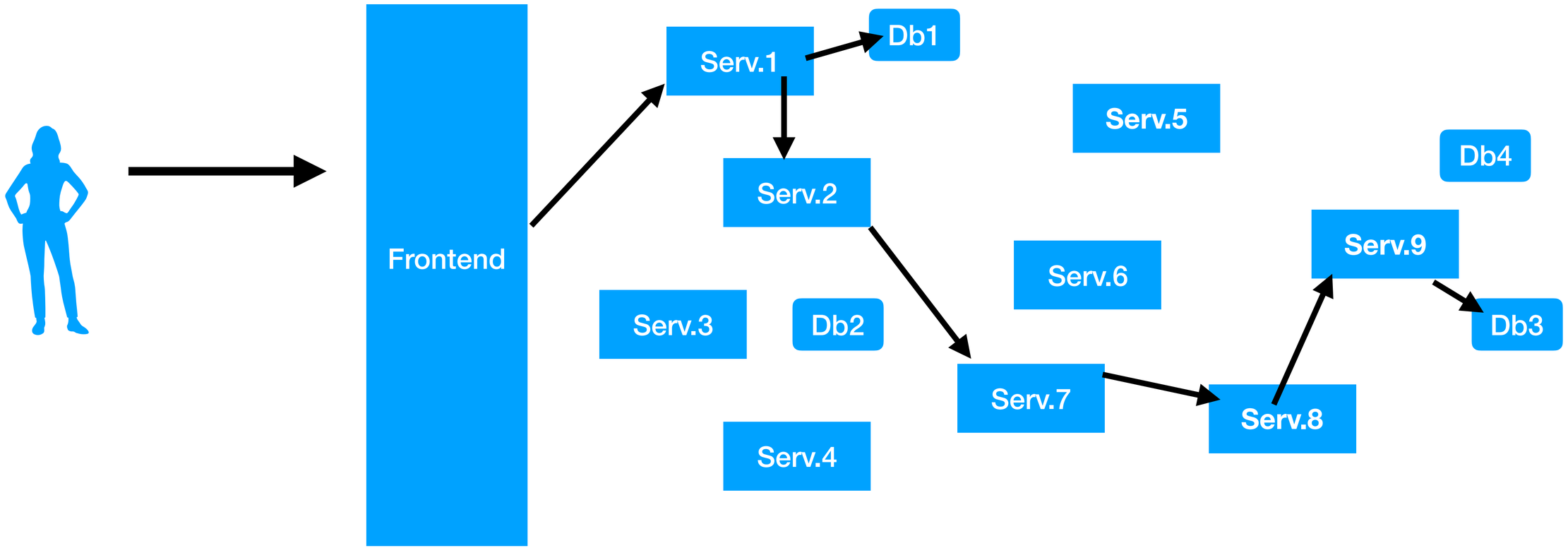


Distributed tracing

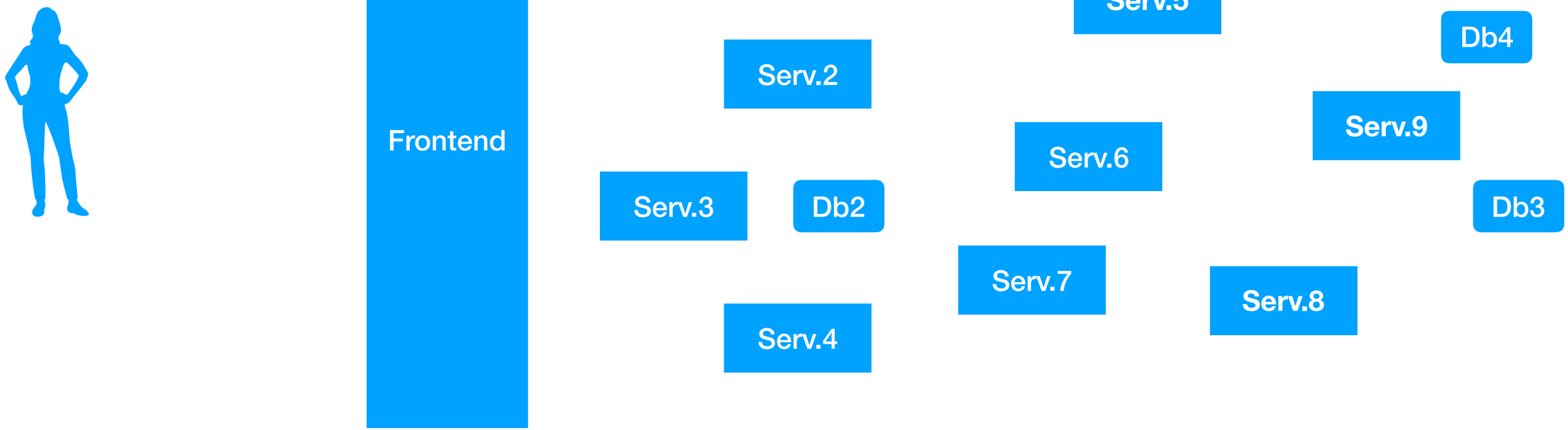


Spans in 1 trace have the same trace ID

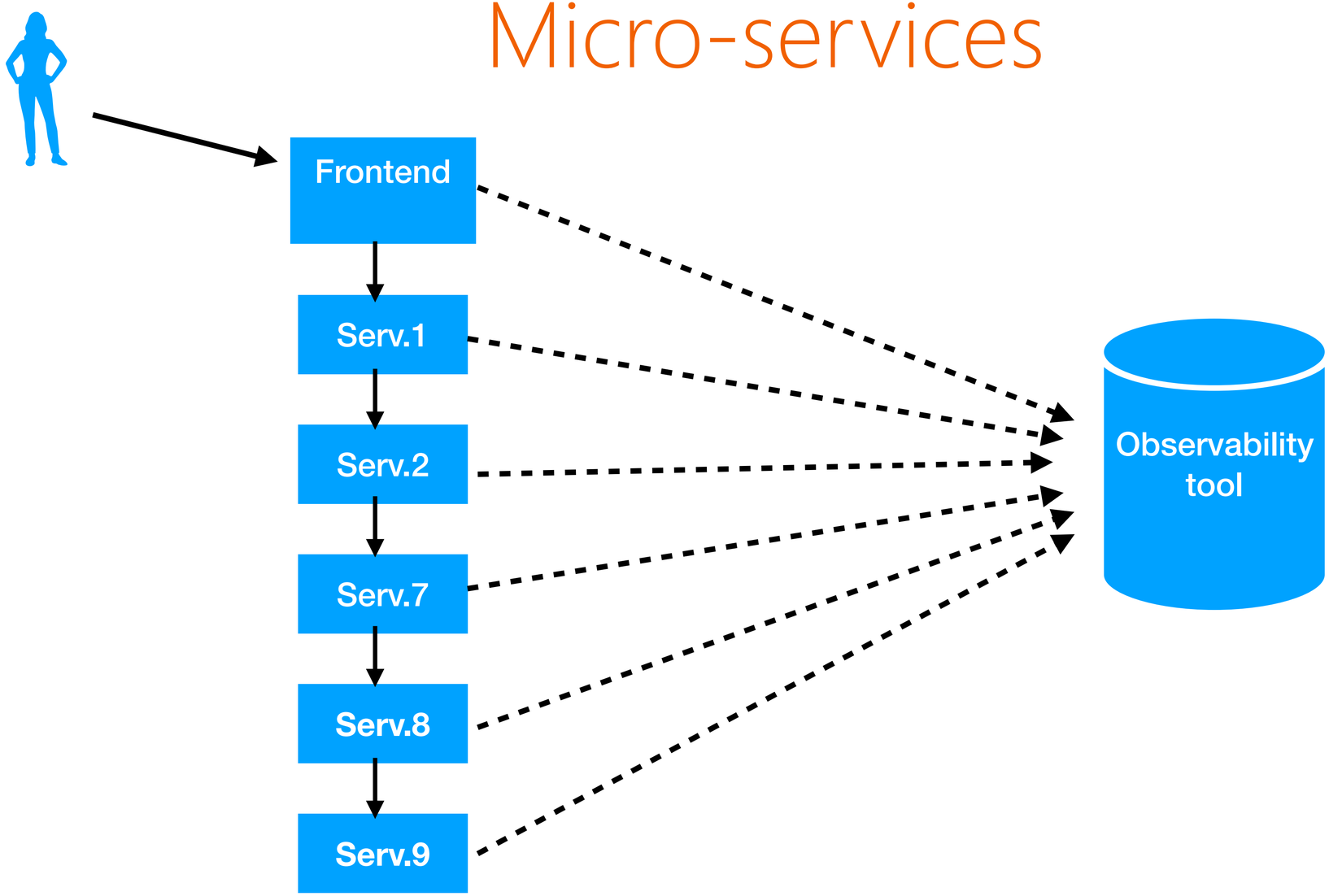
Distributed tracing
Micro-services



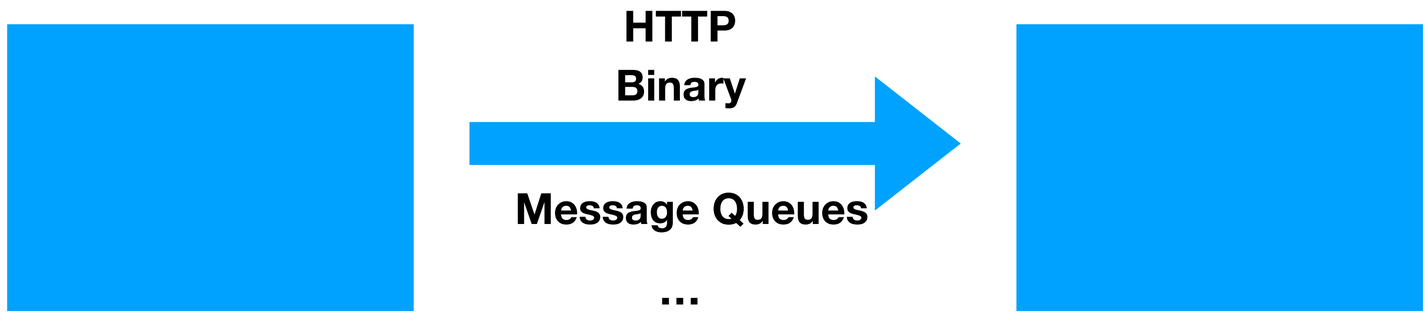
Distributed tracing
Micro-services



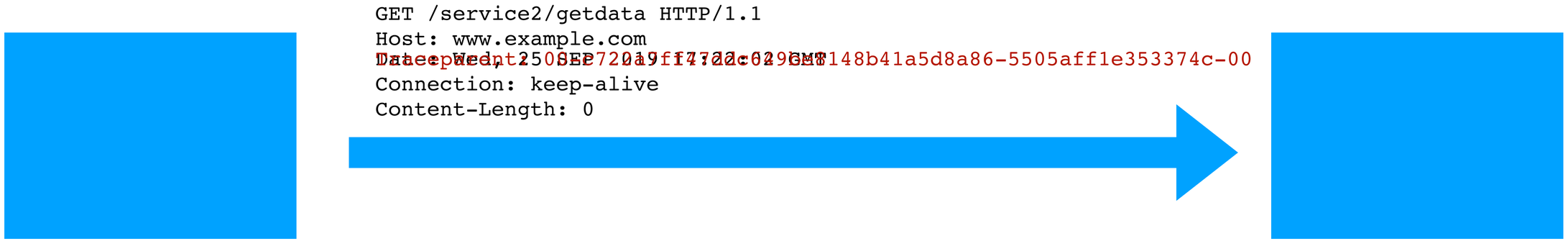
Distributed tracing
Micro-services



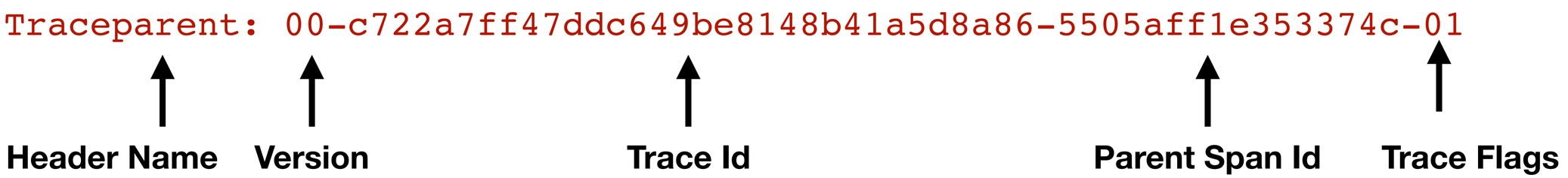
Communication between services



HTTP - W3C TraceContext



HTTP - W3C TraceContext



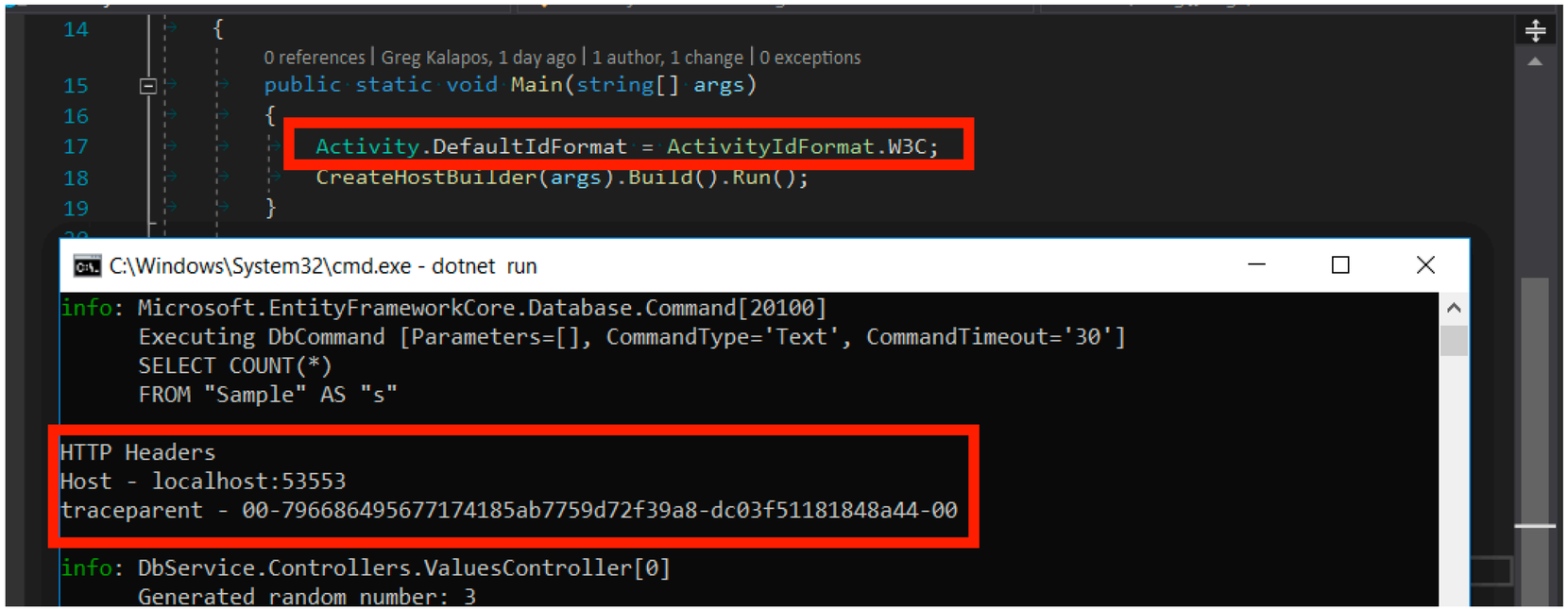
- Made by W3C Trace Context Working Group (https://www.w3.org/TR/trace-context/)
- Status: Candidate Recommendation
- Another header: tracestate - enables tracing by using multiple tracing tools from multiple vendors

HTTP - W3C TraceContext

- System.Diagnostics.Activity: By default it uses its own format (not W3C trace context) (Uses a header called Request-Id)

- Since .NET Core 3.0 you can turn W3C format on:

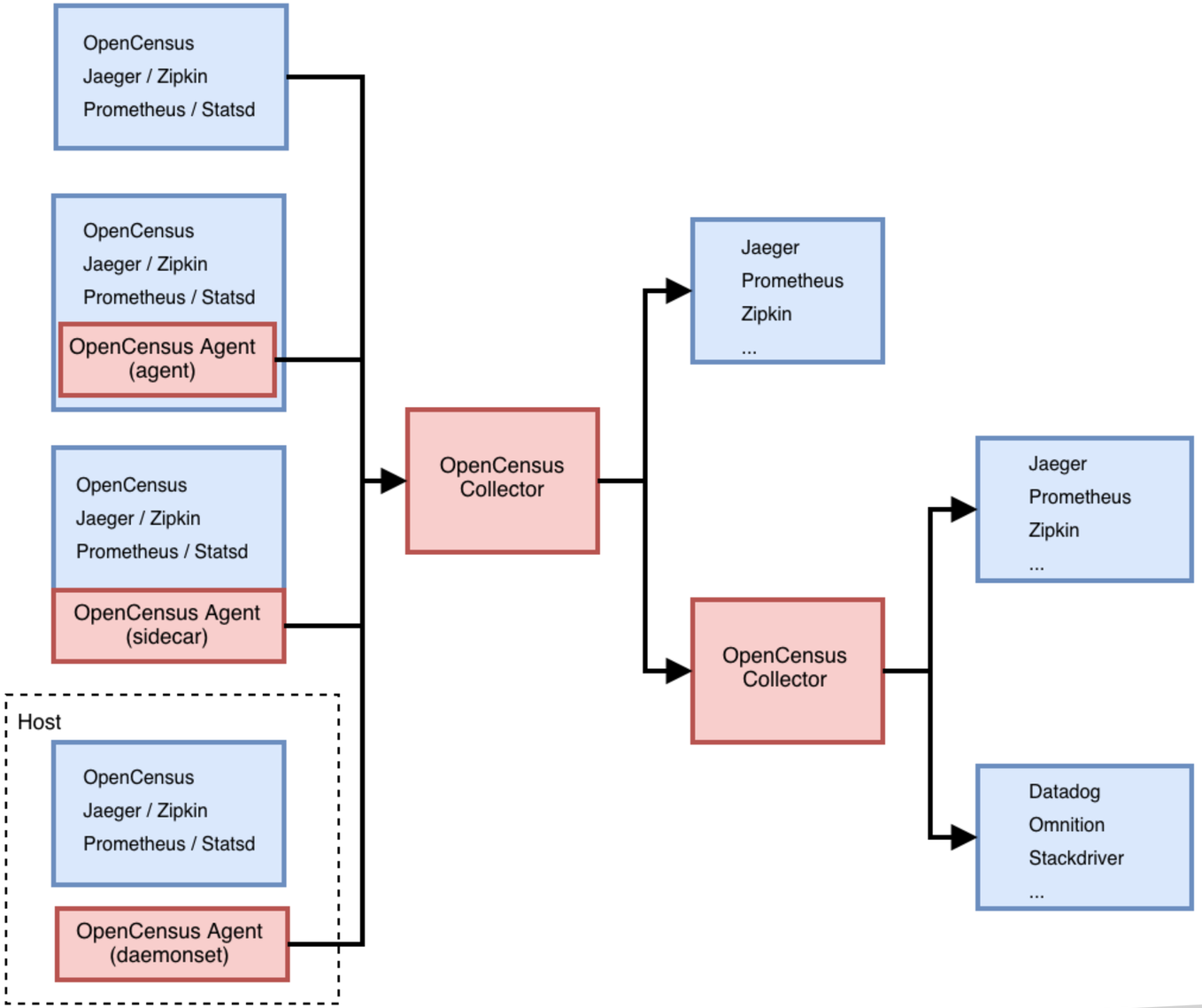
```
Activity.DefaultIdFormat = ActivityIdFormat.W3C;
```



OpenTelemetry

- Merger of OpenTracing and OpenCensus
- CNCF project
- Very early stage
- Unified set of libraries and specifications for observability telemetry
- Avoids vendor lock-in
- Latest OpenTracing .NET version - 0.12.0
- Latest OpenCensus .NET version - 0.1.0-alpha-42253

OpenTelemetry Architecture



OpenTelemetry .NET

- https://github.com/open-telemetry/opentelemetry-dotnet

```
using (tracer.WithSpan(tracer.SpanBuilder("DoWork").StartSpan()))
{
   var span = tracer.CurrentSpan;
   try
   {
       // Simulate some work.
      Console.WriteLine("Doing busy work");
      Thread.Sleep(1000);
   }
   catch (ArgumentOutOfRangeException e)
   {
      // Set status upon error
      span.Status = Status.Internal.WithDescription(e.ToString());
   }
   // Annotate our span to capture metadata about our operation
   var attributes = new Dictionary<string, object>();
   attributes.Add("use", "demo");
   span.AddEvent("Invoking DoWork", attributes);
}
```

Demo

Summary

- Observability tools (Application Insights, Elastic APM, etc): telemetry data from production, correlated logs+metrics+traces
- Manual instrumentation in .NET: Activity and DiagnosticSource
- TelemetryClient in Application Insights, Public API in Elastic APM: bridge your activities
- Distributed tracing, W3C Trace Context: standards for tracing across services
- OpenTelemetry: OpenSource library to avoid vendor lock in - not production ready yet

Demo

Credit/Resources

- Twitter: @gregkalapos
- Web: www.kalapos.net
- Sample code and slides: https://bit.ly/2ksf65E
- Distributed tracing slides/idea - Thomas Watson — An introduction to distributed tracing - https://www.youtube.com/watch?v=g7XSEdriSFM
- DiagnosticSourcery 101 - Mark Rendle: https://www.youtube.com/watch?v=g3SZxsskZE
- Elastic Apm .NET Docs: https://www.elastic.co/guide/en/apm/agent/dotnet/current/index.html
- https://docs.microsoft.com/en-us/azure/azure-monitor/app/api-custom-events-metrics
- TraceContext: https://www.w3.org/TR/trace-context/
- https://opentelemetry.io