white duck

Dapr und Grafana Tempo

Cloud Native Rosenheim Meetup, April 2022

CLOUD NATIVE ROSENHEIM MEET UP

Microsoft Partner

Microsoft

Gold Cloud Platform
Gold DevOps
Silver Application Development
Silver Security
Silver Application Integration

GitHub

Who we are?

Nico Meisenzahl (Cloud Solution Architect, MVP, GitLab Hero)

Email:	nico.meisenzahl@whiteduck.de
Twitter:	@nmeisenzahl
LinkedIn:	https://www.linkedin.com/in/nicomeisenzahl

Dario Brozovic (DevOps Engineer)

Email:	dario.brozovic@whiteduck.de
LinkedIn:	https://www.linkedin.com/in/dariobrozovic

Markus Voit (Software & DevOps Engineer)

Email:	markus.voit@whiteduck.de
Twitter:	@mvoitdev

CLOUD NATIVE ROSENHEIM MEET UP

twitter.com/CloudNative_Ro

twitter.com/whiteduck_gmbh

AZURE ROSENHEIM MEETUP

twitter.com/AzureMeetup

Housekeeping

- this meetup will be streamed on YouTube!
- want to participate?
  - join our meetup to get access to the Zoom meeting
    - https://www.meetup.com/CloudNative-Rosenheim-Meetup
  - we do also monitor the comments on YouTube

Agenda

- Polyglotte Microservice-Entwicklung mit Dapr
- Distributed tracing mit Grafana Tempo
- Q&A

Polyglotte Microservice-Entwicklung mit Dapr

Motivation

„Microservices can be written in different languages, use different libraries, and use different data stores.“

Martin Fowler

What are polyglot microservices

- Are a flavour of microservice architecture
- Use multiple technologies and programming languages
- Are distributed systems

GO
node js
python™
Application code
Java
.NET
Any code or framework...

Pros of this approach

- Each microservice can use the “best” tech stack
- Easier adoption of new and innovative technology
- Developers can use the programming language they love the most
- Developers can contribute to more projects

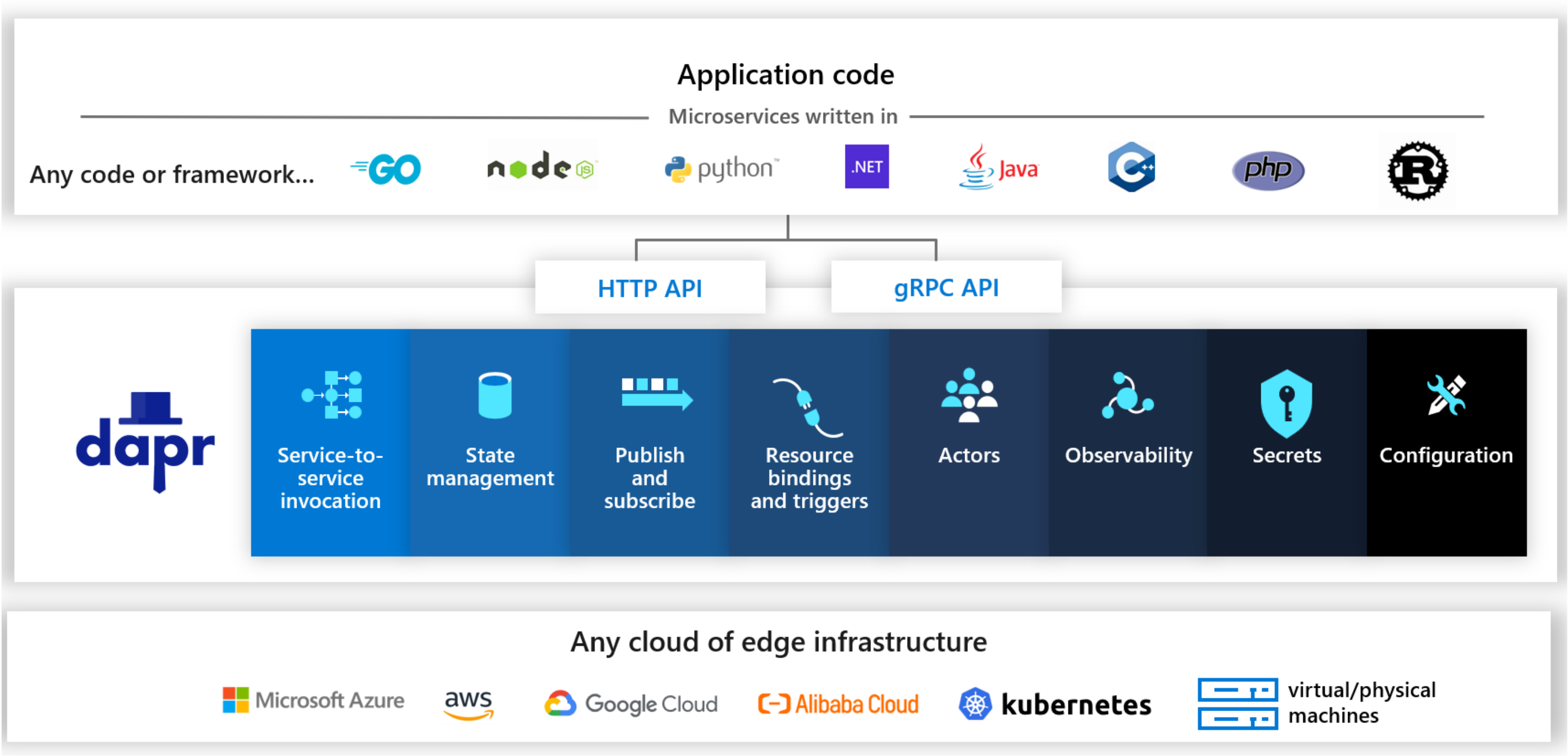
Cons of this approach

- Differences in programming languages
- Differences in tooling (libraries, SDKs, frameworks)
- Sharing of implementations between different programming languages is difficult
- Time for implementation and complexity are increasing with each additional programming language

So polyglot microservices are bad

- Complexity is an issue
- We are missing “polyglot libraries and frameworks”
- But what if there were tooling that could help us with that

What is Dapr



Application code

Microservices written in

Any code or framework...

GO · node · python · .NET · Java · C++ · php · R

HTTP API

gRPC API

dapr

Service-to-service invocation

State management

Publish and subscribe

Resource bindings and triggers

Actors

Observability

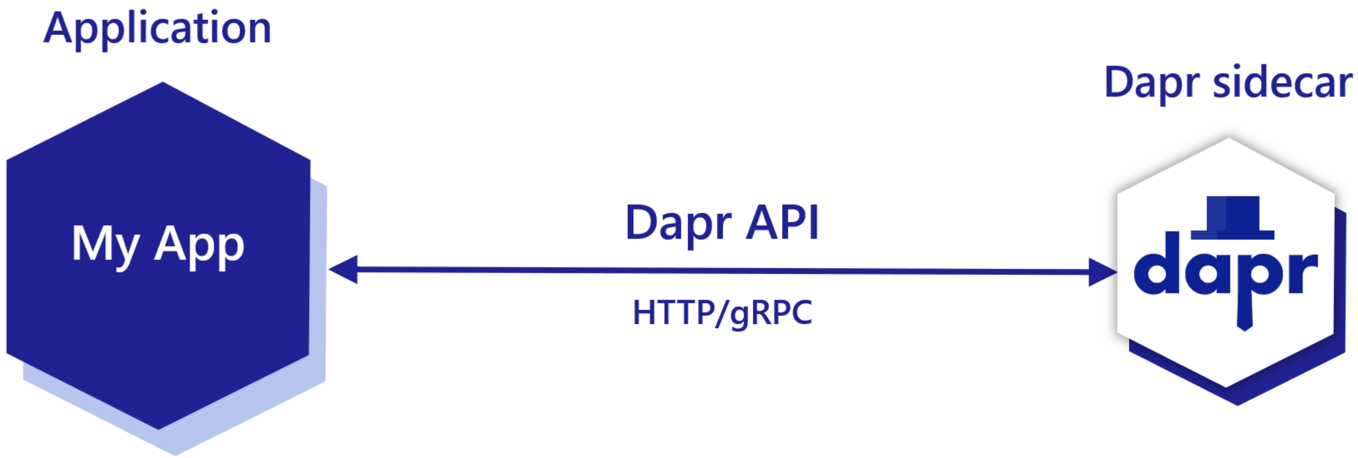
Secrets

Configuration

Any cloud of edge infrastructure

Microsoft Azure · aws · Google Cloud · Alibaba Cloud · kubernetes · virtual/physical machines

How does it work



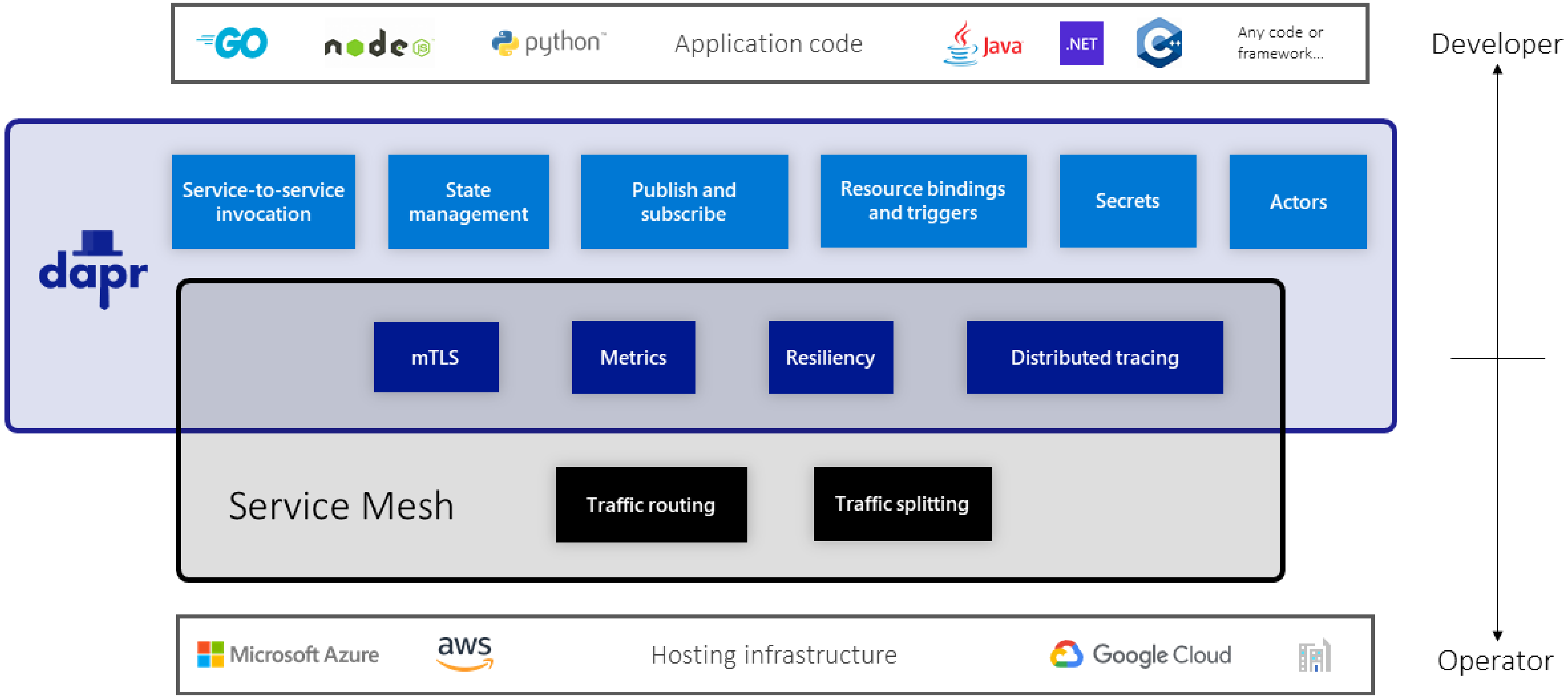
```
POST http://localhost:3500/v1.0/invoke/cart/method/neworder

GET http://localhost:3500/v1.0/state/inventory/item67

POST http://localhost:3500/v1.0/publish/shipping/orders

GET http://localhost:3500/v1.0/secrets/keyvault/password
```

Why do we need it

- Abstract third-party dependencies
- Abstract implementation of non-functional requirements
- Reduce code that must be repeated in every programming language

- Reduce the complexity of polyglot microservice

Dapr vs Service Mesh



Limitations

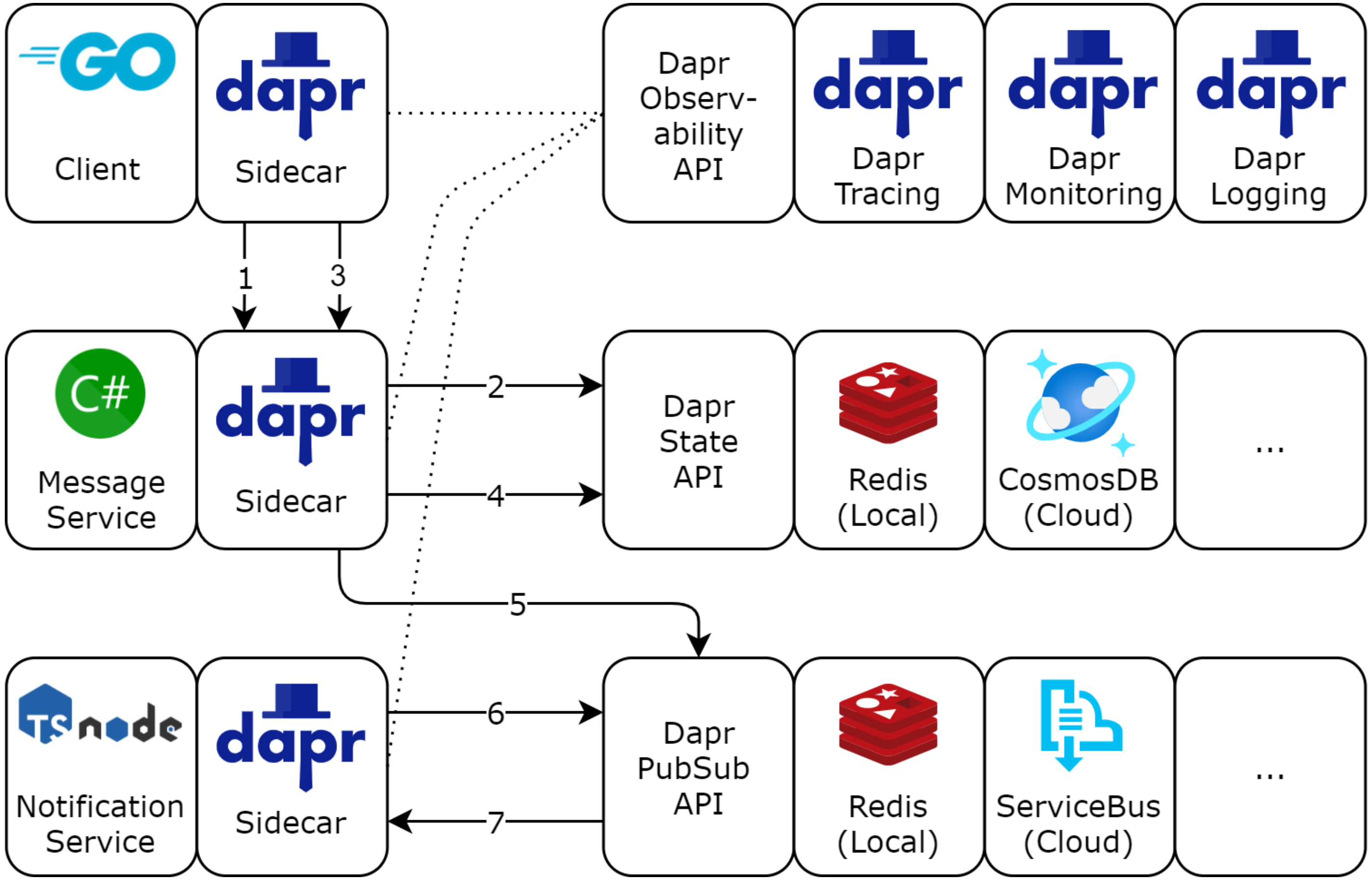
- Is an abstraction layer
- State management is only suitable for simple use cases
- For now Observability is mostly limited to Dapr itself

When to use it

- It depends ☺

- When you:
  - are developing polyglot microservices
  - could profit from Dapr capabilities
  - can justify the effort to adopt the Dapr programming model
  - don’t have critical performance requirements
  - need the flexibility Dapr offers (more on the demo)

Demo


Client
Sidecar
Dapr Observ-ability API
Dapr Tracing
Dapr Monitoring
Dapr Logging
1
3
Message Service
Sidecar
2
4
Dapr State API
Redis (Local)
CosmosDB (Cloud)
...
5
Notification Service
Sidecar
6
7
Dapr PubSub API
Redis (Local)
ServiceBus (Cloud)
...

Demo

- Three microservices (Go, C#, TypeScript)
- Service invocation, State management, PubSub, and Observability
- Switch between local and cloud deployment using different services

Further reading

- See my three parts Dapr blog post series:
- 1/3 Polyglot microservice development with Dapr
- 2/3 Polyglot microservice development with Dapr
- 3/3 Polyglot microservice development with Dapr

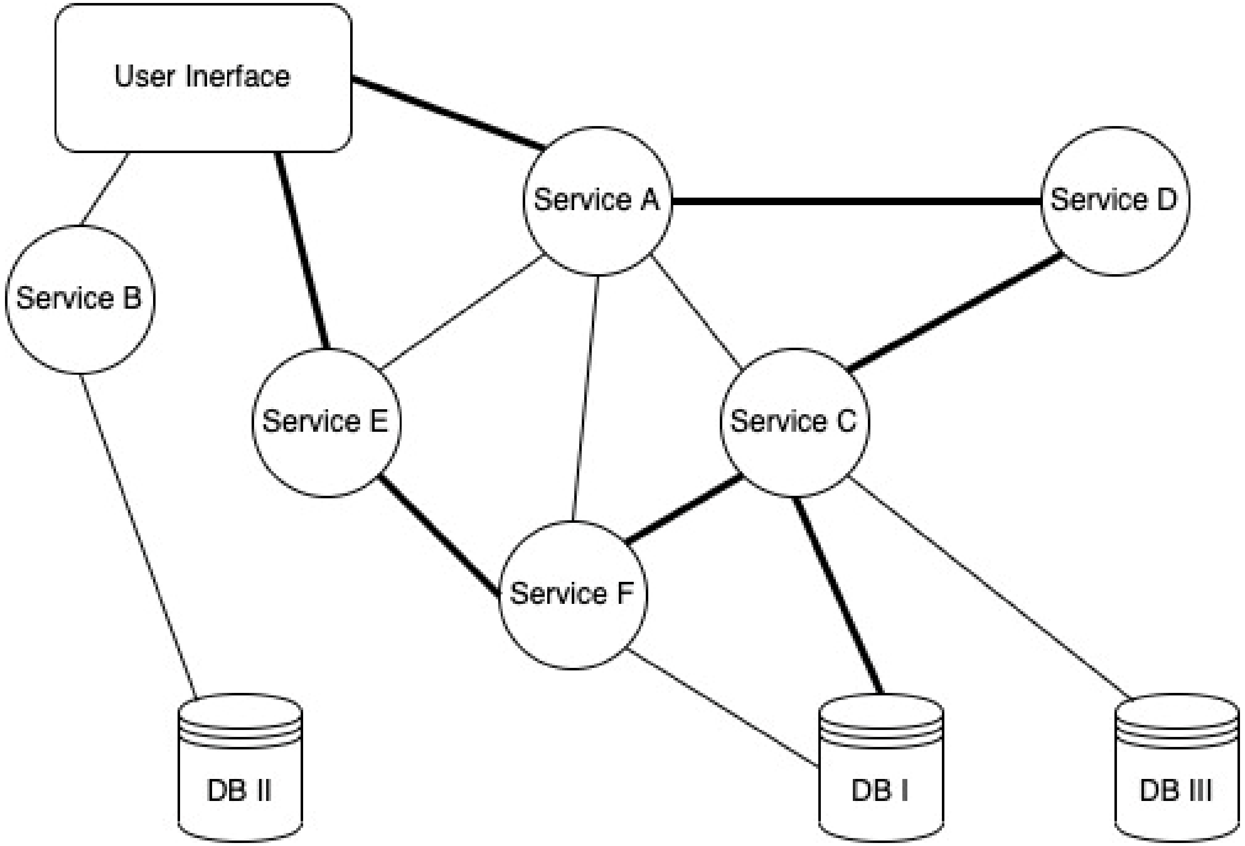
Distributed tracing with Grafana Tempo

What is distributed tracing

- Method used to profile and monitor applications, especially those built using a microservice architecture

- Used to help pinpoint where failures occur and what causes poor performance

Why do we need tracing



Why do we need tracing

- Measure overall system health
- Identify and resolve issues to minimize the impact on business outcomes
- Prioritize areas for improvement to optimize service quality

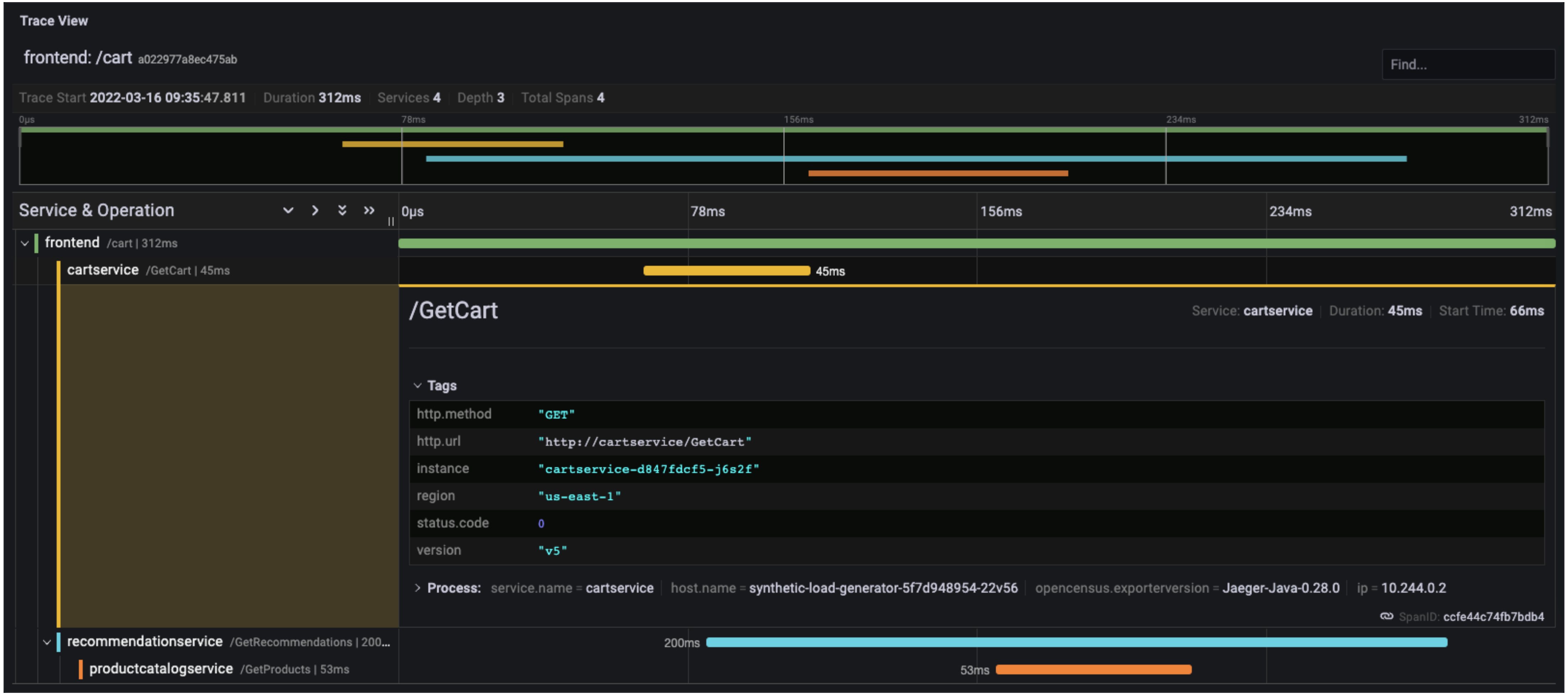
Terminology: Trace

- Performance data about requests as they flow through microservices
- Collection of operations that represent a unique transaction handled by an application and its constituent services

Terminology: Span

- Single operations or segments that are part of a trace
  - Root span is the first span in a trace
  - Child span is a subsequent span
- Represents an individual unit of work in a distributed system

Example



Trace View

frontend: /cart a022977a8ec475ab

Trace Start **2022-03-16 09:35:47.811** | Duration **312ms** | Services **4** | Depth **3** | Total Spans **4**

Service & Operation

- frontend /cart | 312ms
  - cartservice /GetCart | 45ms
  - recommendationservice /GetRecommendations | 200...
    - productcatalogservice /GetProducts | 53ms

/GetCart

Service: **cartservice** | Duration: **45ms** | Start Time: **66ms**

Tags

http.method	"GET"
http.url	"http://cartservice/GetCart"
instance	"cartservice-d847fdcf5-j6s2f"
region	"us-east-1"
status.code	0
version	"v5"

Process: service.name = **cartservice** | host.name = **synthetic-load-generator-5f7d948954-22v56** | opencensus.exporterversion = **Jaeger-Java-0.28.0** | ip = **10.244.0.2**

SpanID: **ccfe44c74fb7bdb4**

Grafana Tempo

- Highly scalable, distributed, Open-source tracing backend
- Compatible with many tracing protocols
    - Zipkin
    - Jaeger
    - OpenTelemetry

Grafana Tempo

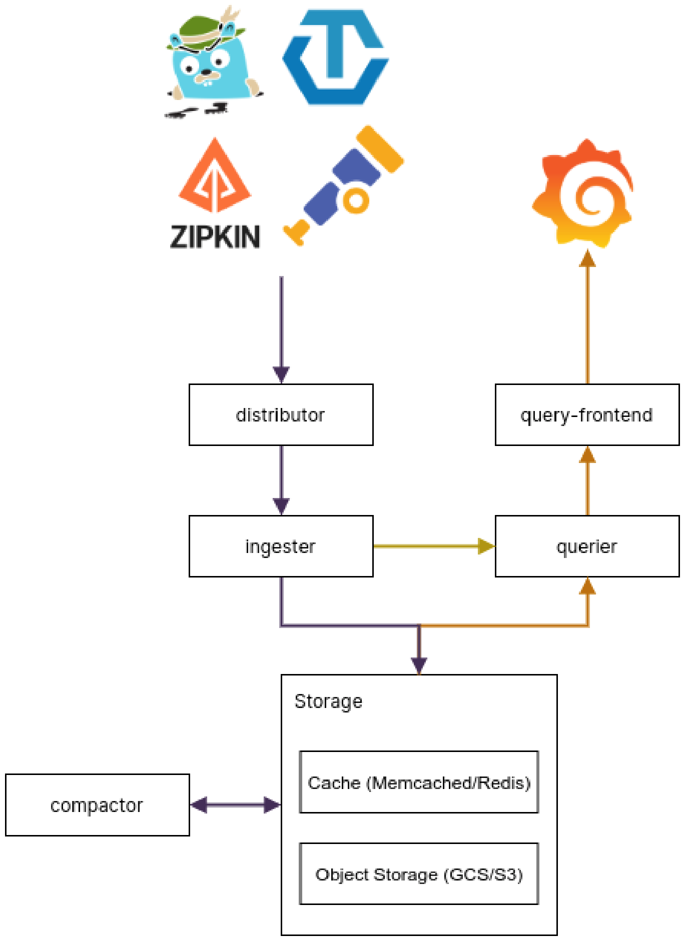
- Advantages
  - Uses managed object storage for persistence
  - Lightweight, no need for Cassandra or Elastic
  - Great integration with Logs and Metrics in Grafana

Instrumentation

- Happens programmatically depending on protocol and programming language
  - Example
    - Protocol
      - OpenTelemetry
    - Language
      - Go

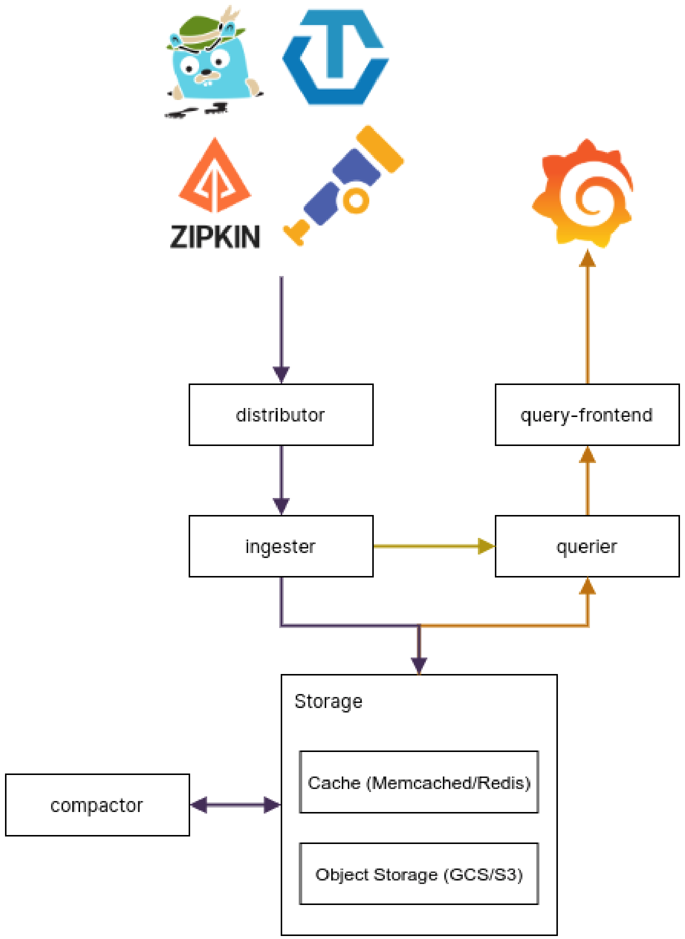
Architecture

- Distributor
  - The distributor accepts spans in multiple formats including Jaeger, OpenTelemetry, Zipkin
  - Routes spans to Ingesters



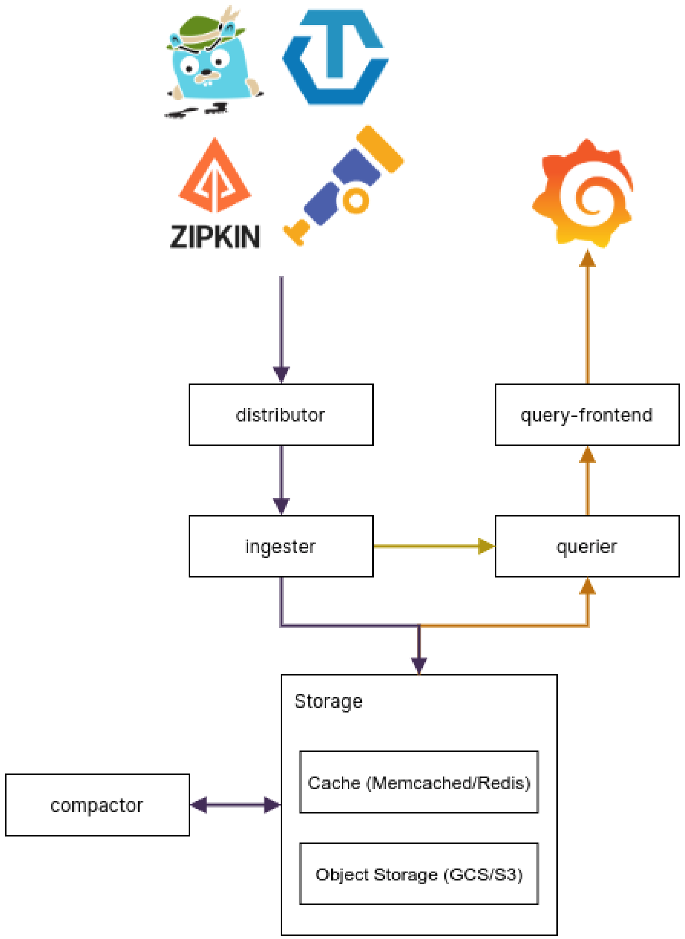
Architecture

- Ingester
  - Create batches of traces called blocks
  - Flush the blocks to the backend storage



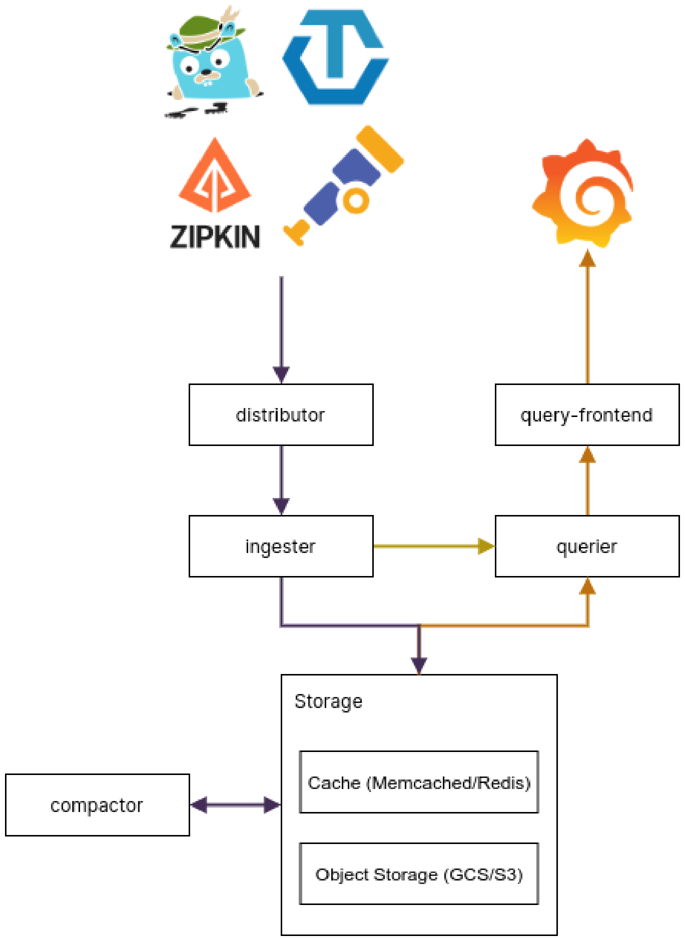
Architecture

- Storage
  - Local
  - GCS
  - S3
  - Azure
  - ..



Architecture

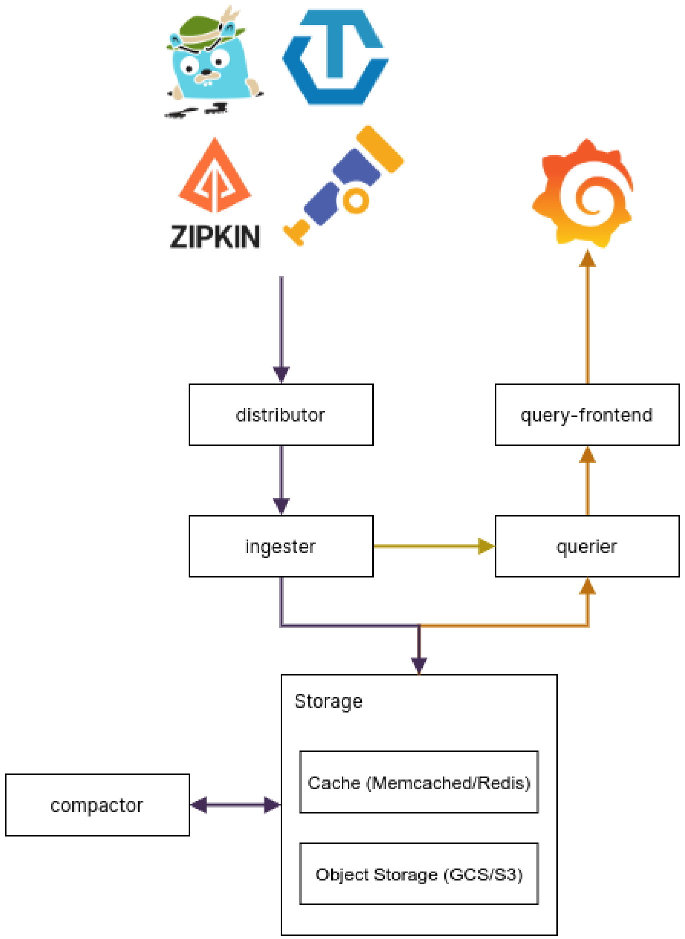
- Compactor
  - stream blocks from the backend storage, combines them, and sends them back to reduce the number of blocks in the storage



Architecture

- Querier
  - Responsible for finding the requested trace id in either the ingesters or backend storage

- Query-frontend
  - Optimizes the query



Demo

- Environment
  - Load generator (simulates traces)
  - Tempo (microservice deployment)
  - Grafana
  - Azure Storage account

Slides & Demos

- https://github.com/whiteducksoftware/cloud-native-rosenheim-meetup