



Day 1: Observability in Microservice Architectures

Lesley Cordero

Staff Engineer, Observability
The New York Times





Agenda

Day 1:

- Observability & role in microservice architectures
- Standardization & OpenTelemetry
- Telemetry signals & best practices

Day 2:

- Sociotechnical challenges of observability
- Architecture challenges of observability
- Observability platform architecture

Themes:

- Standardization & platformization
- Other best practices
- Sociotechnical approaches



Poll

“Which role best describes your current role?”

1. *Software Engineer*
2. *Site Reliability Engineer*
3. *IT Professional*
4. *DevOps Engineer*
5. *Engineering Manager*
6. *Other*



Poll

“Does your organization utilize the OpenTelemetry Standard?”

1. Yes
2. No
3. *Unclear*
4. *It's on the roadmap!*

Hi! I'm **Lesley Cordero**.

Fan of boba, dogs, and strong developer platforms.

My experience debugging production issues when **schools went remote** has me thinking about how we can build robust observability platforms.



Hi! I'm **Lesley Cordero**.

Fan of boba, dogs, and strong developer platforms.

My experience debugging production issues when **schools went remote** has me thinking about how we can build robust observability platforms.



Defining Observability

“the ability to understand what’s happening inside of your software systems to debug problems you’ve never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It’s also not just about a specific tool, it’s about a team’s ability to analyze that telemetry data.”

- Liz Fong-Jones, Honeycomb

Defining Observability

“the ability to understand what’s happening inside of your software systems to debug problems you’ve never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It’s also not just about a specific tool, it’s about a team’s ability to analyze that telemetry data.”

- Liz Fong-Jones, Honeycomb

Defining Observability

“the ability to understand what’s happening inside of your software systems to debug problems you’ve never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It’s also not just about a specific tool, it’s about a team’s ability to analyze that telemetry data.”

- Liz Fong-Jones, Honeycomb

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Adapting for complex applications



Microservices

Addresses complexity related to *developing* large applications.



Observability

Addresses complexity related to *debugging* large applications.

Emerging organizational challenges

1. Silos & Drift



Microservices

Engineers *build* services in many different ways



Observability

Engineers *observe* services in many different ways

Emerging organizational challenges

1. Silos & Drift



Microservices

Engineers *build* services in many different ways



Observability

Engineers *observe* services in many different ways

Emerging organizational challenges

1. Silos & Drift



Microservices

Engineers *build* services in many different ways



Observability

Engineers *observe* services in many different ways

Emerging organizational challenges

1. Silos & Drift
2. **Multiple points of failure**



Microservices

One service's failures can cascade to another service.



Observability

One service's telemetry gap can lead to a telemetry gap in another service.

Emerging organizational challenges

1. Silos & Drift
2. **Multiple points of failure**



Microservices

One service's failures can cascade to another service.



Observability

One service's telemetry gap can lead to a telemetry gap in another service.

Emerging organizational challenges

1. Silos & Drift
2. Multiple points of failure
3. **Inherent lack of certainty**



Microservices

Teams can't know if other services are working as expected.



Observability

Teams can't know that other services are emitting complete telemetry data.

Emerging organizational challenges

1. Silos & Drift
2. Multiple points of failure
3. **Inherent lack of certainty**



Microservices

Teams can't know if other services are working as expected.



Observability

Teams can't know that other services are emitting complete telemetry data.

Emerging organizational challenges

1. Silos & Drift
2. Multiple points of failure
3. **Inherent lack of certainty**



Microservices

Teams can't know if other services are working as expected.



Observability

Teams can't know that other services are emitting complete telemetry data.

Introducing

Standardized Observability Platforms



Q&A



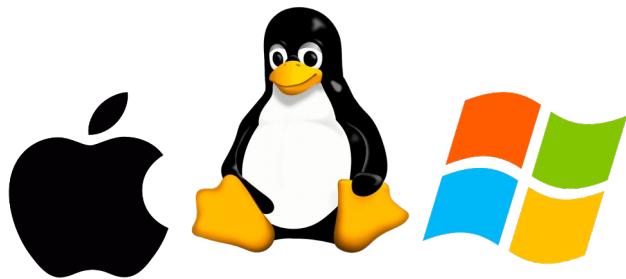
Standardization

By driving standardization (the patterns we follow, through the tools we use), we'll be better supported to understand and analyze our telemetry data.

Benefits of Standardization

1. Shared language & understanding

Standardization drives clarity by providing a shared language as a foundation for how we communicate.



Benefits of Standardization

1. Shared language & understanding

Standardization drives clarity by providing a shared language as a foundation for how we communicate.

2. Efficient decision making

This foundation enables us to make decisions more quickly and thoroughly.



Lightstep

from ServiceNow



splunk®



Stackdriver



dynatrace



APPDYNAMICS



AWS CloudWatch



INSTANA



DATADOG

 **Cribl**®

Benefits of Standardization

1. Shared language & understanding

Standardization drives clarity by providing a shared language as a foundation for how we communicate.

2. Efficient decision making

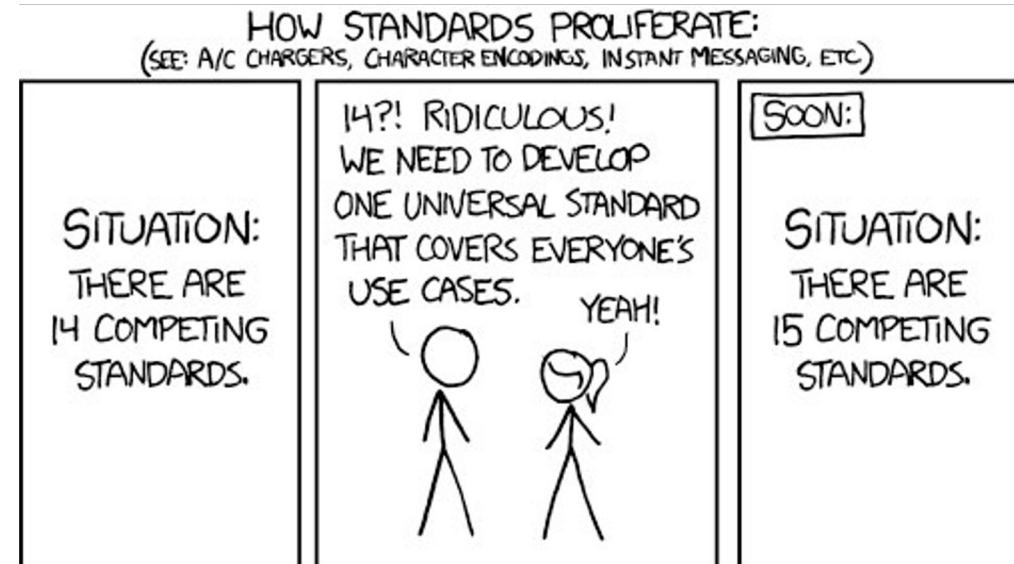
This foundation enables us to make decisions more quickly and thoroughly.

3. Technical maturity

Over time, the quality or maturity of our domain or system increases.

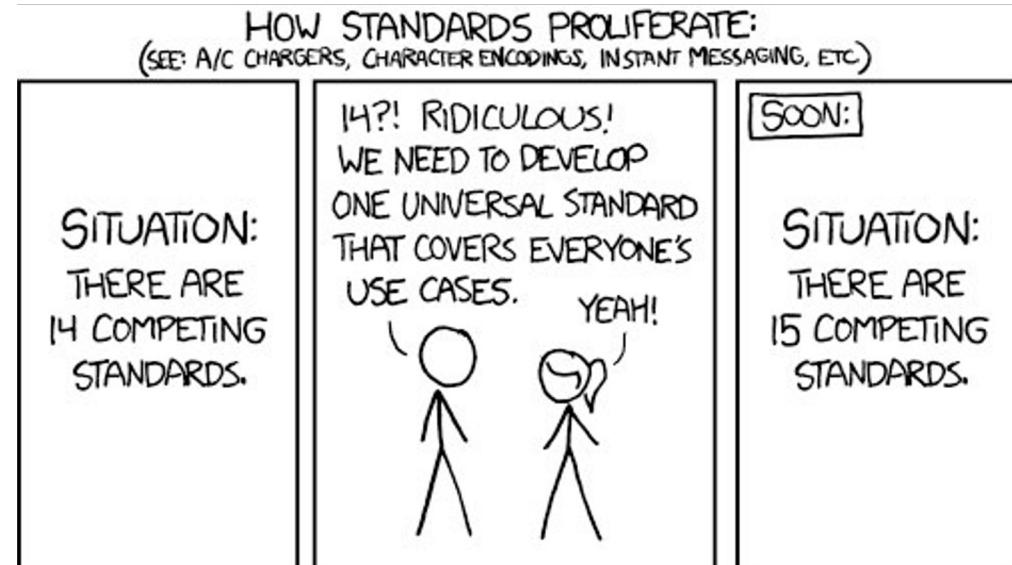


Standards Paradox





Standards Paradox





Introducing OpenTelemetry

“To provide a set of standardized vendor-agnostic SDKs, APIs, and tools for ingesting, transforming, and sending data to an observability backend”



Introducing OpenTelemetry

“To provide a set of standardized vendor-agnostic SDKs, APIs, and tools for ingesting, transforming, and sending data to an observability backend”



Introducing OpenTelemetry

“To provide a set of standardized vendor-agnostic SDKs, APIs, and tools for **ingesting, transforming, and sending data to an observability backend**”



Introducing OpenTelemetry

“To provide a set of standardized vendor-agnostic SDKs, APIs, and tools for ingesting, transforming, and sending data to an observability backend”



Introducing OpenTelemetry

“To provide a set of standardized vendor-agnostic SDKs, APIs, and tools for ingesting, transforming, and sending data to an observability backend”

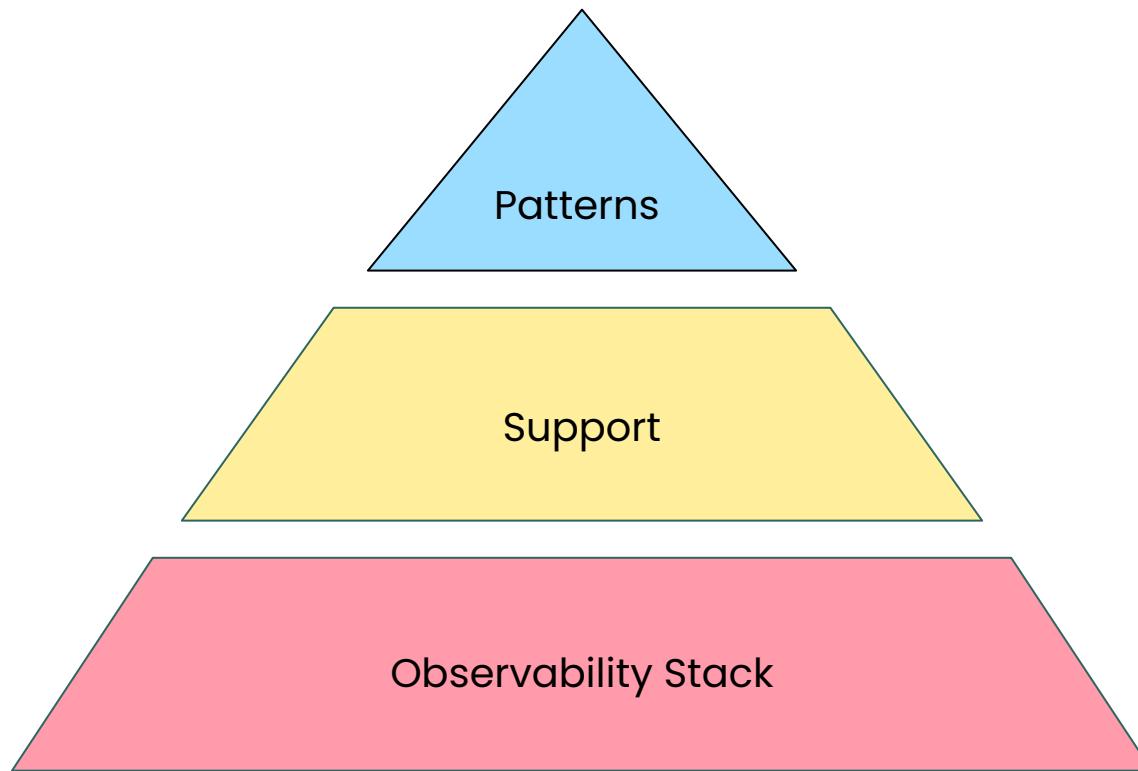


Poll

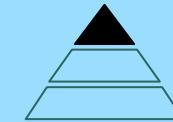
“Which language(s) does your organization use?
Check all that apply.”

- 1. *Front-end Typescript/Javascript*
- 2. *Node.js Typescript/Javascript*
- 3. *Go*
- 4. *Python*
- 5. *Java / JVM*
- 6. *PHP*
- 7. *Erlang / Elixir*
- 8. *C++*
- 9. *Rust*
- 10. *Ruby*
- 11. *Swift*
- 12. *Other*

Standardized observability platforms



Standardized Communication



Patterns
Support
Stack

Standardized Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

Standardized Communication



Patterns
Support
Stack

1. Shared data specifications

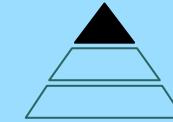
Defines telemetry data definitions that drive consistency in how telemetry is collected.

```
{  
  traceId: 123  
  ...  
}
```

```
{  
  trace_id: 123  
  ...  
}
```

“traceId: 123”

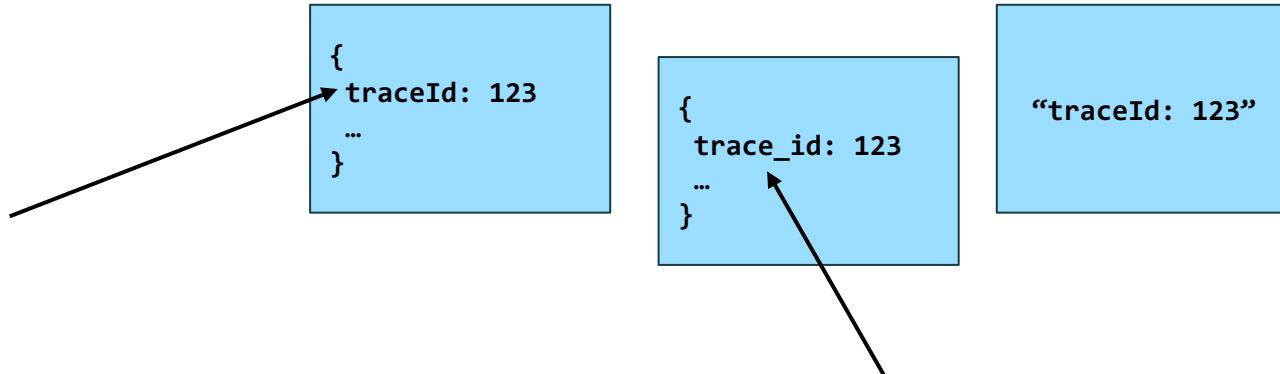
Standardized Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.



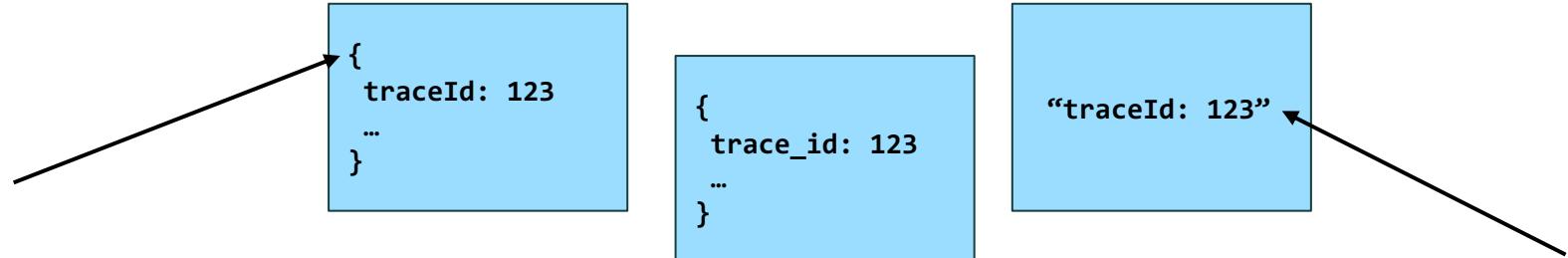
Standardized Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.



Telemetry



Distributed Traces



Metrics

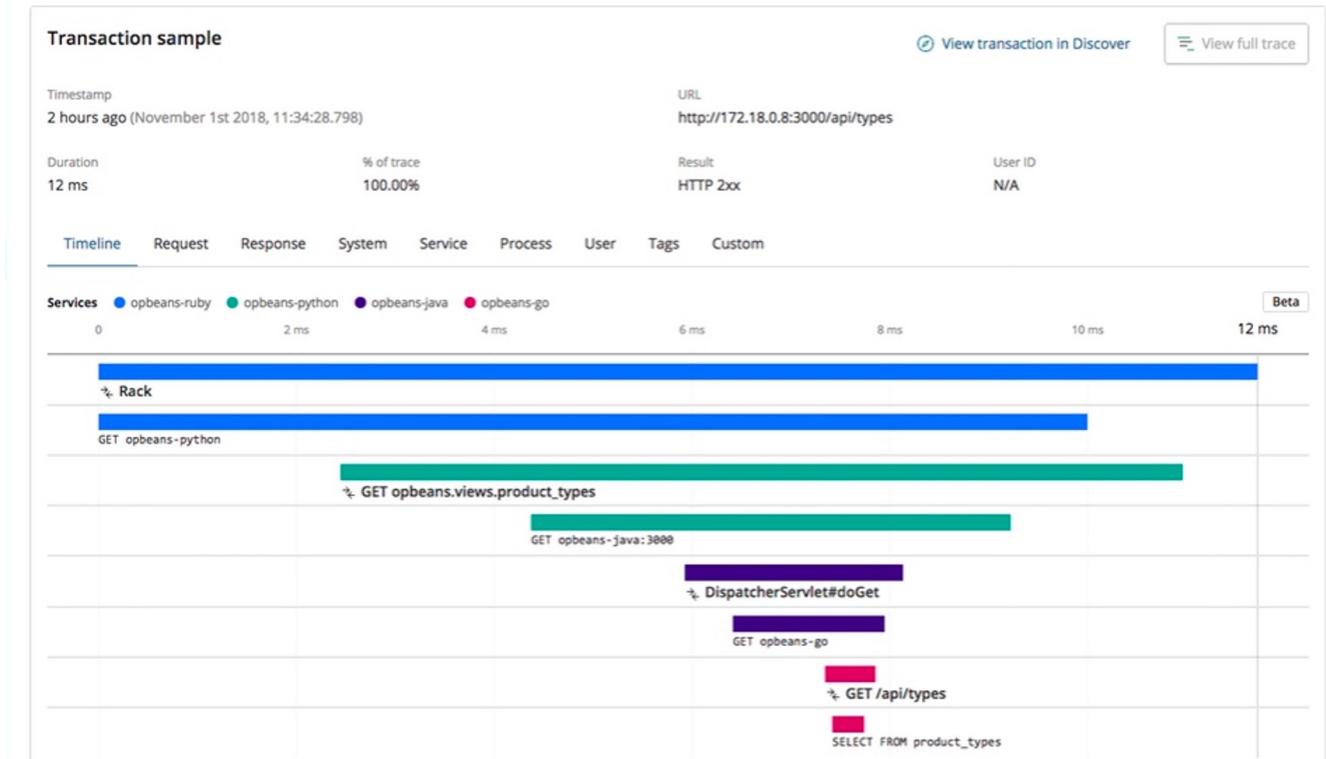


Logs

Tracks & provides the context
& flow of tasks as they move
through different system
components.

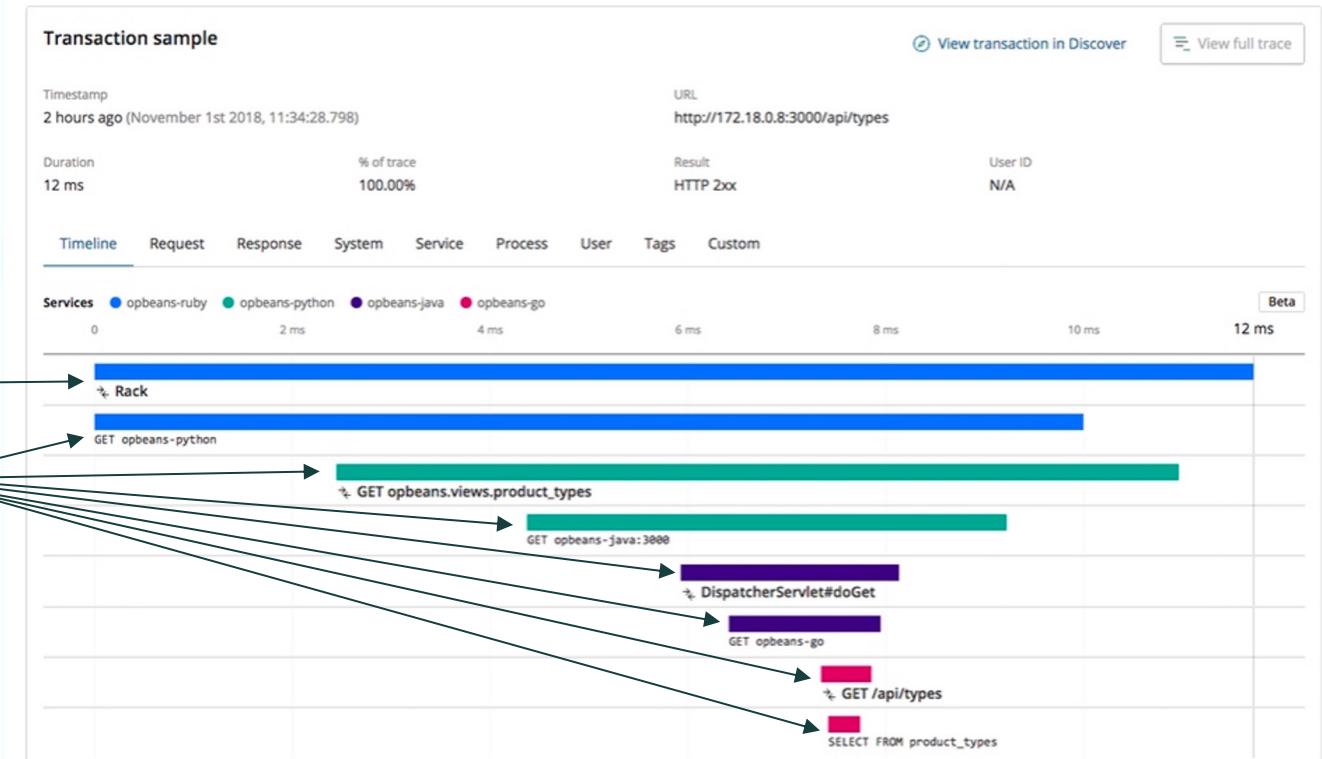


Viewing Traces





Viewing Traces: Spans





Q&A

Tracers & Trace Context

Trace Context

The necessary metadata that enables tracers to manage, correlate, and assemble spans.

Tracers

```
{  
  "trace_id": "1234",  
}
```



Trace Context

The necessary metadata that enables tracers to manage, correlate, and assemble spans.



Tracers



Viewing Traces

```
graphql.execute query AssetsContainerQuery($asset_type_slug:AssetTypeSlug,$cursor:String,$limit:Int,$order:SortOrder,$provider:GUID,$query...
```

```
{  
  "trace_id": "1234",  
}
```



Trace Context

The necessary metadata that enables tracers to manage, correlate, and assemble spans.



Tracers

```
{  
  "trace_id": "1234",  
  "span_id": "9876",  
  "name": "/health",  
  "parent_id": "",  
  "start_time": "UTC timestamp",  
  "end_time": "UTC timestamp",  
  "status_code": "STATUS_CODE",  
}
```

Span Context

Additional metadata that enables tracers to manage, correlate, and assemble spans.

Tracers

```
{  
  "trace_id": "1234",  
  "span_id": "9876",  
  "name": "/health",  
  "parent_id": "",  
  "start_time": "UTC timestamp",  
  "end_time": "UTC timestamp",  
  "status_code": "STATUS_CODE",  
}
```

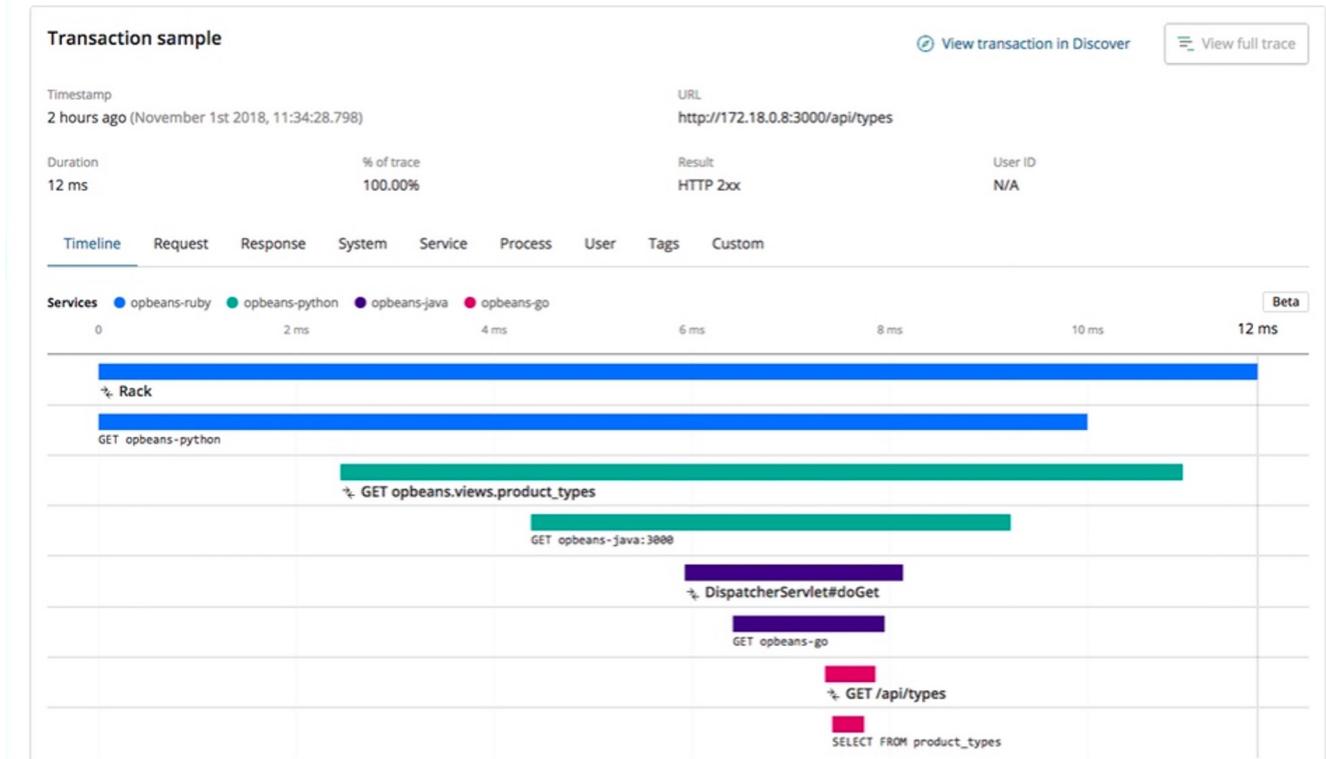
Span Context

Additional metadata that enables tracers to manage, correlate, and assemble spans.

Tracers

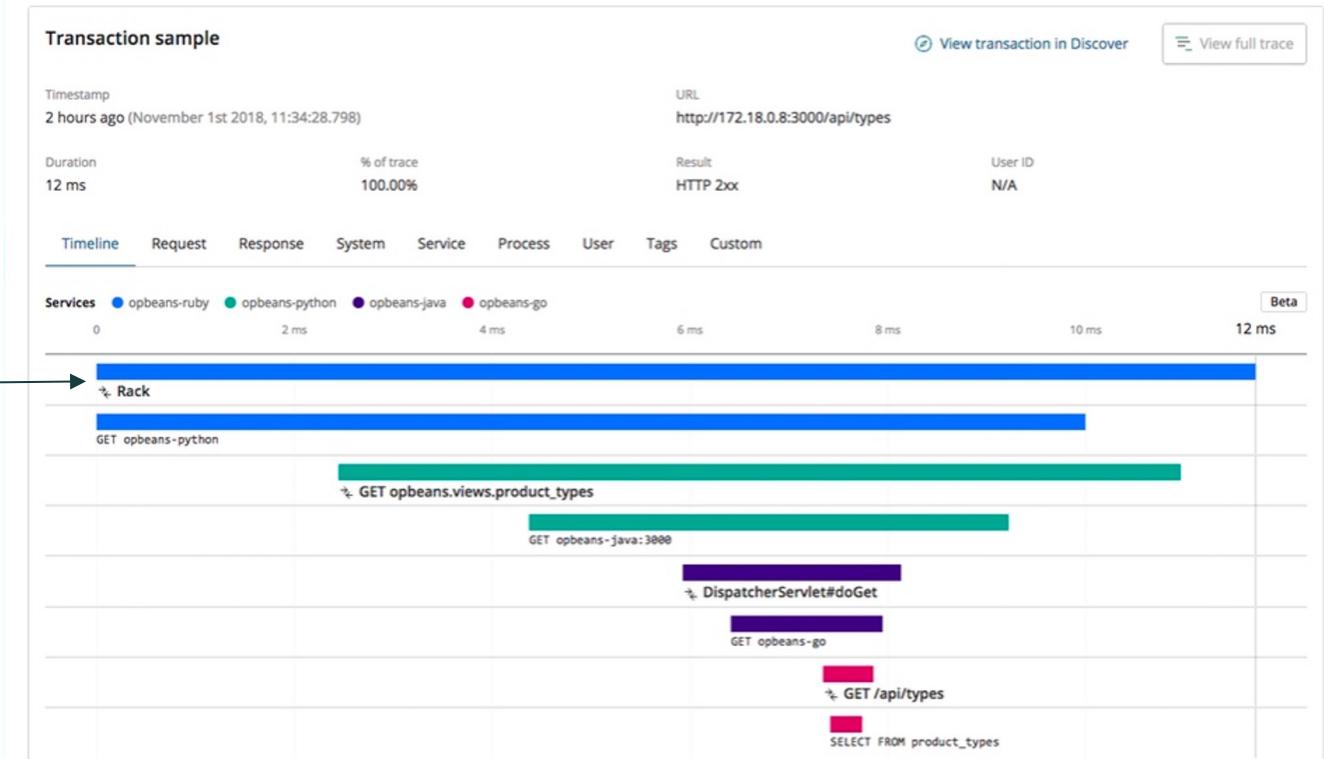


Viewing Traces





Viewing Traces: Spans



```
{  
  "trace_id": "1234",  
  "span_id": "9876",  
  "name": "/health",  
  "parent_id": "",  
  "start_time": "UTC timestamp",  
  "end_time": "UTC timestamp",  
  "status_code": "STATUS_CODE",  
}
```

Span Context

Additional metadata that enables tracers to manage, correlate, and assemble spans.

Tracers

```
{  
  "trace_id": "1234",  
  "span_id": "9876",  
  "name": "/health",  
  "parent_id": "",  
  "start_time": "UTC timestamp",  
  "end_time": "UTC timestamp",  
  "status_code": "STATUS_CODE",  
}
```

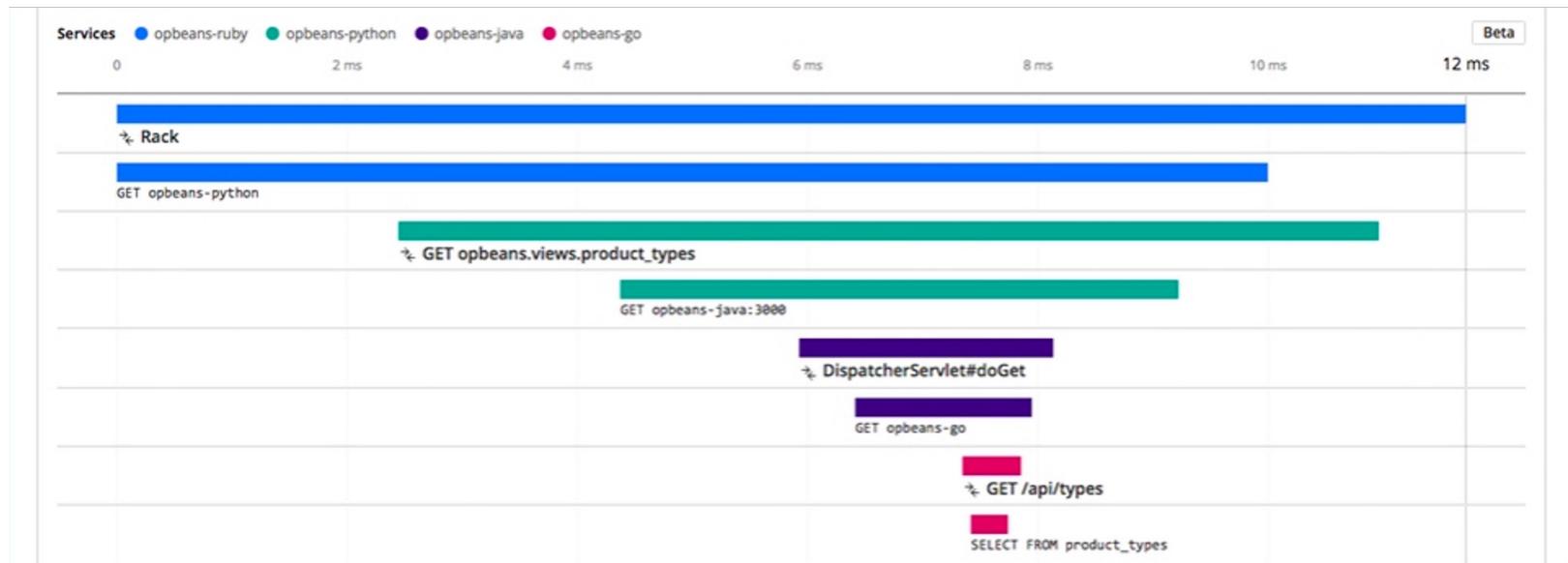
Span Context

Additional metadata that enables tracers to manage, correlate, and assemble spans.

Tracers



Viewing Traces



Tracers & Trace Context



Trace & Span Context

The necessary metadata that enables tracers to manage, correlate, and assemble spans.



Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracers & Trace Context



Trace & Span Context

The necessary metadata that enables tracers to manage, correlate, and assemble spans.

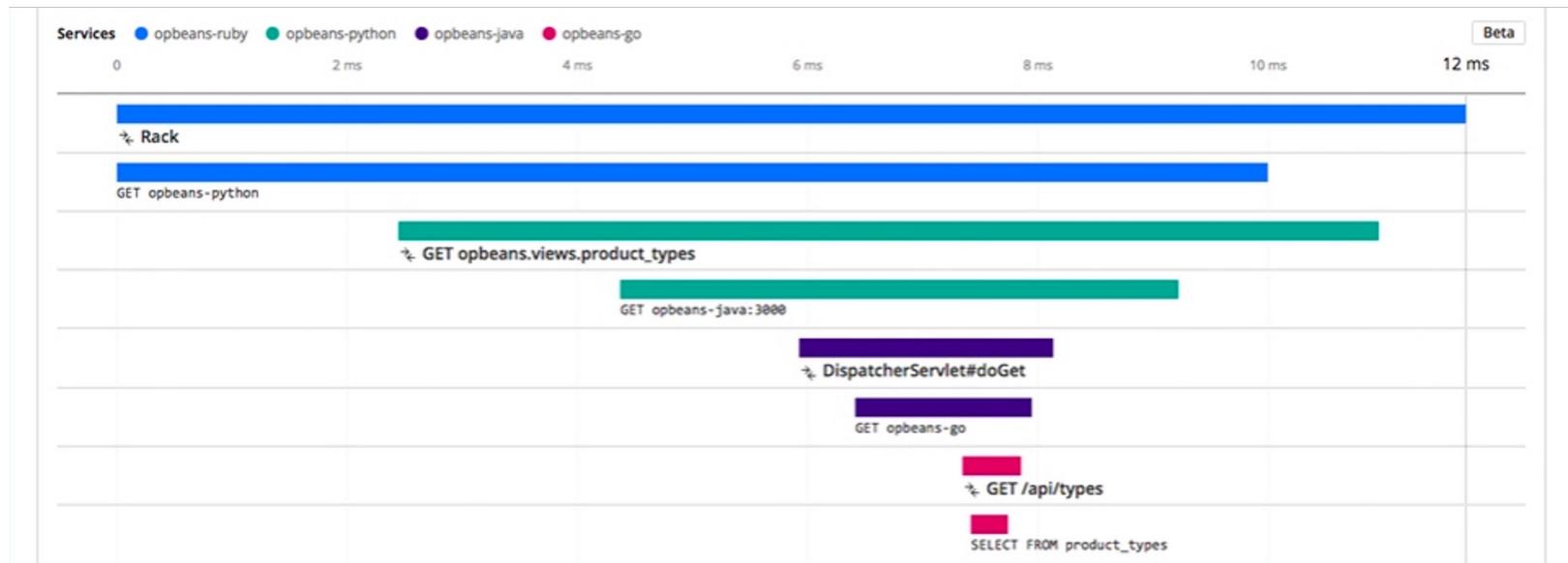


Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

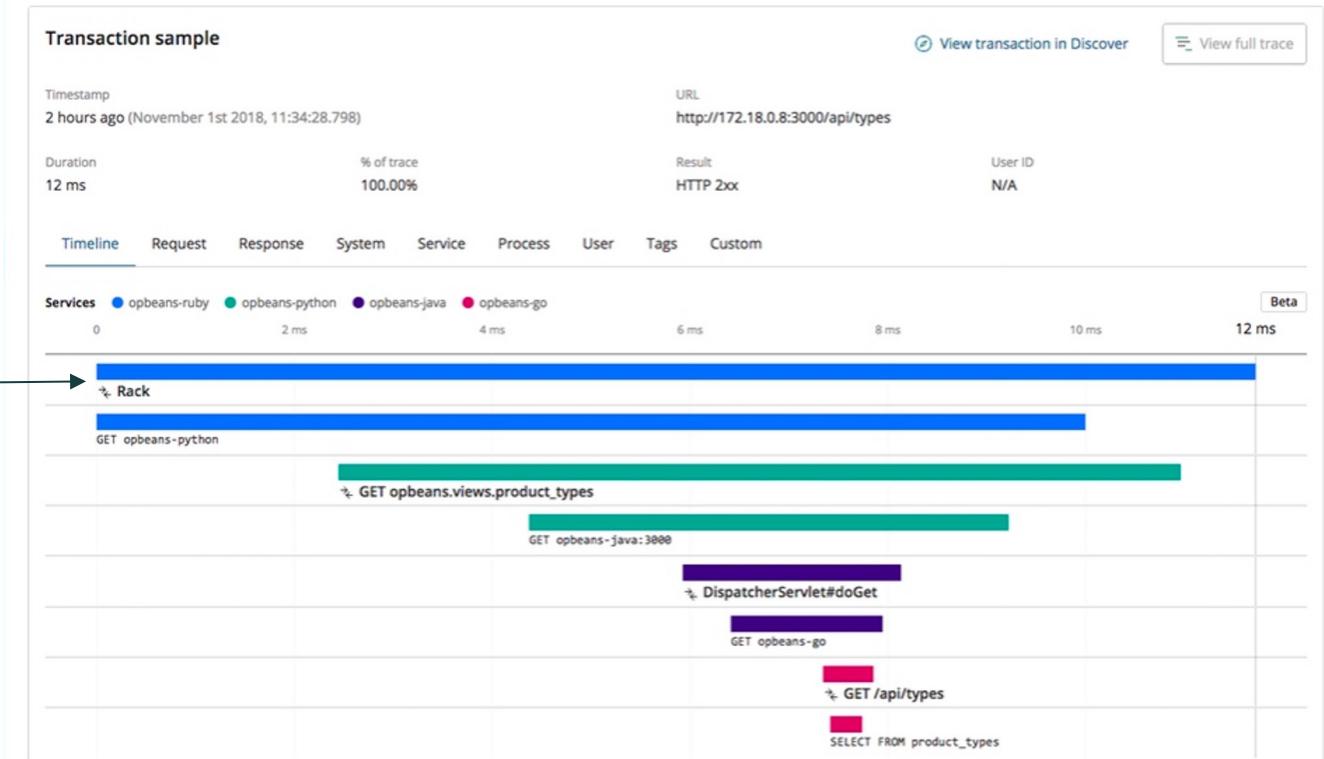


Viewing Traces



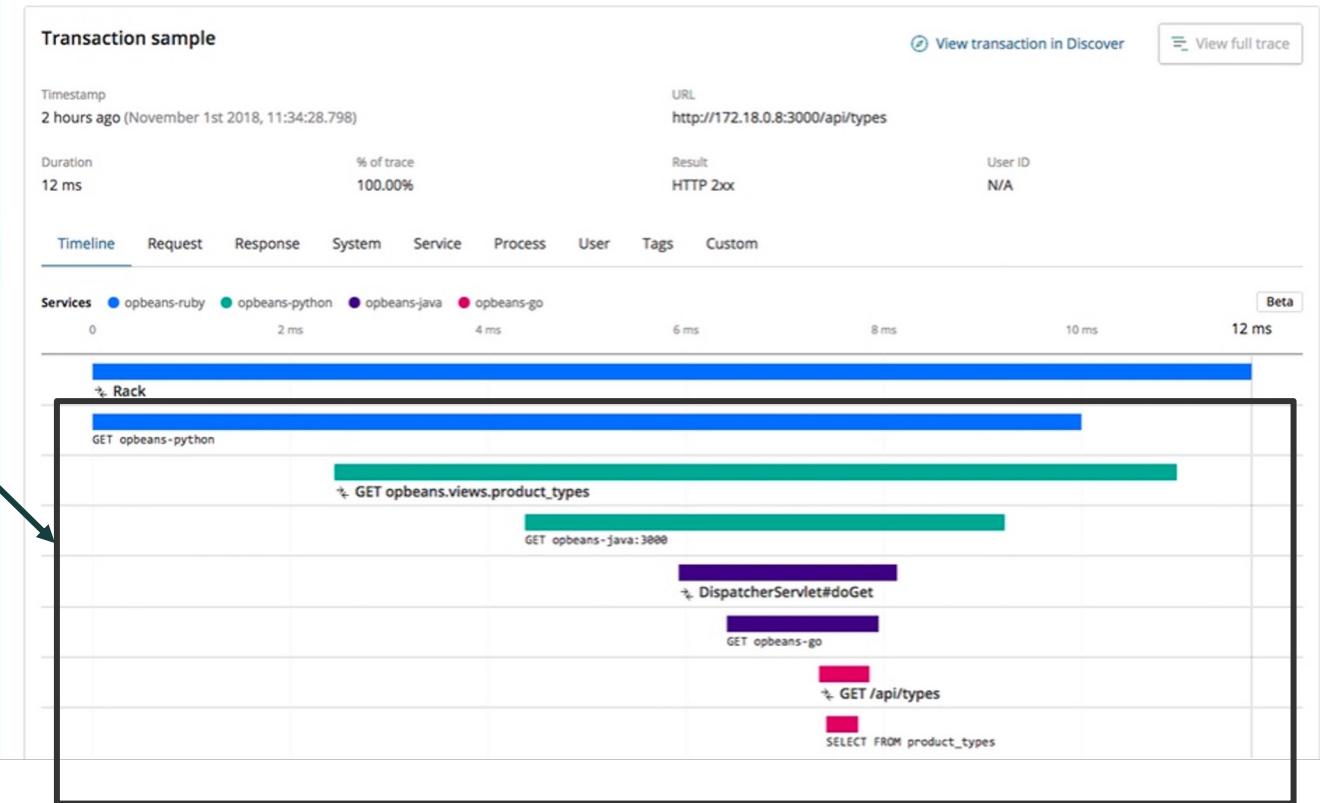


Viewing Traces: Spans



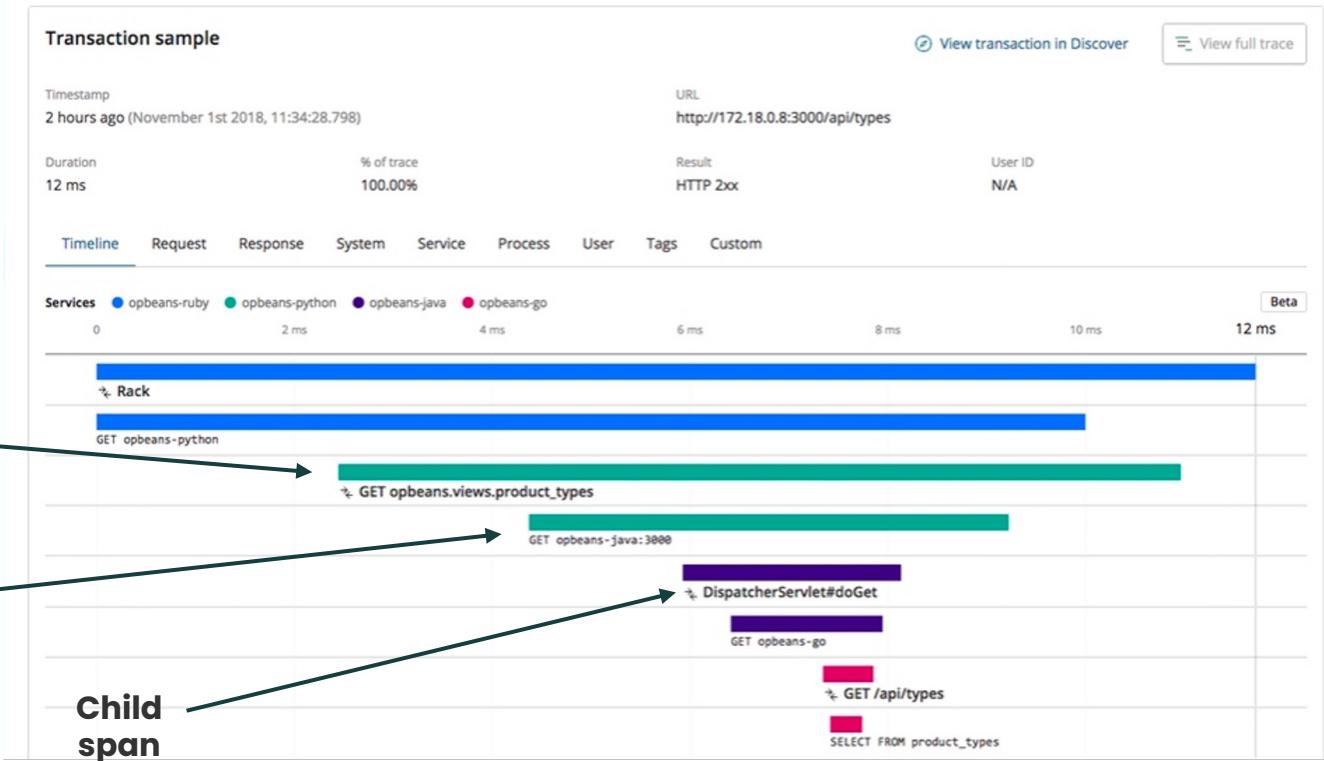


Viewing Traces: Spans



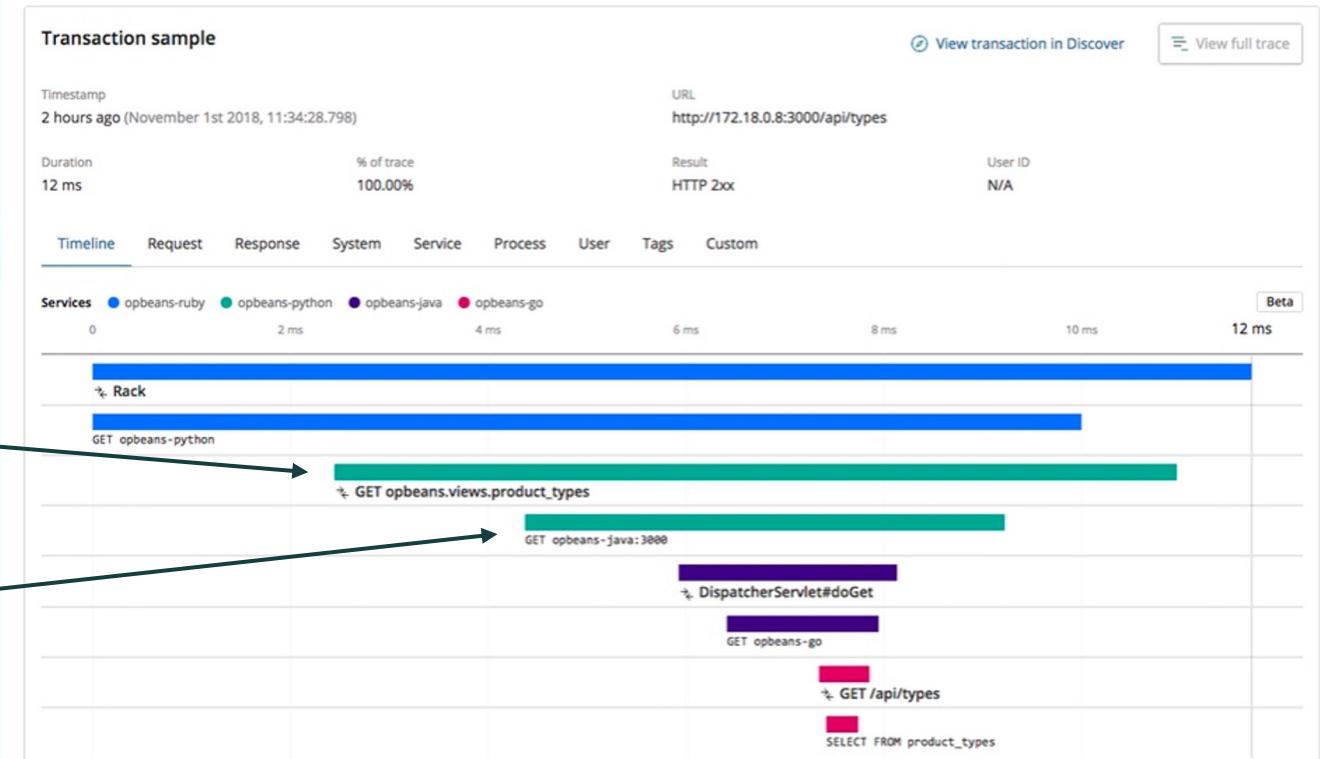


Viewing Traces: Spans





Viewing Traces: Spans

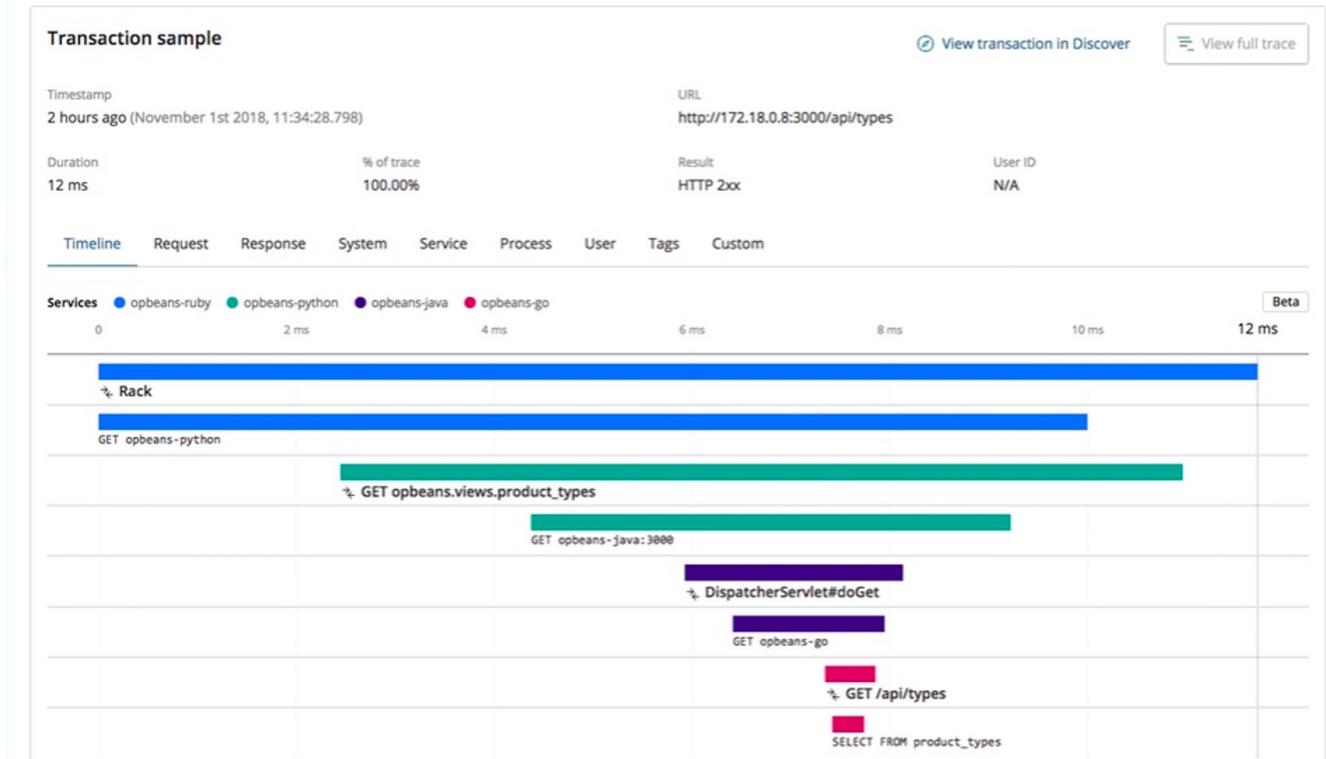




Q&A



Viewing Traces: Spans



Tracers & Trace Providers



Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.



Tracer Provider

Manages and provides Tracer *instances*.



Exercise

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-o11y/provider

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();

const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();

const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();

const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();

const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();
const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();

const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.

```
import { NodeTracerProvider } from '@opentelemetry/node';
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { SemanticResourceAttributes } from
  '@opentelemetry/semantic-conventions';

const tracerProvider = new NodeTracerProvider();
const consoleExporter = new ConsoleSpanExporter();

// Set global attributes
tracerProvider.setResources({
  [SemanticResourceAttributes.SERVICE_NAME]: 'service-name',
});

// Register the tracer provider globally
tracerProvider.register();

// Access the tracer
const tracer = tracerProvider.getTracer('service-name');

export default tracer;
```

Tracers

Responsible for creating & managing spans by providing a Tracer API for instrumentation libraries and managing context.

Tracer Provider

Manages and provides Tracer *instances*.



Q&A

Spans

Spans

Span Context

Span Kind

Span Status

Span Attributes

Span Links

Span Events

Spans

Span Context

Static object contained on every span that's propagated with trace context.

Span Kind

Span Attributes

Span Links

Span Status

Span Events

Spans

Span Context

Static object contained on every span that's propagated with trace context.

Span Kind

An Enum that defines different types of spans to categorize spans in relation to one another.

Span Status

Span Attributes

Span Links

Span Events

Span Kind

An *Enum* that defines different types of spans to categorize spans in relation to one another.

```
enum SpanKind {

    // Internal spans that don't fit other communication models.
    INTERNAL = 0,

    // Initiated by the receiving end of a client-server interaction.
    SERVER = 1,

    // Initiated by the sending end of a client-server interaction.
    CLIENT = 2,

    // Associated with message or event production.
    PRODUCER = 3,

    // Associated with message or event consumption.
    CONSUMER = 4,
}
```

Spans

Span Context

Static object contained on every span that's propagated with trace context.

Span Kind

An Enum that defines different types of spans to categorize spans in relation to one another.

Span Status

Denotes whether the span failed, completed, or unset.

Span Attributes

Span Links

Span Events

Span Status

Denotes whether the span failed, completed, or unset.

```
interface SpanStatus {
    // The status code
    code: SpanStatusCode;
    // Optional developer facing message
    message?: string;
}
```

```
enum SpanStatusCode {
    // Default status
    UNSET = 0,
    // Completed successfully
    OK = 1,
    // Contains an error
    ERROR = 2,
}
```

Spans

Span Context

Static object contained on every span that's propagated with trace context.

Span Kind

Communicates how the spans should be assembled together.

Span Status

Denotes whether the span failed, completed, or unset.

Span Attributes

Key-value pairs with metadata used to annotate a span with task-specific information.

Span Events

Span Links

Span Attributes

Key-value pairs with metadata used to annotate a span with task-specific information.

```
const opentelemetry = require('@opentelemetry/api');

const currentSpan = opentelemetry.getActiveSpan();

currentSpan.set_attribute("operation.value", 1);
```

Spans

Span Context

Static object contained on every span that's propagated with trace context.

Span Kind

An Enum that defines different types of spans to categorize spans in relation to one another.

Span Status

Denotes whether the span failed, completed, or unset.

Span Attributes

Key-value pairs with metadata used to annotate a span with task-specific information.

Span Events

Structured log message on a span used to denote a meaningful, singular point in time during the span's duration.

Span Links

Span Events

Optional information that creates causal relationships between spans across multiple traces.

```
const opentelemetry = require('@opentelemetry/api');

const currentSpan = opentelemetry.getActiveSpan();

currentSpan?.addEvent('Span Event', {
  'log.severity': 'error',
  'log.message': 'Data not found',
  'request.id': 'requestId',
});
```

Spans

Span Context

Static object contained on every span that's propagated with trace context.

Span Kind

An Enum that defines different types of spans to categorize spans in relation to one another.

Span Status

Denotes whether the span failed, completed, or unset.

Span Attributes

Key-value pairs with metadata used to annotate a span with task-specific information.

Span Events

Structured log message on a span used to denote a meaningful, singular point in time during the span's duration.

Span Links

Optional information that creates causal relationships between spans across multiple traces.



Exercise

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-olly/links

Span Links

Optional information that creates causal relationships between spans across multiple traces.

```
const opentelemetry = require('@opentelemetry/api');

const currentSpan = opentelemetry.getActiveSpan();

const options = {
  links: [
    {
      context: currentSpan.spanContext()
    }
  ]
};

// Start a new span
const span = opentelemetry.tracer.startActiveSpan('Span
Link', options, span => {
  // Do Something
  span.end();
}) ;=
```



Discussion

“Are there any tracing metadata standards that would be useful for your organization?”

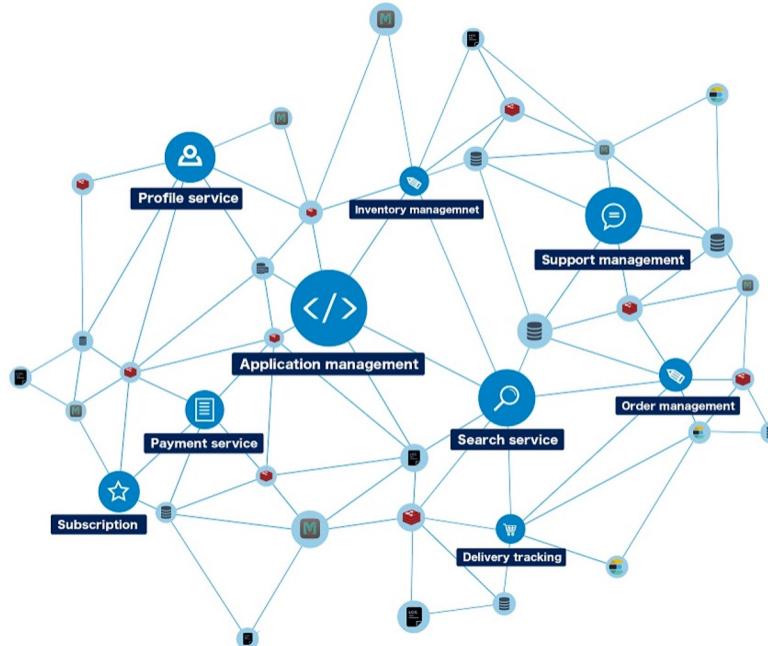


Discussion, follow-up

“Given those, how would you can incorporate those with OpenTelemetry?”



Service Graphs



Telemetry



Distributed Traces

Tracks & provides the context & flow of tasks as they move through different system components.



Metrics

Quantitative measures of a system's internal state that indicates performance and behavior.



Logs

Standard Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Standard context sharing

The set of rules for how context is transferred between different services.

Context Propagation

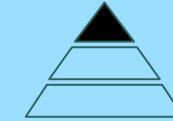
Context

Object that enables services communicating with one another to correlate spans with the same trace.

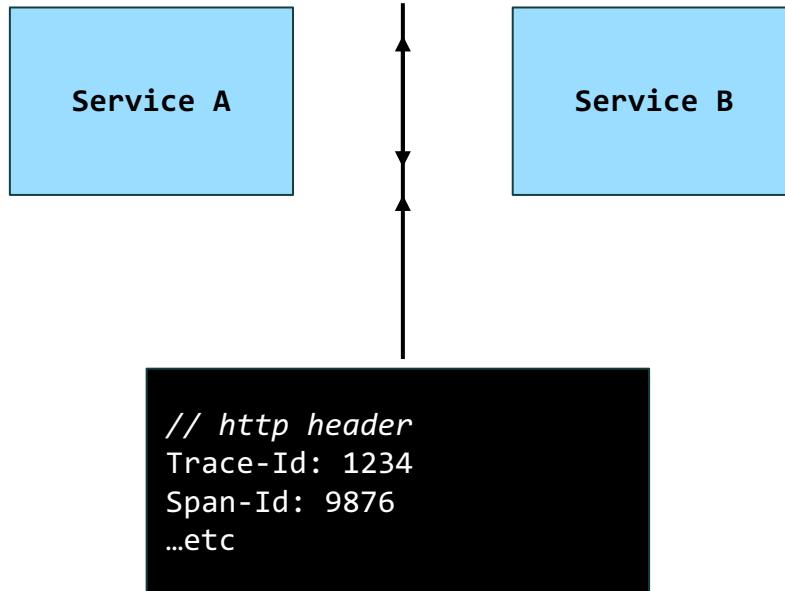
Propagation

Method for moving context between services & processes to assemble the full trace.

Standard Communication



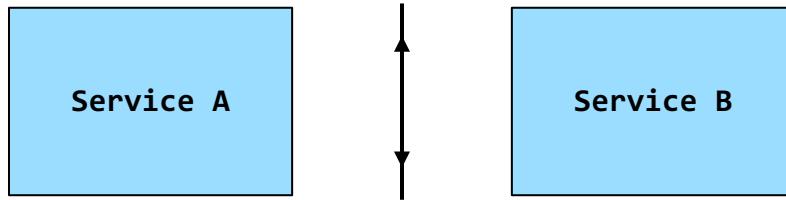
Patterns
Support
Stack



2. Standard context *sharing*

The set of rules for how context is transferred between different services.

Context Propagation



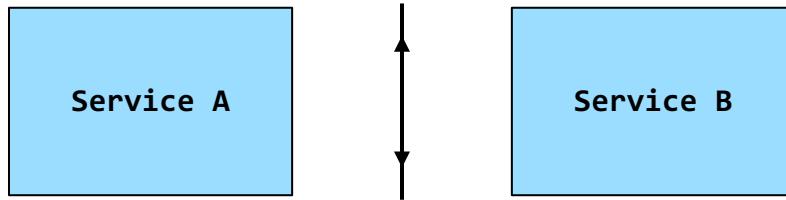
```
// w3c spec  
traceparent: version-traceid-parentid-flags  
tracestate: key1=value1,key2=value2,...
```

```
// b3 spec  
X-B3-TraceId: traceid  
X-B3-ParentSpanId: parentid  
X-B3-SpanId: spanId  
X-B3-Sampled: boolean
```

OpenTelemetry recommended standard

commonly supported, but less recommended

Context Propagation



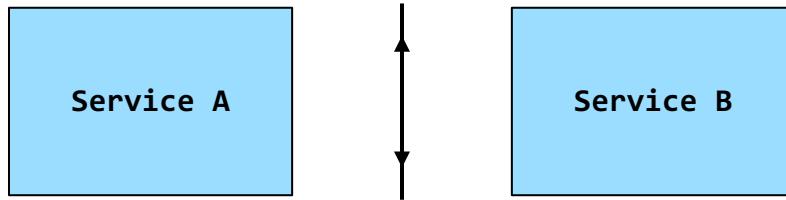
```
// w3c spec  
traceparent: version-traceid-parentid-flags  
tracestate: key1=value1,key2=value2,...
```

```
// b3 spec  
X-B3-TraceId: traceid  
X-B3-ParentSpanId: parentid  
X-B3-SpanId: spanId  
X-B3-Sampled: boolean
```

OpenTelemetry recommended standard

commonly supported, but less recommended

Context Propagation



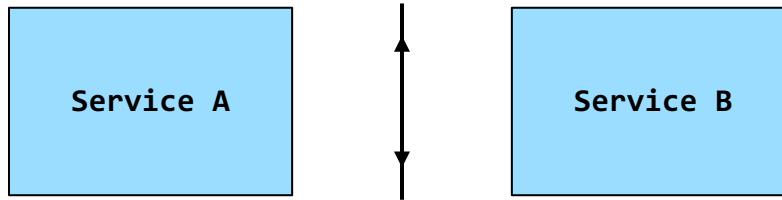
```
// w3c spec  
traceparent: version-traceid-parentid-flags  
tracestate: key1=value1,key2=value2,...
```

```
// b3 spec  
X-B3-TraceId: traceid  
X-B3-ParentSpanId: parentid  
X-B3-SpanId: spanId  
X-B3-Sampled: boolean
```

OpenTelemetry recommended standard

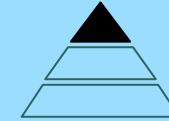
commonly supported, but less recommended

Context Propagation



```
// w3c spec  
traceparent: version-traceid-parentid-flags  
tracestate: key1=value1,key2=value2,...
```

Standard Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Standard context sharing

The set of rules for how context is transferred between different services.

3. Standard instrumentation techniques

Provide reliable & rich telemetry data by standardizing the ways we instrument.

Application Instrumentation Types



Auto-Instrumentation Libraries

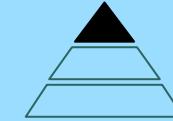
Simplifies telemetry collection by providing built-in instrumentation for supported frameworks and libraries.



Manual Instrumentation

Involves adding explicit code instrumentation and telemetry collection points to an application's codebase.

Standard Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Standard context sharing

The set of rules for how context is transferred between different services.

3. Standard instrumentation techniques

Provide reliable & rich telemetry data by standardizing the ways we instrument.

4. Well-defined interfaces

The interfaces for tools that enable collecting, processing, or analyzing telemetry data.



APIs

Standard specification &
interfaces for generating &
exporting telemetry
between systems.



APIs

Standard specification & interfaces for generating & exporting telemetry between systems.



SDKs

Language-specific tools for instrumentation and generating telemetry data using observability APIs.



Exercise

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-oly/auto

Auto-instrumentation with OTel SDKs



Patterns
Support
Stack

```
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { NodeSDK } from '@opentelemetry/sdk-node';
import * as otel from '@opentelemetry/auto-instrumentations-node';

const sdk = new NodeSDK({
  traceExporter: new ConsoleSpanExporter(),
  instrumentations: [
    otel.getNodeAutoInstrumentations(),
  ]
});

sdk.start();
```

Auto-instrumentation with OTel SDKs



Patterns
Support
Stack

```
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { NodeSDK } from '@opentelemetry/sdk-node';
import * as otel from '@opentelemetry/auto-instrumentations-node';

const sdk = new NodeSDK({
  traceExporter: new ConsoleSpanExporter(),
  instrumentations: [
    otel.getNodeAutoInstrumentations(),
  ]
});

sdk.start();
```

Auto-instrumentation with OTel SDKs



Patterns
Support
Stack

```
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { NodeSDK } from '@opentelemetry/sdk-node';
import * as otel from '@opentelemetry/auto-instrumentations-node';

const sdk = new NodeSDK({
  traceExporter: new ConsoleSpanExporter(),
  instrumentations: [
    otel.getNodeAutoInstrumentations(),
  ]
});

sdk.start();
```

Auto-instrumentation with OTel SDKs



Patterns
Support
Stack

```
import { ConsoleSpanExporter } from '@opentelemetry/tracing';
import { NodeSDK } from '@opentelemetry/sdk-node';
import * as otel from '@opentelemetry/auto-instrumentations-node';

const sdk = new NodeSDK({
  traceExporter: new ConsoleSpanExporter(),
  instrumentations: [
    otel.getNodeAutoInstrumentations(),
  ]
});

sdk.start();
```



APIs

Standard specification & interfaces for generating & exporting telemetry between systems.



SDKs

Language-specific tools for instrumentation and generating telemetry data using observability APIs.



Agents

Software components that run on a system to automatically collect & transmit telemetry without manual code changes.



Exercise

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-olly/instrument

Telemetry



Distributed Traces

Tracks & provides the context & flow of tasks as they move through different system components.



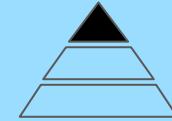
Metrics

Quantitative measures of a system's internal state that indicates performance and behavior.



Logs

Standardized Communication

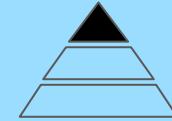


Specs
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

Standardized Communication



Specs
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

Measurement Specs

```
{  
  name: String;  
  description: String;  
  unit: String;  
  kind: InstrumentType;  
  valueType: ValueType;  
}
```

```
enum InstrumentType {  
  COUNTER = 'COUNTER',  
  HISTOGRAM = 'HISTOGRAM',  
  UP_DOWN_COUNTER = 'UP_DOWN_COUNTER',  
  OBSERVABLE_COUNTER = 'OBSERVABLE_COUNTER',  
  OBSERVABLE_GAUGE = 'OBSERVABLE_GAUGE',  
  OBSERVABLE_UP_DOWN_COUNTER = 'OBSERVABLE_UP_DOWN_COUNTER',  
}  
  
enum ValueType {  
  INT,  
  DOUBLE,  
}
```

Measurement Specs

```
{  
  name: String;  
  description: String;  
  unit: String;  
  kind: InstrumentType;  
  valueType: ValueType;  
}
```

```
enum InstrumentType {  
  COUNTER = 'COUNTER',  
  HISTOGRAM = 'HISTOGRAM',  
  UP_DOWN_COUNTER = 'UP_DOWN_COUNTER',  
  OBSERVABLE_COUNTER = 'OBSERVABLE_COUNTER',  
  OBSERVABLE_GAUGE = 'OBSERVABLE_GAUGE',  
  OBSERVABLE_UP_DOWN_COUNTER = 'OBSERVABLE_UP_DOWN_COUNTER',  
}  
  
enum ValueType {  
  INT,  
  DOUBLE,  
}
```

Measurement Specs

```
{
  name: String;
  description: String;
  unit: String;
  kind: InstrumentType;
  valueType: ValueType;
}
```

```
enum InstrumentType {
  COUNTER = 'COUNTER',
  HISTOGRAM = 'HISTOGRAM',
  UP_DOWN_COUNTER = 'UP_DOWN_COUNTER',
  OBSERVABLE_COUNTER = 'OBSERVABLE_COUNTER',
  OBSERVABLE_GAUGE = 'OBSERVABLE_GAUGE',
  OBSERVABLE_UP_DOWN_COUNTER = 'OBSERVABLE_UP_DOWN_COUNTER',
}

enum ValueType {
  INT,
  DOUBLE,
}
```

Measurement Specs

```
{  
  name: String;  
  description: String;  
  unit: String;  
  kind: InstrumentType;  
  valueType: ValueType;  
}
```

```
enum InstrumentType {  
  COUNTER = 'COUNTER',  
  HISTOGRAM = 'HISTOGRAM',  
  UP_DOWN_COUNTER = 'UP_DOWN_COUNTER',  
  OBSERVABLE_COUNTER = 'OBSERVABLE_COUNTER',  
  OBSERVABLE_GAUGE = 'OBSERVABLE_GAUGE',  
  OBSERVABLE_UP_DOWN_COUNTER = 'OBSERVABLE_UP_DOWN_COUNTER',  
}  
  
enum ValueType {  
  INT,  
  DOUBLE,  
}
```

Measurement Specs

```
{  
  name: String;  
  description: String;  
  unit: String;  
  kind: InstrumentType;  
  valueType: ValueType;  
}
```

```
enum InstrumentType {  
  COUNTER = 'COUNTER',  
  HISTOGRAM = 'HISTOGRAM',  
  UP_DOWN_COUNTER = 'UP_DOWN_COUNTER',  
  OBSERVABLE_COUNTER = 'OBSERVABLE_COUNTER',  
  OBSERVABLE_GAUGE = 'OBSERVABLE_GAUGE',  
  OBSERVABLE_UP_DOWN_COUNTER = 'OBSERVABLE_UP_DOWN_COUNTER',  
}  
  
enum ValueType {  
  INT,  
  DOUBLE,  
}
```

Types of Metrics

Counters

Gauges

Histograms

Types of Metrics

Counters

Values that accumulate over time, e.g., number of 200 status code returns.

Gauges

Histograms

Types of Metrics

Counters

Values that accumulate over time, e.g., number of 200 status code returns.

Gauges

Current value at read time, e.g., CPU usage.

Histograms

Types of Metrics

Counters

Values that accumulate over time, e.g., number of 200 status code returns.

Gauges

Current value at read time, e.g., CPU usage.

Histograms

Client-side aggregation of values, e.g., distribution of request status codes.

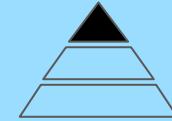
Measurement Specs

```
{
  name: String;
  description: String;
  unit: String;
  kind: InstrumentType;
  valueType: ValueType;
}
```

```
enum InstrumentType {
  COUNTER = 'COUNTER',
  HISTOGRAM = 'HISTOGRAM',
  UP_DOWN_COUNTER = 'UP_DOWN_COUNTER',
  OBSERVABLE_COUNTER = 'OBSERVABLE_COUNTER',
  OBSERVABLE_GAUGE = 'OBSERVABLE_GAUGE',
  OBSERVABLE_UP_DOWN_COUNTER = 'OBSERVABLE_UP_DOWN_COUNTER',
}

enum ValueType {
  INT,
  DOUBLE,
}
```

Standardized Communication

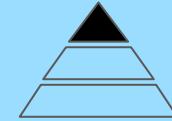


Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

Standardized Communication



Patterns
Support
Stack

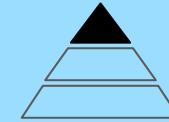
1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Standard context sharing

The set of rules for how context is transferred between different services.

Standardized Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Standard context sharing

The set of rules for how context is transferred between different services.

3. Standard instrumentation techniques

Provide reliable & rich telemetry data by standardizing the ways we instrument.

4. Well-defined interfaces

The interfaces for tools that enable collecting, processing, or analyzing telemetry data.

Meters & Meter Providers

Meters

Responsible for creating & managing Instruments (or measurements) by providing an API.

Meter Providers

Manages and provides configuration for Meter instances.

Meters & Meter Providers

Meters

Responsible for creating & managing Instruments (or measurements) by providing an API.

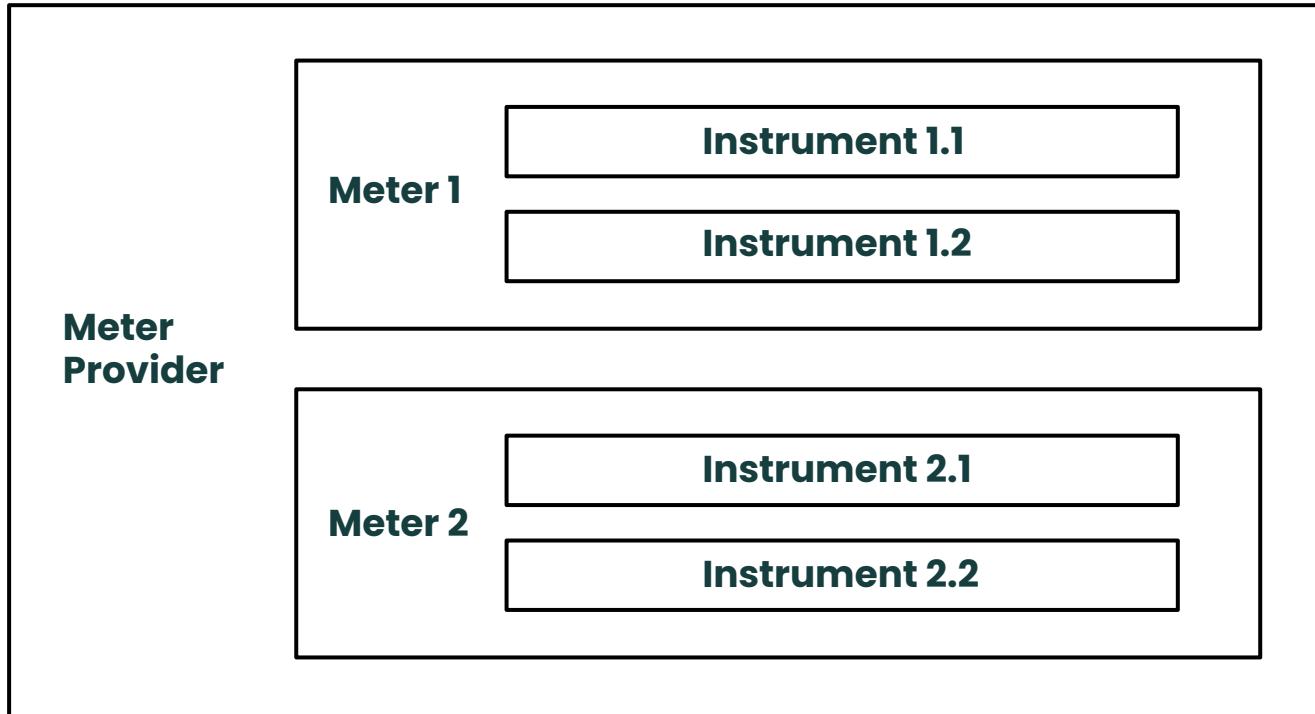
Meter Providers

Manages and provides configuration for Meter instances.

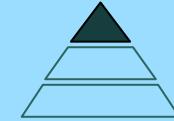
Instruments, Meters, & MeterProviders



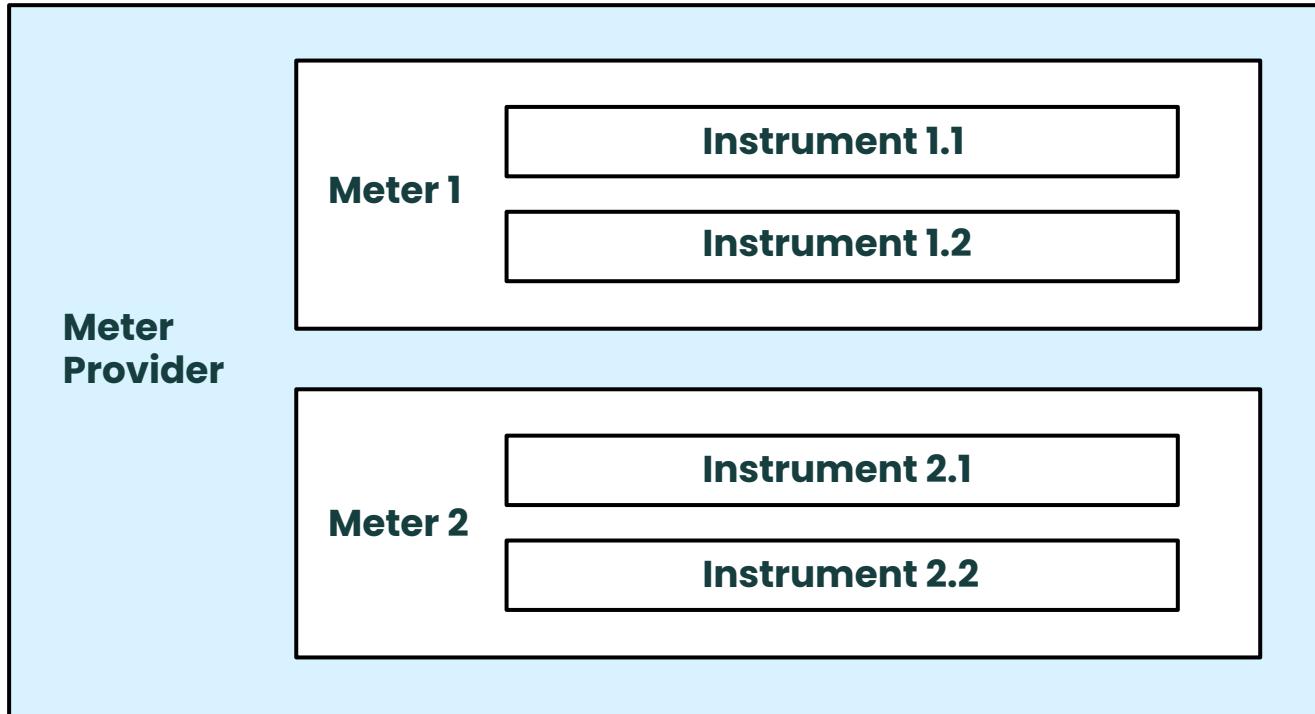
Specs
Tooling
Community



Instruments, Meters, & MeterProviders



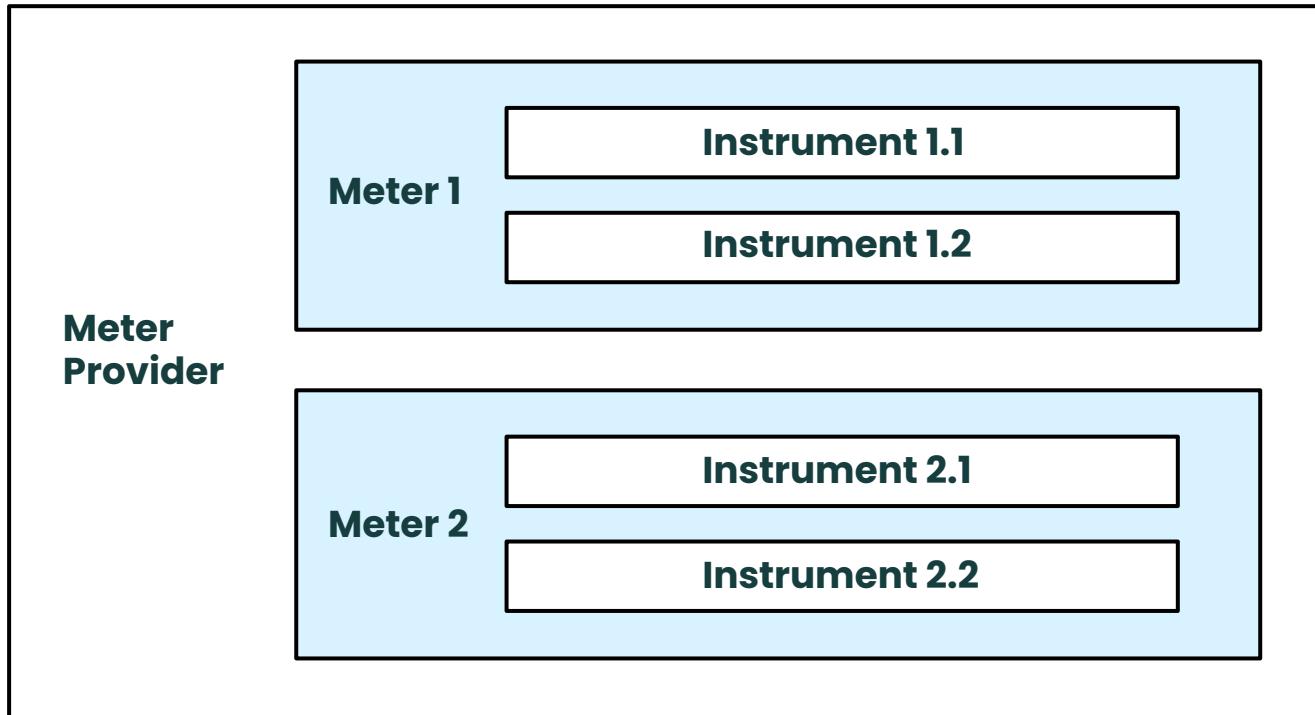
Specs
Tooling
Community



Instruments, Meters, & MeterProviders



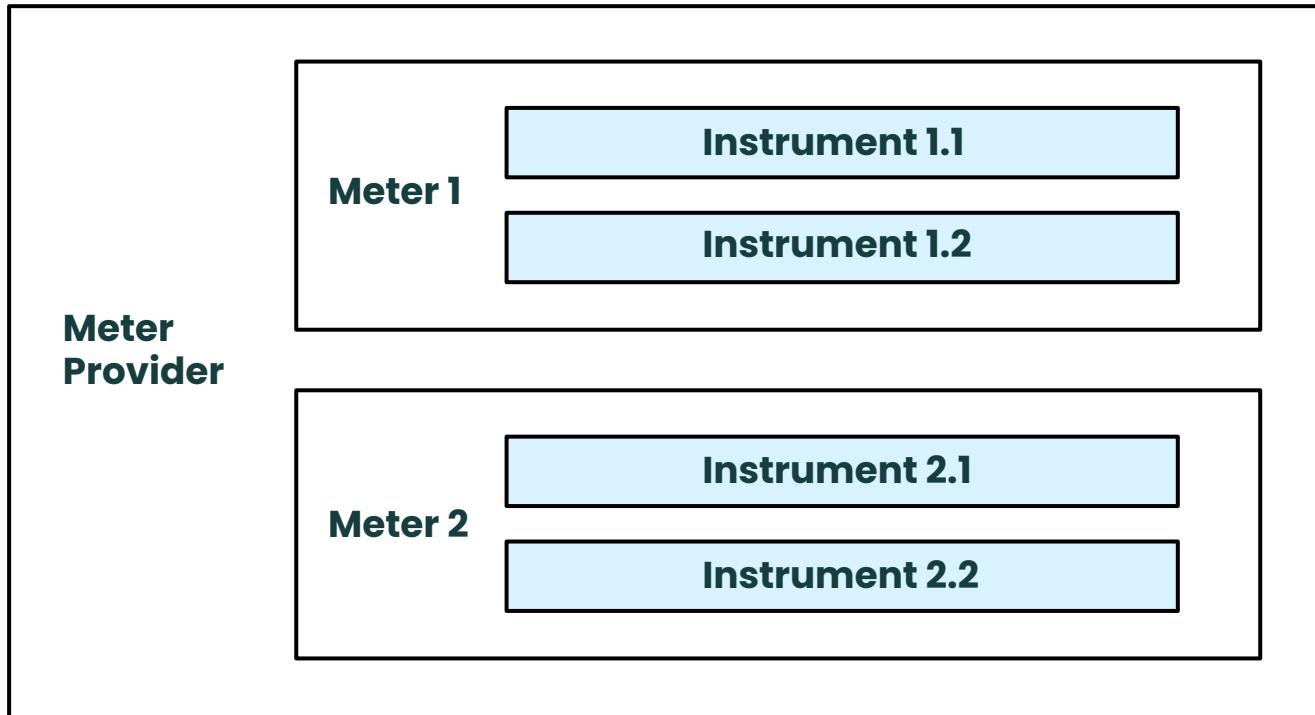
Specs
Tooling
Community



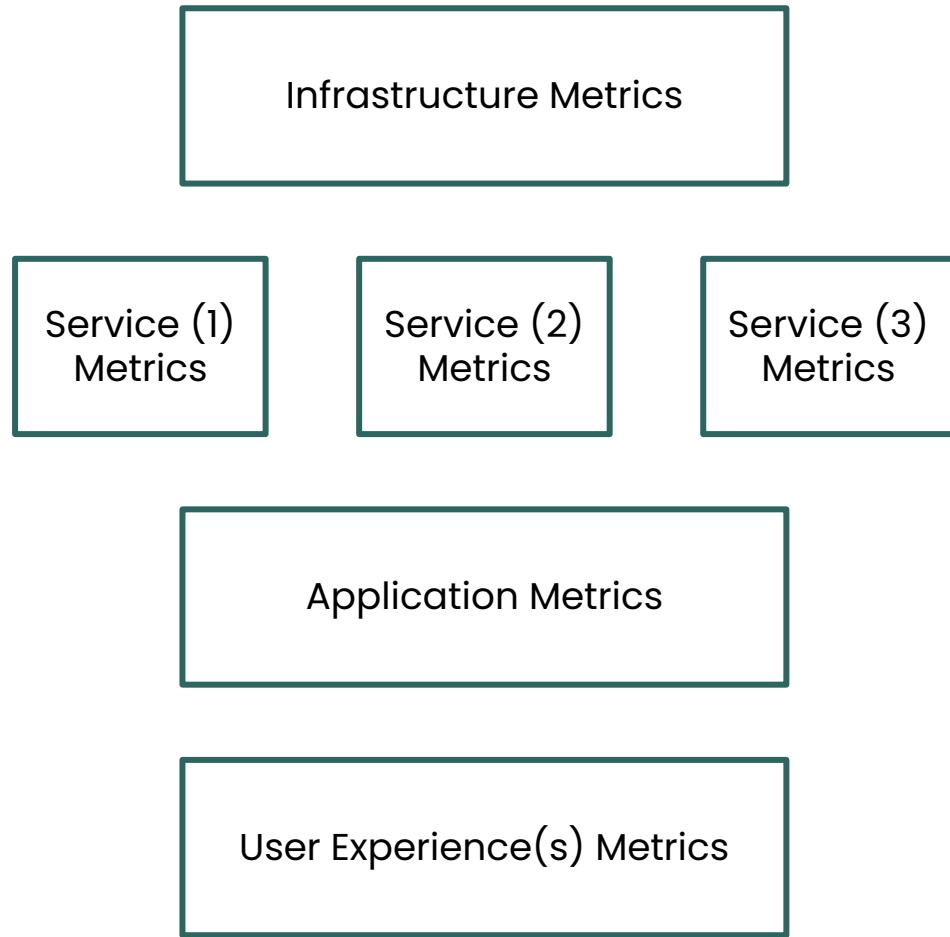
Instruments, Meters, & MeterProviders



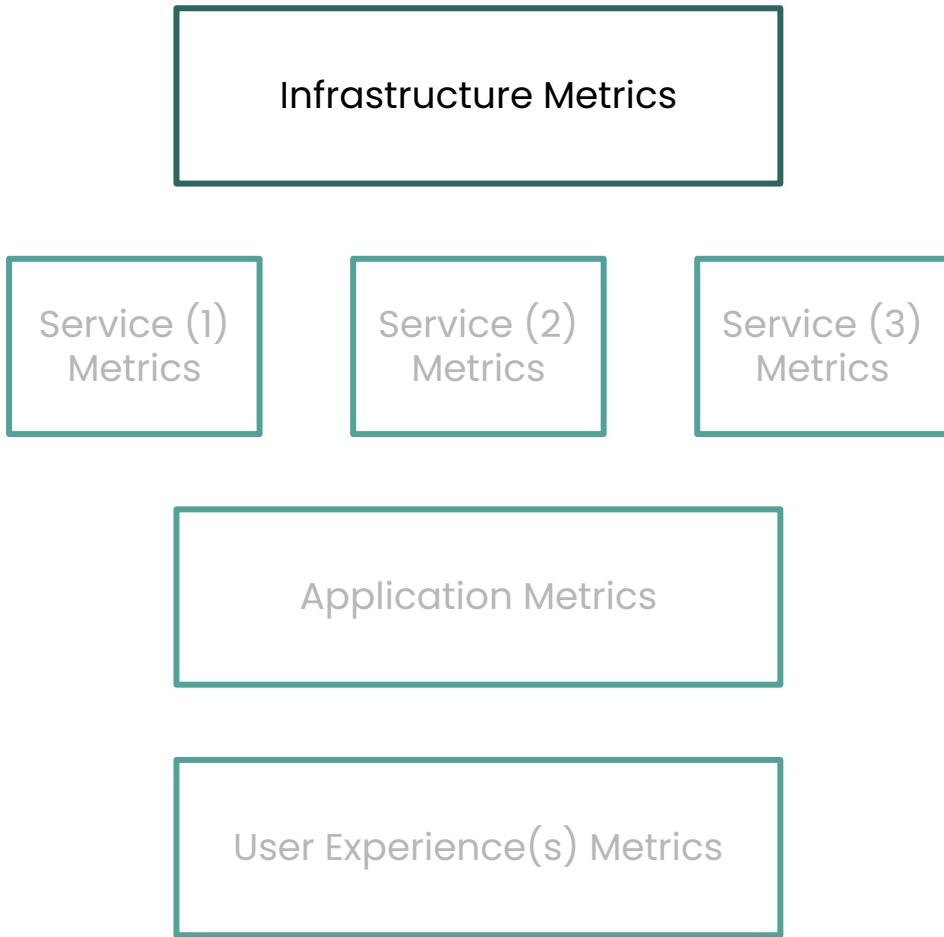
Specs
Tooling
Community



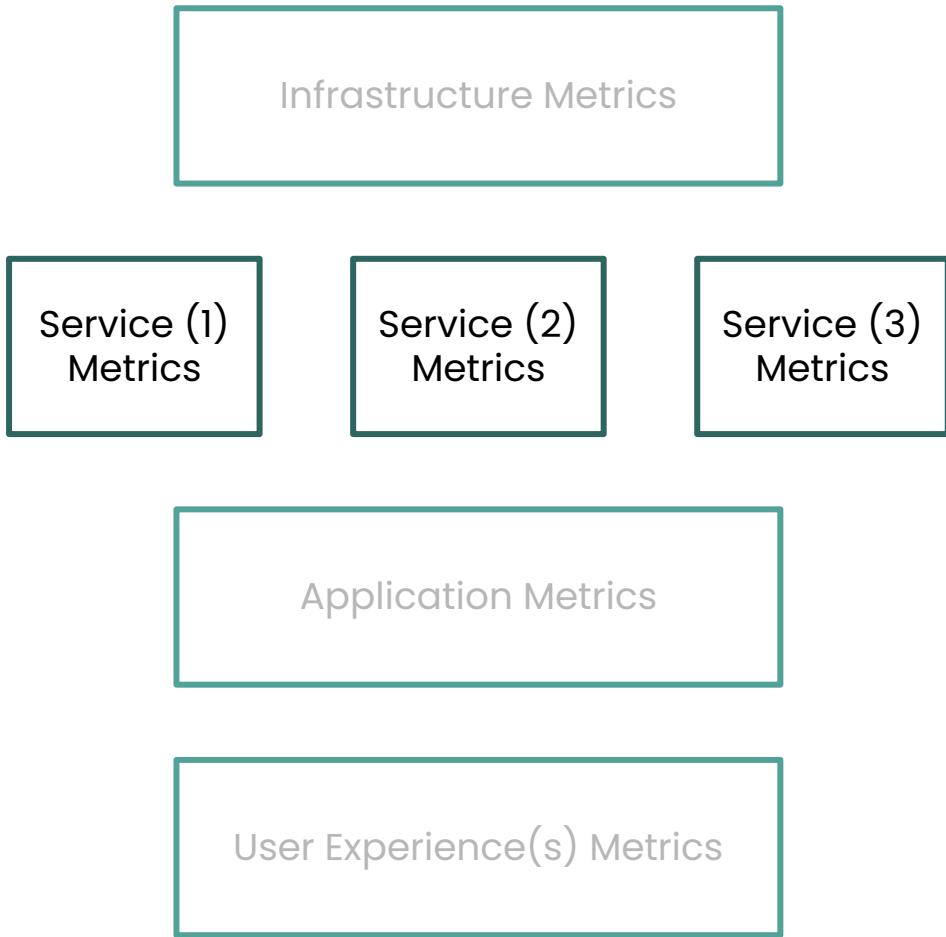
Metric Categories



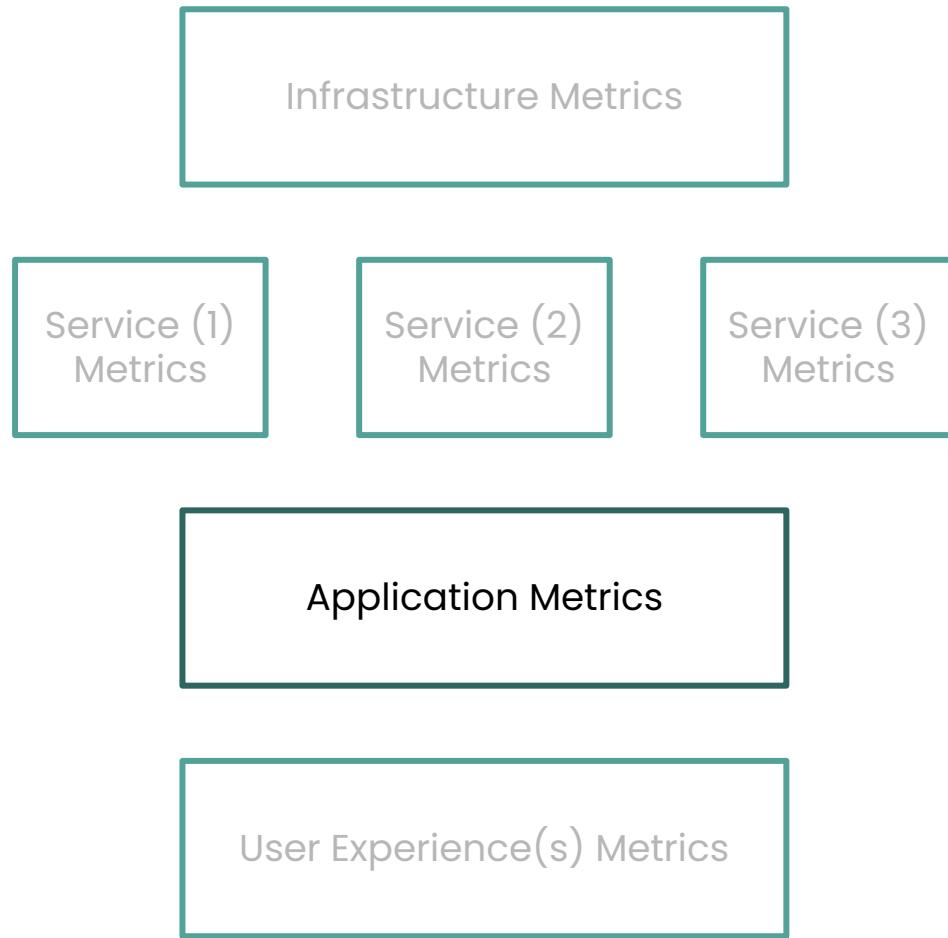
Metric Categories: Infra Layer



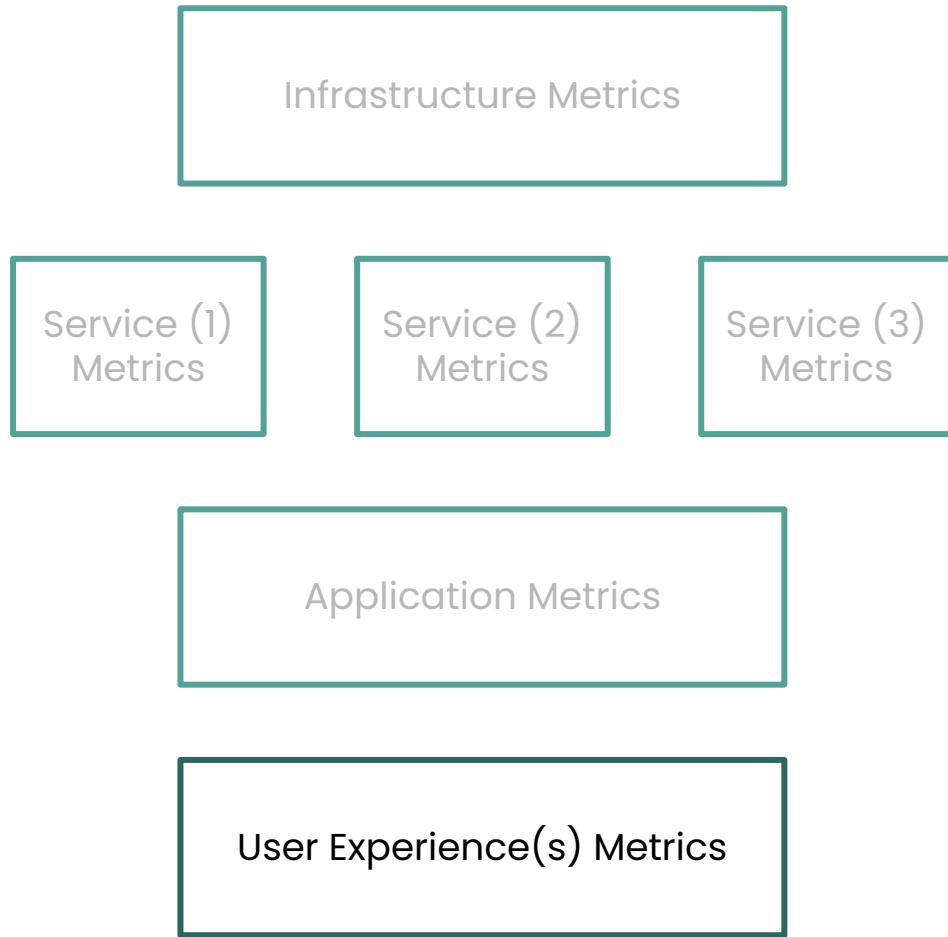
Metric Categories: Service Layer



Metric Categories: App Layer



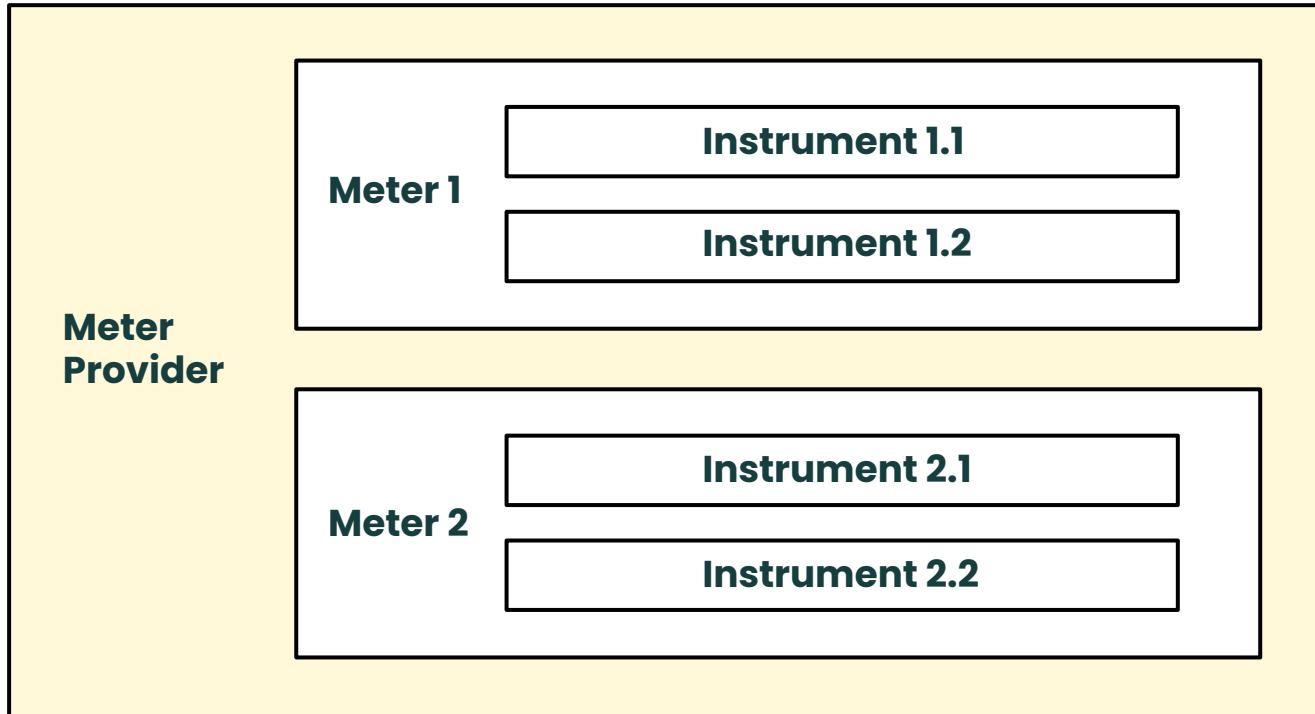
Metric Categories: User Experience



Instruments, Meters, & MeterProviders



Specs
Tooling
Community



MeterProvider Example

```
import otel from "@opentelemetry/api";
import { MeterProvider } from '@opentelemetry/sdk-metrics';

// Initialize MeterProvider, similar to TracerProvider({})
export const meterProvider = new MeterProvider({});

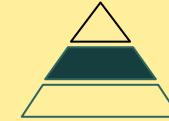
// Retrieve a Meter instance
const myMeter = otel.metrics.getMeter(
  'my-service-meter'
);

// Create a Counter Instrument
const counter = myMeter.createCounter('events.counter');

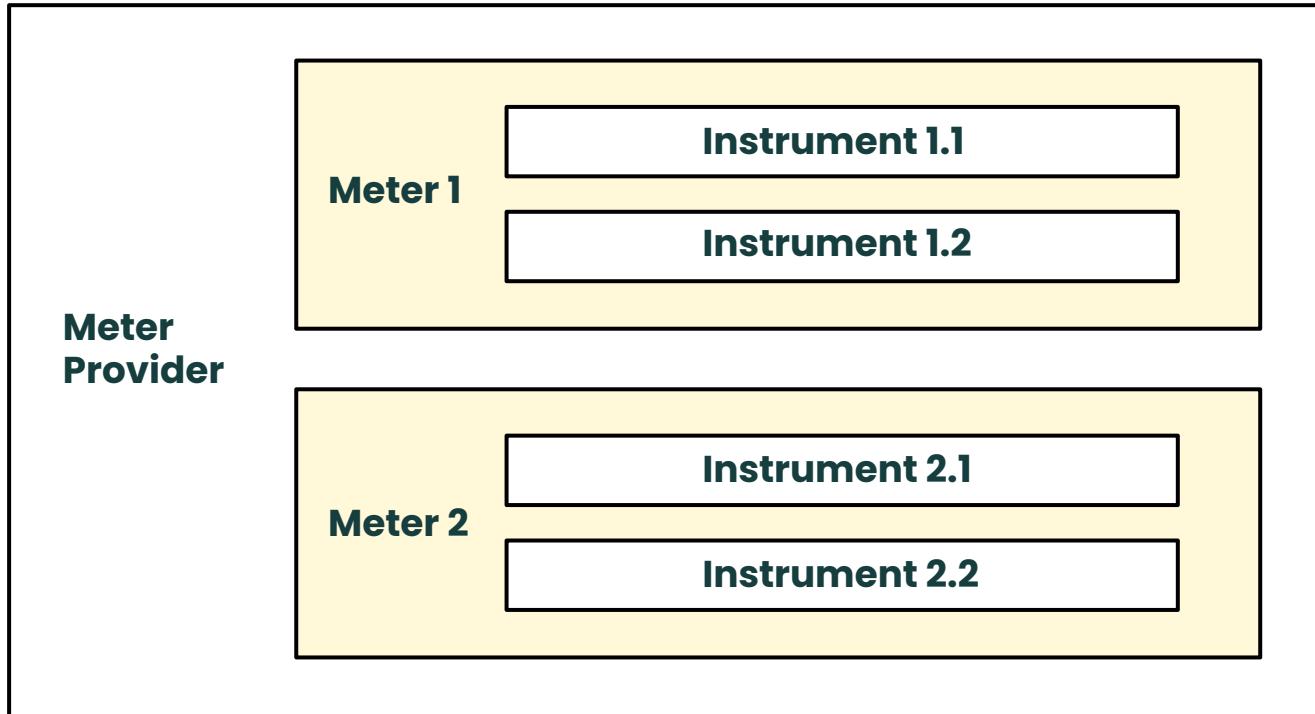
// Example Usage - Counter
export const hello = (_req: Request, res: Response): void => {
  // Increment the Counter
  counter.add(1)

  const data = { message: 'Hello World!' };
  res.status(200).json(data);
}
```

Instruments, Meters, & MeterProviders



Specs
Tooling
Community



Meter Example

```
import otel from "@opentelemetry/api";
import { MeterProvider } from '@opentelemetry/sdk-metrics';

// Initialize MeterProvider, similar to TracerProvider({})
export const meterProvider = new MeterProvider({});

// Retrieve a Meter instance
const myMeter = otel.metrics.getMeter(
  'my-service-meter'
);

// Create a Counter Instrument
const counter = myMeter.createCounter('events.counter');

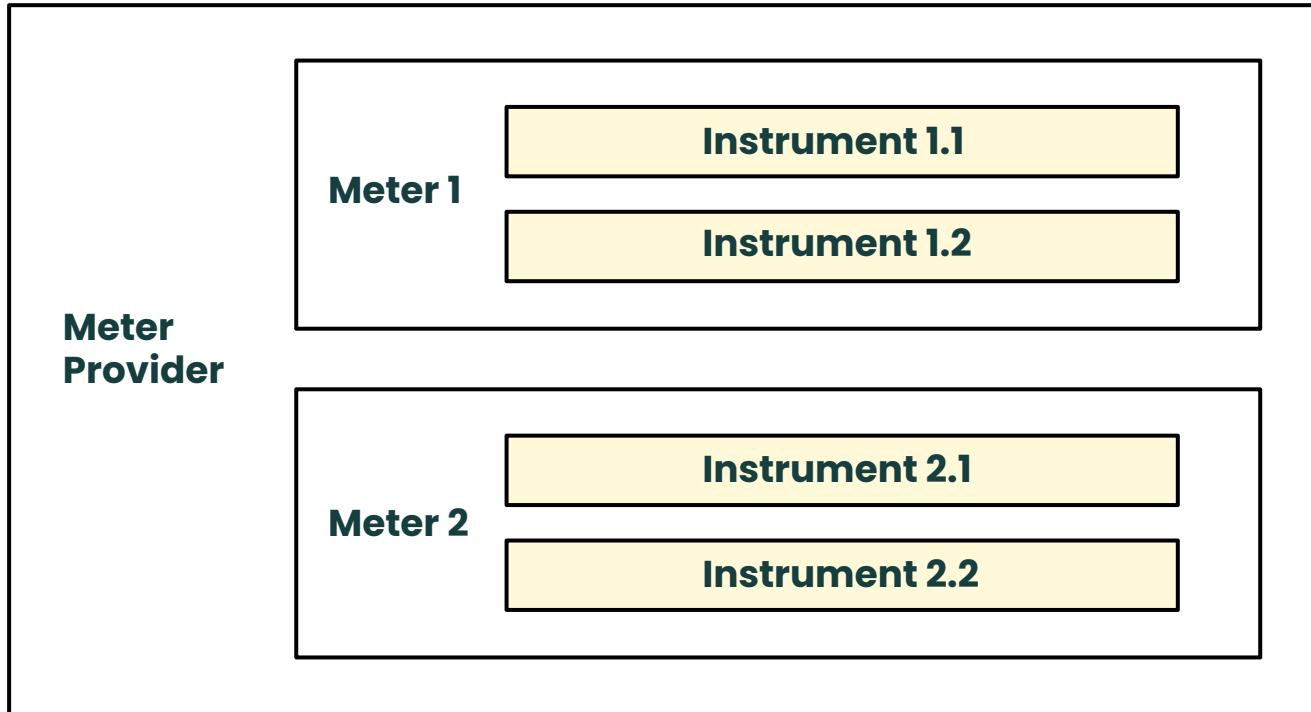
// Example Usage - Counter
export const hello = (_req: Request, res: Response): void => {
  // Increment the Counter
  counter.add(1)

  const data = { message: 'Hello World!' };
  res.status(200).json(data);
}
```

Instruments, Meters, & MeterProviders



Specs
Tooling
Community



Types of Metrics



Counters

Values that accumulate over time, e.g., number of 200 status code returns.

Gauges

Histograms

Instrument Example

Creating a Counter

```
import otel from "@opentelemetry/api";
import { MeterProvider } from '@opentelemetry/sdk-metrics';

// Initialize MeterProvider, similar to TracerProvider({})
export const meterProvider = new MeterProvider({});

// Retrieve a Meter instance
const myMeter = otel.metrics.getMeter(
  'my-service-meter'
);

// Create a Counter Instrument
const counter = myMeter.createCounter('events.counter');

// Example Usage - Counter
export const hello = (_req: Request, res: Response): void => {
  // Increment the Counter
  counter.add(1)

  const data = { message: 'Hello World!' };
  res.status(200).json(data);
}
```

Usage Example

Increment a Counter

```
import otel from "@opentelemetry/api";
import { MeterProvider } from '@opentelemetry/sdk-metrics';

// Initialize MeterProvider, similar to TracerProvider({})
export const meterProvider = new MeterProvider({});

// Retrieve a Meter instance
const myMeter = otel.metrics.getMeter(
  'my-service-meter'
);

// Create a Counter Instrument
const counter = myMeter.createCounter('events.counter');

// Example Usage - Counter
export const hello = (_req: Request, res: Response): void => {
  // Increment the Counter
  counter.add(1)

  const data = { message: 'Hello World!' };
  res.status(200).json(data);
}
```

Types of Metrics



Counters

Values that accumulate over time, e.g., number of 200 status code returns.

Gauges

Current value at read time, e.g., CPU usage.

Histograms

Instrument Example

Creating a Gauge

```
import otel from "@opentelemetry/api";

/**
 * Previous MeterProvider & Meter code here
 */

// Create a Gauge Instrument
const gauge = myMeter.createObservableGauge('temperature.gauge');

// Example Usage - Guage
export const hello = (_req: Request, res: Response): void => {
    let value = 32;

    gauge.addCallback((result) => {
        // Record value to Guadge
        result.observe(temperature);
    });
}

const data = { message: 'Hello World!' };
res.status(200).json(data);
}
```

Usage Example

Record to a Guage

```
import otel from "@opentelemetry/api";

/**
 Previous MeterProvider & Meter code here
**/

// Create a Gauge Instrument
const gauge = myMeter.createObservableGauge('temperature.gauge');

// Example Usage - Guage
export const hello = (_req: Request, res: Response): void => {
    let value = 32;

    gauge.addCallback((result) => {
        // Record value to Guadge
        result.observe(temperature);
    });

    const data = { message: 'Hello World!' };
    res.status(200).json(data);
}
```

Types of Metrics



Counters

Values that accumulate over time, e.g., number of 200 status code returns.

Gauges

Current value at read time, e.g., CPU usage.

Histograms

Client-side aggregation of values, e.g., distribution of request status codes.

Instrument Example

Creating a Histogram

```
import otel from "@opentelemetry/api";

/**
 * Previous MeterProvider & Meter code here
 */

// Create a Histogram Instrument
const histogram = myMeter.createHistogram('task.duration');

// Example Usage - Histogram
export const hello = (_req: Request, res: Response): void => {
    const startTime = new Date().getTime()
    const endTime = new Date().getTime()
    const executionTime = endTime - startTime

    // Record value to histogram
    histogram.record(executionTime)

    const data = { message: 'Hello World!' };
    res.status(200).json(data);
}
```

Usage Example

Record to a Histogram

```
import otel from "@opentelemetry/api";

/**
 Previous MeterProvider & Meter code here
**/

// Create a Histogram Instrument
const histogram = myMeter.createHistogram('task.duration');

// Example Usage - Histogram
export const hello = (_req: Request, res: Response): void => {
    const startTime = new Date().getTime()
    const endTime = new Date().getTime()
    const executionTime = endTime - startTime

    // Record value to histogram
    histogram.record(executionTime)

    const data = { message: 'Hello World!' };
    res.status(200).json(data);
}
```



Exercise

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-011y/metric



Discussion

“Are there any metrics metadata standards that would be useful for your organization?”



Discussion, follow-up

“Given those, how would you can incorporate those with OpenTelemetry?”

O'REILLY®

Day 2: Observability in Microservice Architectures

Lesley Cordero





Agenda

Day 1:

- Observability & role in microservice architectures
- Standardization & OpenTelemetry
- Telemetry signals & standards

Day 2:

- Sociotechnical challenges of observability
- Architecture challenges of observability
- Observability platform architecture

Themes:

- Standardization & platformization
- Other best practices
- Sociotechnical approaches



Takeaways

- Observability addresses the challenges of debugging microservices through a sociotechnical approach to telemetry collection and understandability.
- Standardization & Platformization are effective techniques for driving observability in your organization.
- Reduce technical efforts through the OpenTelemetry standard.
- Three telemetry signals: Distributed Traces, Metrics, & Logs
 - Best practices, implementation, & challenges.
- Connecting & correlating telemetry through Context Propagation.
- Standardized Instrumentation techniques

Telemetry



Distributed Traces

Tracks & provides the context & flow of tasks as they move through different system components.



Metrics

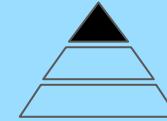
Quantitative measures of a system's internal state that indicates performance and behavior.



Logs

Time-stamped records of events generated by a system.

Standardized Communication

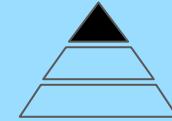


Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

Standardized Communication



Patterns
Support
Stack

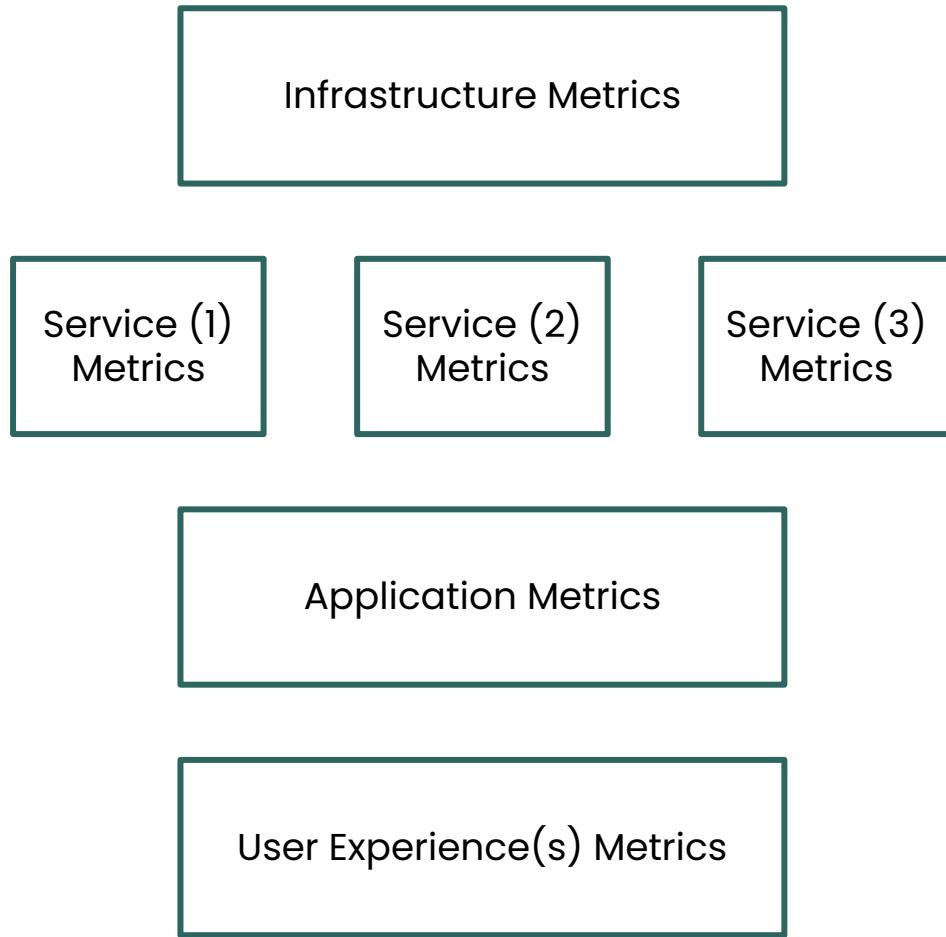
1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

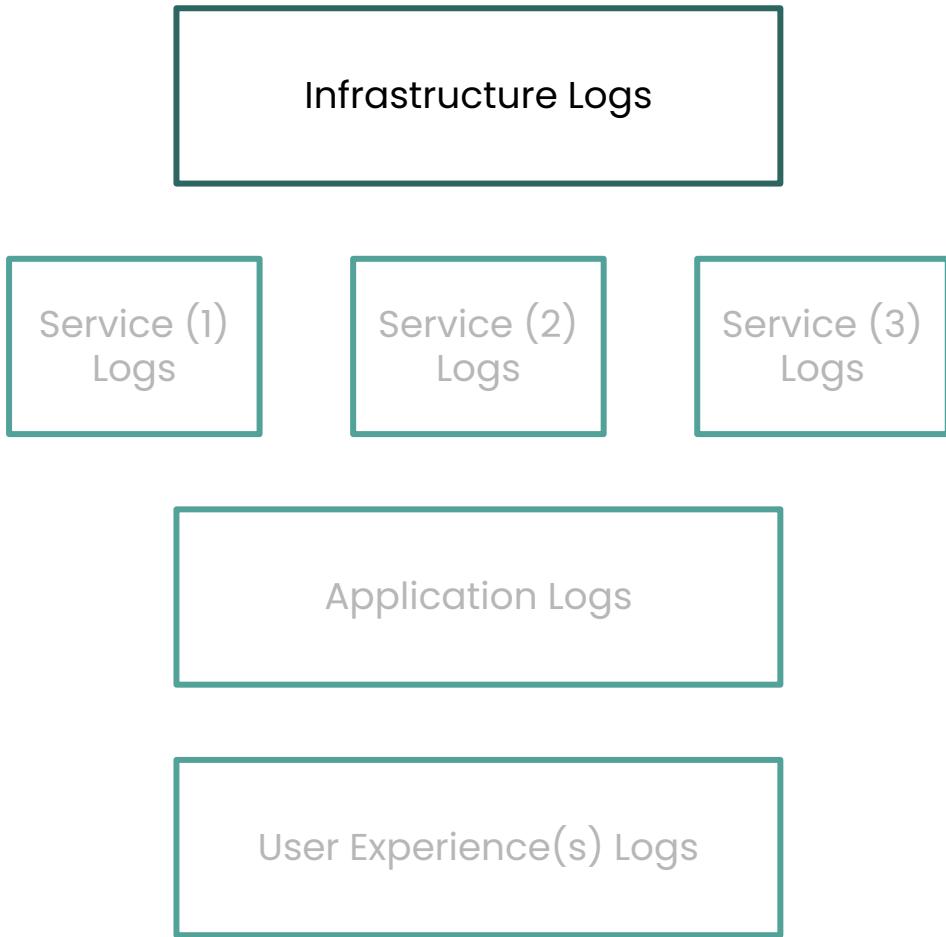
2. Standard context sharing

The set of rules for how context is transferred between different services.

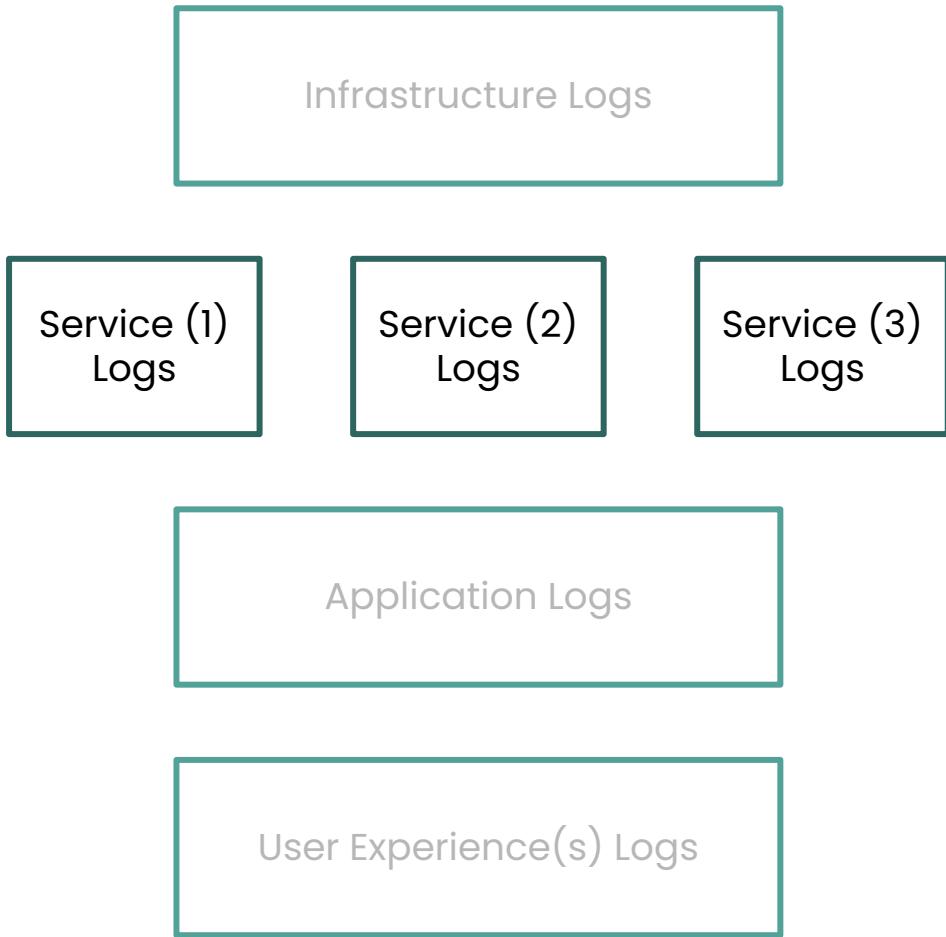
Log Categories



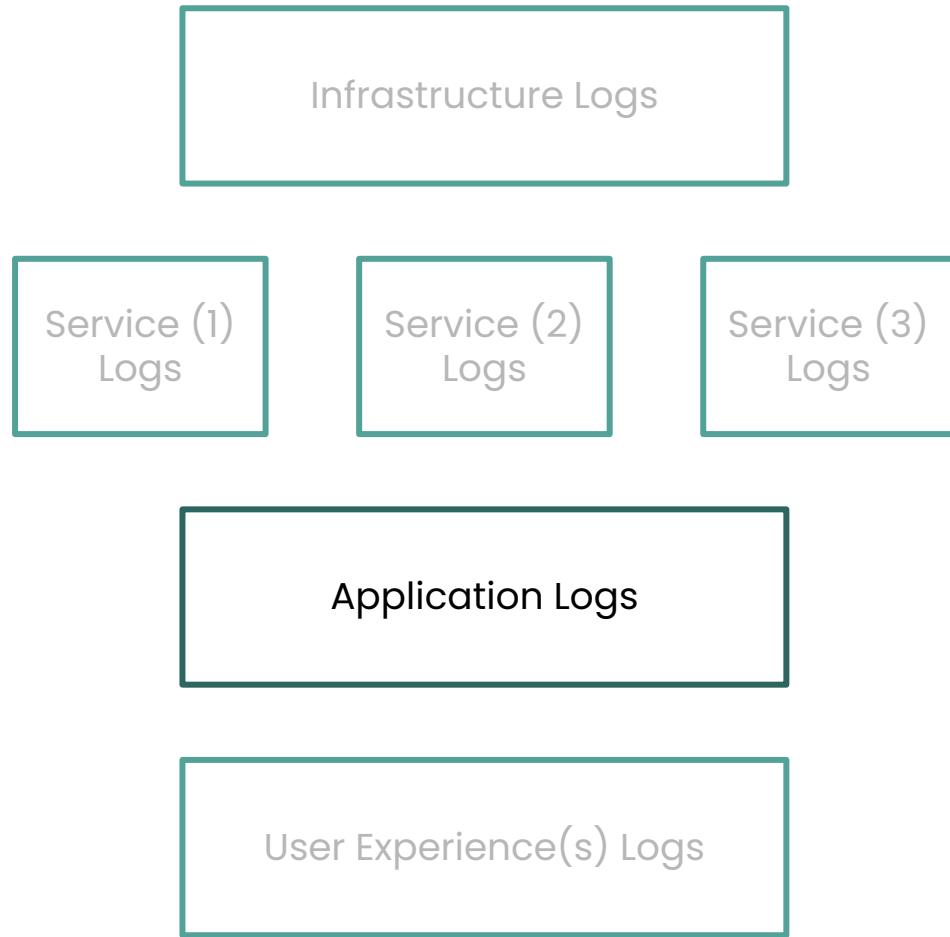
Log Categories: Infra Layer



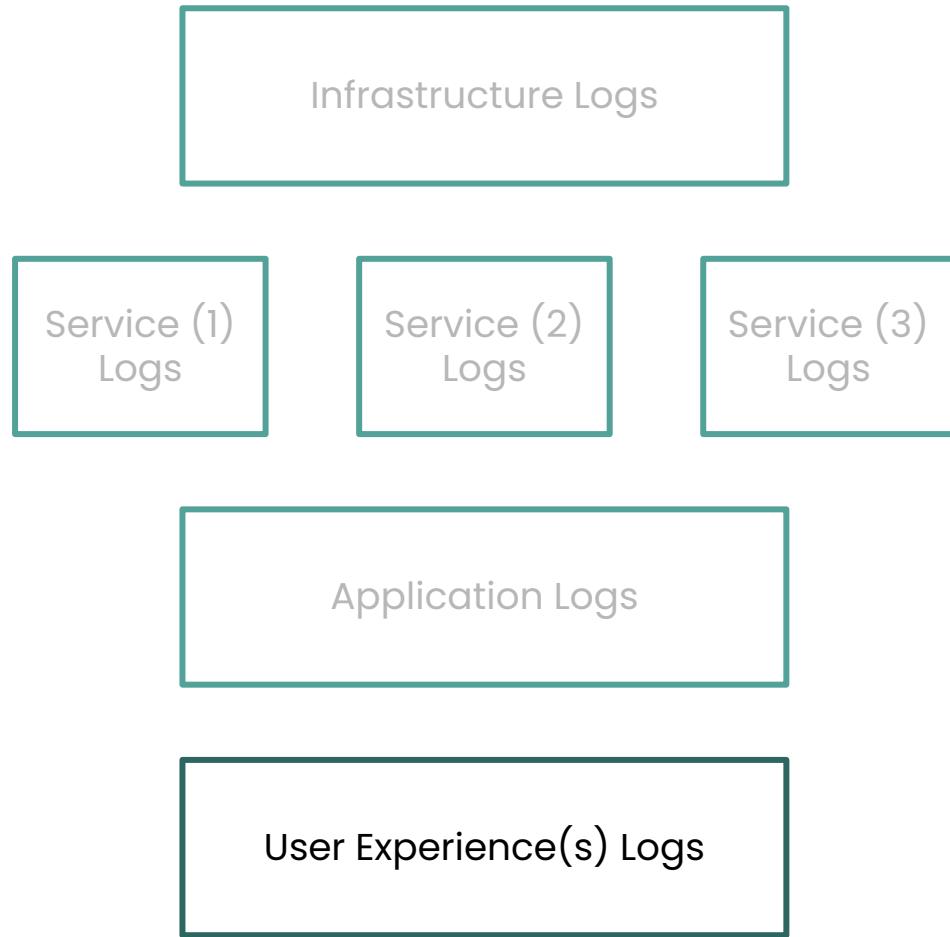
Log Categories: Service Layer



Log Categories: App Layer



Log Categories: User Experience



Example

Hooking logs with traces

```
const { NodeTracerProvider } = require('@opentelemetry/sdk-trace-node');
const { WinstonInstrumentation } = require('@opentelemetry/instrumentation-winston');
const { registerInstrumentations } = require('@opentelemetry/instrumentation');

const provider = new NodeTracerProvider();
provider.register();

registerInstrumentations({
  instrumentations: [
    new WinstonInstrumentation({
      // Optional hook to insert trace context to log metadata.
      logHook: (span, record) => {
        record['resource.service.name'] = provider.resource.attributes['service.name'];
      },
    }),
  ],
});
```

```
const winston = require('winston');
const logger = winston.createLogger({
  transports: [new winston.transports.Console()],
})
logger.info('foobar');
```

Example

Hooking logs with traces

```
const { NodeTracerProvider } = require('@opentelemetry/sdk-trace-node');
const { WinstonInstrumentation } = require('@opentelemetry/instrumentation-winston');
const { registerInstrumentations } = require('@opentelemetry/instrumentation');

const provider = new NodeTracerProvider();
provider.register();

registerInstrumentations({
  instrumentations: [
    new WinstonInstrumentation({
      // Optional hook to insert trace context to log metadata.
      logHook: (span, record) => {
        record['resource.service.name'] = provider.resource.attributes['service.name'];
      },
    }),
  ],
});
```

```
const winston = require('winston');
const logger = winston.createLogger({
  transports: [new winston.transports.Console()],
})
logger.info('foobar');
```

Example

Hooking logs with traces

```
const { NodeTracerProvider } = require('@opentelemetry/sdk-trace-node');
const { WinstonInstrumentation } = require('@opentelemetry/instrumentation-winston');
const { registerInstrumentations } = require('@opentelemetry/instrumentation');

const provider = new NodeTracerProvider();
provider.register();

registerInstrumentations({
  instrumentations: [
    new WinstonInstrumentation({
      // Optional hook to insert trace context to log metadata.
      logHook: (span, record) => {
        record['resource.service.name'] = provider.resource.attributes['service.name'];
      },
    }),
  ],
});
```

```
const winston = require('winston');
const logger = winston.createLogger({
  transports: [new winston.transports.Console()],
})
logger.info('foobar');
```

Example

Hooking logs with traces

```
const { NodeTracerProvider } = require('@opentelemetry/sdk-trace-node');
const { WinstonInstrumentation } = require('@opentelemetry/instrumentation-winston');
const { registerInstrumentations } = require('@opentelemetry/instrumentation');

const provider = new NodeTracerProvider();
provider.register();

registerInstrumentations({
  instrumentations: [
    new WinstonInstrumentation({
      // Optional hook to insert trace context to log metadata.
      logHook: (span, record) => {
        record['resource.service.name'] = provider.resource.attributes['service.name'];
      },
    }),
  ],
});
```

```
const winston = require('winston');
const logger = winston.createLogger({
  transports: [new winston.transports.Console()],
})
logger.info('foobar');
```

Example

Hooking logs with traces

```
const { NodeTracerProvider } = require('@opentelemetry/sdk-trace-node');
const { WinstonInstrumentation } = require('@opentelemetry/instrumentation-winston');
const { registerInstrumentations } = require('@opentelemetry/instrumentation');

const provider = new NodeTracerProvider();
provider.register();

registerInstrumentations({
  instrumentations: [
    new WinstonInstrumentation({
      // Optional hook to insert trace context to log metadata.
      logHook: (span, record) => {
        record['resource.service.name'] = provider.resource.attributes['service.name'];
      },
    }),
  ],
});
```

```
const winston = require('winston');
const logger = winston.createLogger({
  transports: [new winston.transports.Console()],
})
logger.info('foobar');
```

Example

Hooking logs with traces

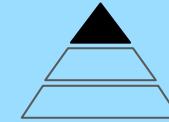
```
const { NodeTracerProvider } = require('@opentelemetry/sdk-trace-node');
const { WinstonInstrumentation } = require('@opentelemetry/instrumentation-winston');
const { registerInstrumentations } = require('@opentelemetry/instrumentation');

const provider = new NodeTracerProvider();
provider.register();

registerInstrumentations({
  instrumentations: [
    new WinstonInstrumentation({
      // Optional hook to insert trace context to log metadata.
      logHook: (span, record) => {
        record['resource.service.name'] = provider.resource.attributes['service.name'];
      },
    }),
  ],
});
```

```
const winston = require('winston');
const logger = winston.createLogger({
  transports: [new winston.transports.Console()],
})
logger.info('foobar');
```

Standardized Communication



Patterns
Support
Stack

1. Shared data specifications

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Standard context sharing

The set of rules for how context is transferred between different services.

3. Standard instrumentation techniques

Provide reliable & rich telemetry data by standardizing the ways we instrument.

4. Well-defined interfaces

The interfaces for tools that enable collecting, processing, or analyzing telemetry data.

Best Practices

1. Use structured data

Logs emitted in unstructured formats should be transformed to structured formats like json, xml, etc.

Best Practices

1. Use structured data

Logs emitted in unstructured formats should be transformed to structured formats like json, xml, etc.

2. Connection & correlation data

Trace context, span context, and timestamps should be included to enable connecting and correlating logs with other telemetry signals.

Best Practices

1. Use structured data

Logs emitted in unstructured formats should be transformed to structured formats like json, xml, etc.

2. Connection & correlation data

Trace context, span context, and timestamps should be included to enable connecting and correlating logs with other telemetry signals.

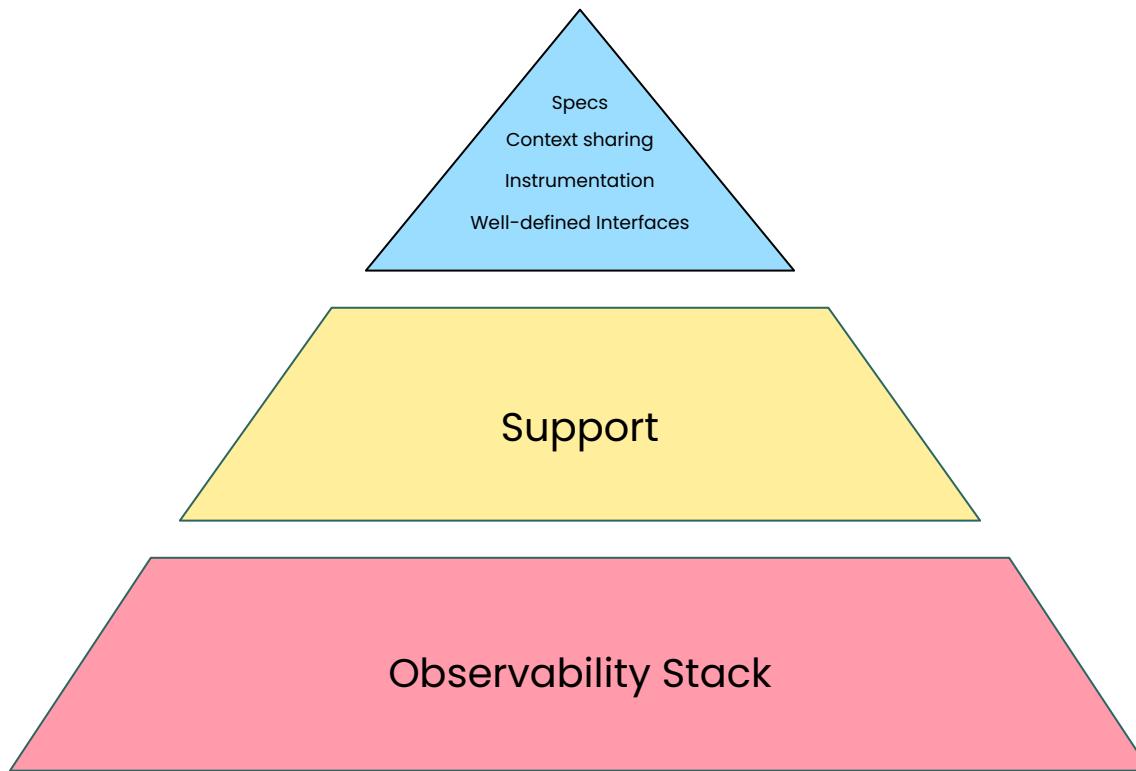
3. Sampling techniques

To reduce noise & cut costs, use strategic sampling techniques.



Q&A

Standardized observability platforms



Defining Observability

“the ability to understand what’s happening inside of your software systems to debug problems you’ve never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It’s also not just about a specific tool, it’s about a team’s ability to analyze that telemetry data.”

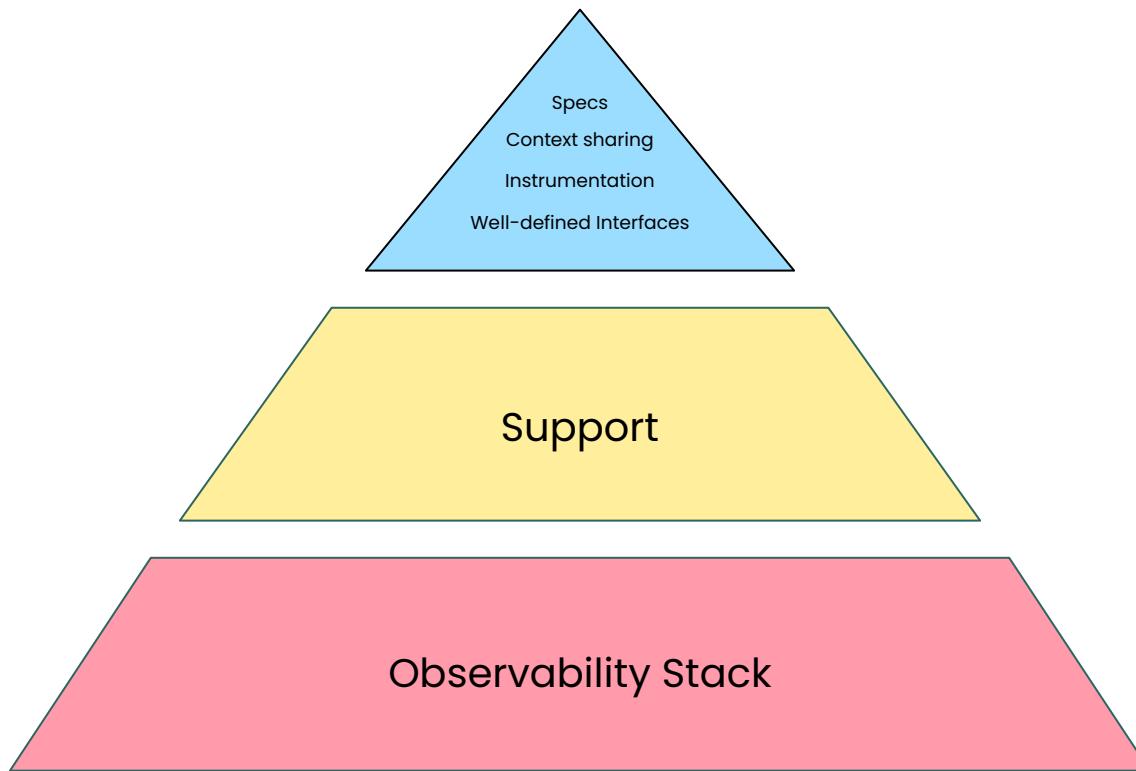
- Liz Fong-Jones, Honeycomb

Defining Observability

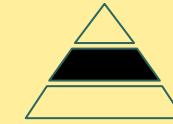
"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Standardized observability platforms

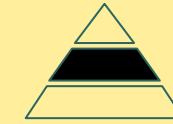


Platform Support Strategies



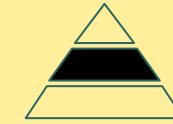
Patterns
Support
Stack

Platform Support Strategies



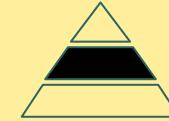
Patterns
Support
Stack

Platform Support Strategies



Patterns
Support
Stack

Platform Support Strategies

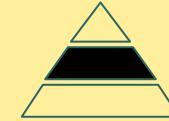


Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system
observable.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system
observable.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system observable.

2. Support engineers in refining their skills

Support engineers in their ability to analyze telemetry data effectively.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system observable.

2. Support engineers in refining their skills

Support engineers in their ability to analyze telemetry data effectively.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system observable.

2. Support engineers in refining their skills

Support engineers in their ability to analyze telemetry data effectively.

3. Embed observability into your the culture

Reinforce the idea that observability is a crucial part of managing services.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system observable.

2. Support engineers in refining their skills

Support engineers in their ability to analyze telemetry data effectively.

3. Embed observability into your the culture

Reinforce the idea that observability is a crucial part of managing services.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system observable.

2. Support engineers in refining their skills

Support engineers in their ability to analyze telemetry data effectively.

3. Embed observability into your the culture

Reinforce the idea that observability is a crucial part of managing services.

Platform Support Strategies



Patterns
Support
Stack

1. Make building observable services easy

Reduce the work needed to make a system observable.

2. Support engineers in refining their skills

Support engineers in their ability to analyze telemetry data effectively.

3. Embed observability into your culture

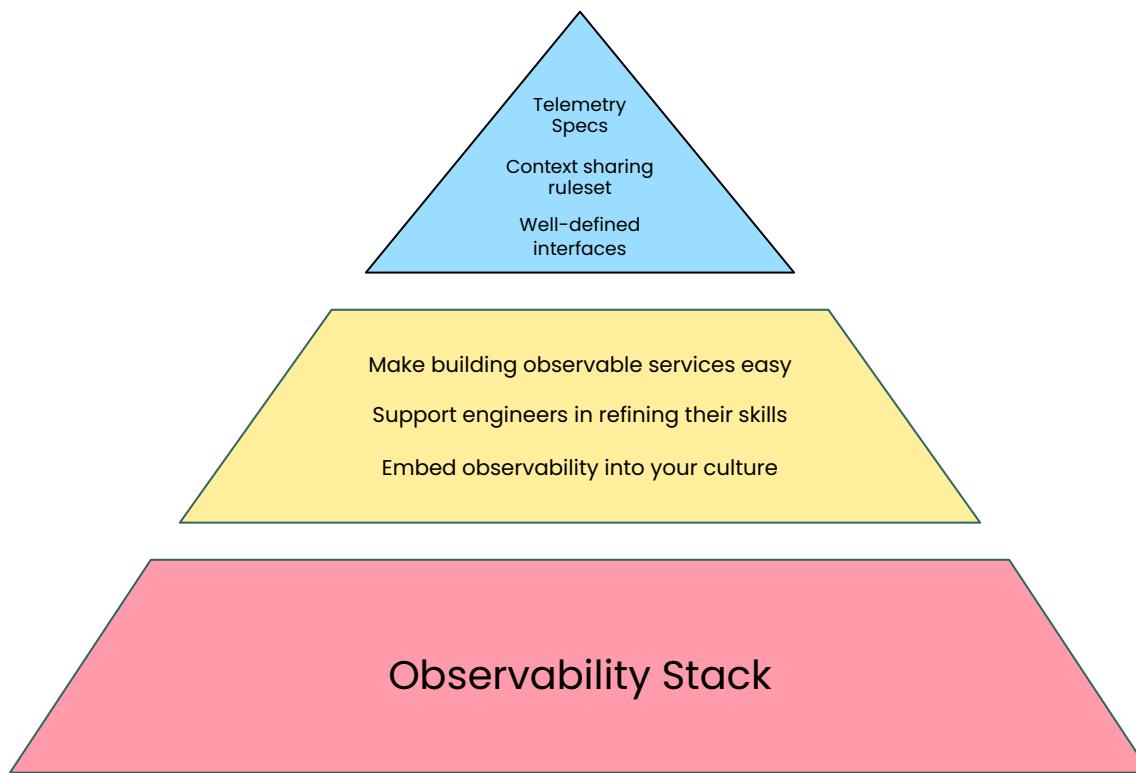
Reinforce the idea that observability is a crucial part of managing services.



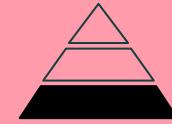


Q&A

Standardized observability platforms



Observability Stack



Patterns
Support
Stack

Architecture-to-Engineer Communication



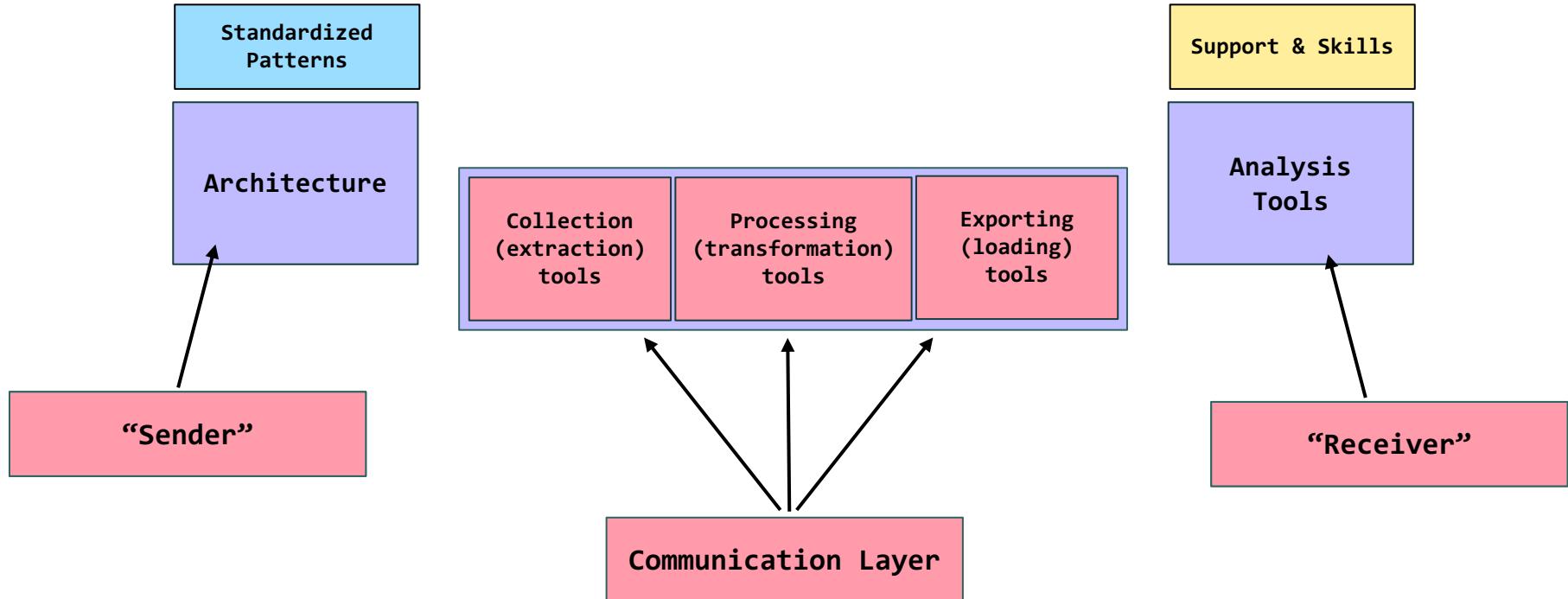
Patterns
Support
Stack



Architecture-to-Engineer Communication



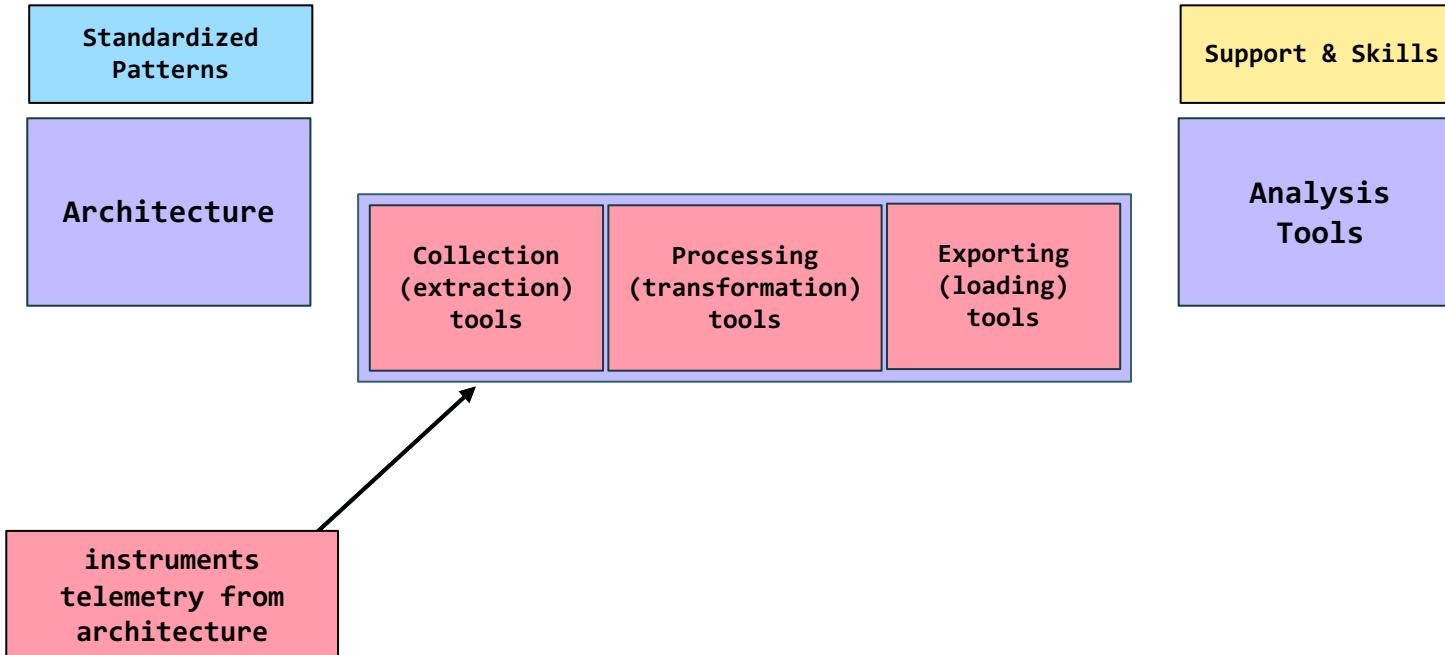
Patterns
Support
Stack



Architecture-to-Engineer Communication



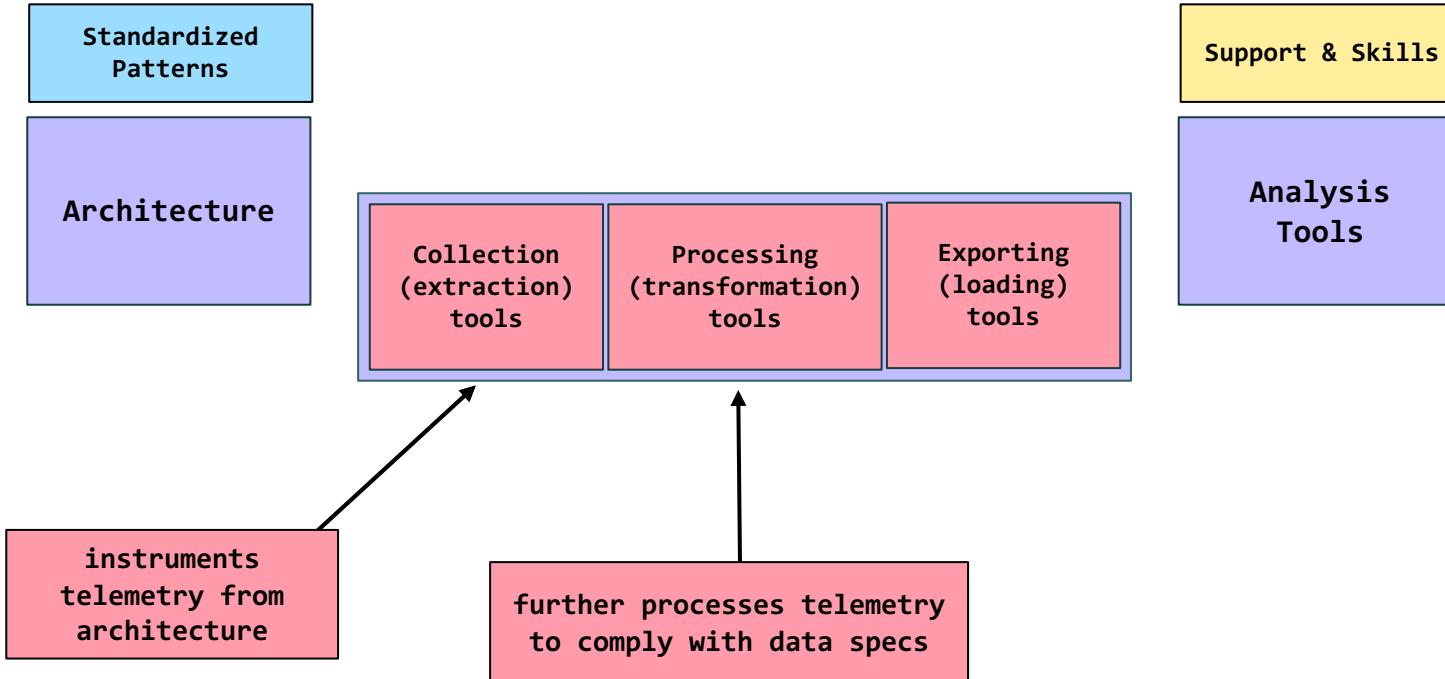
Patterns
Support
Stack



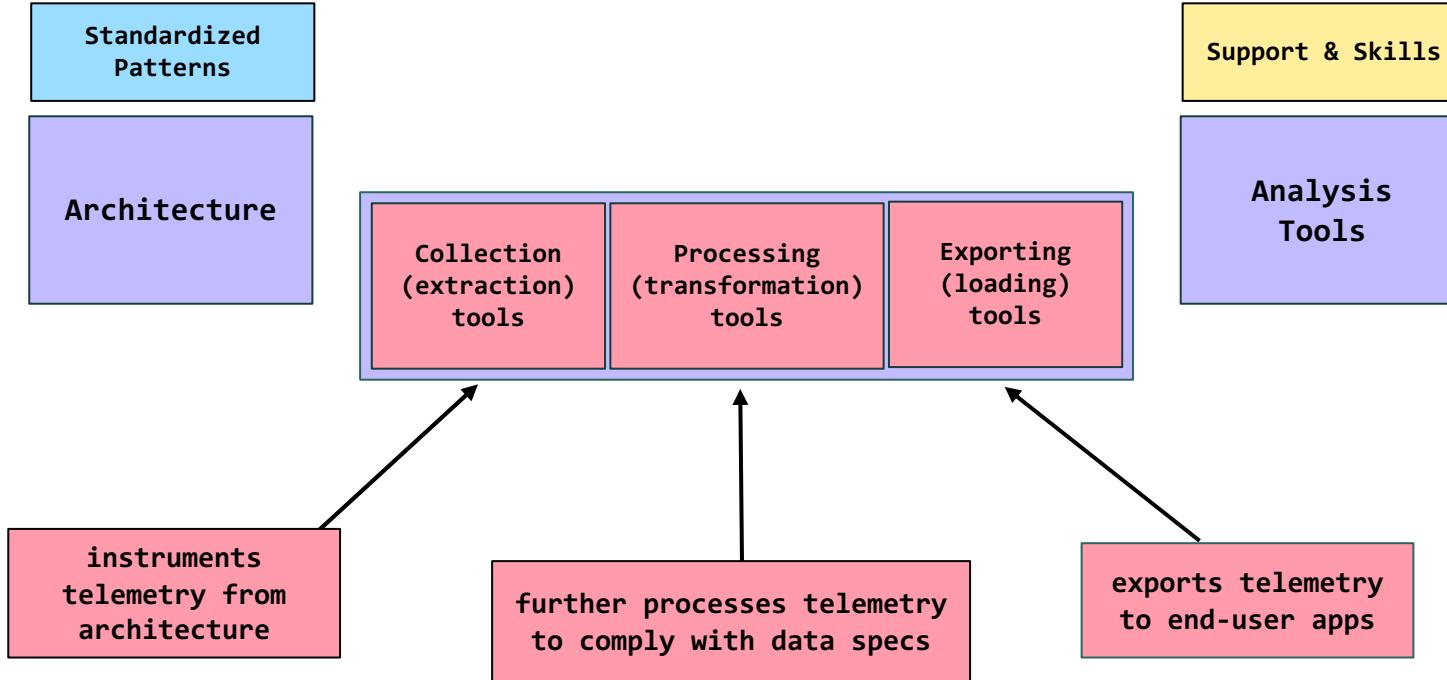
Architecture-to-Engineer Communication



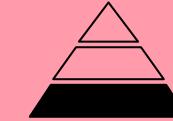
Patterns
Support
Stack



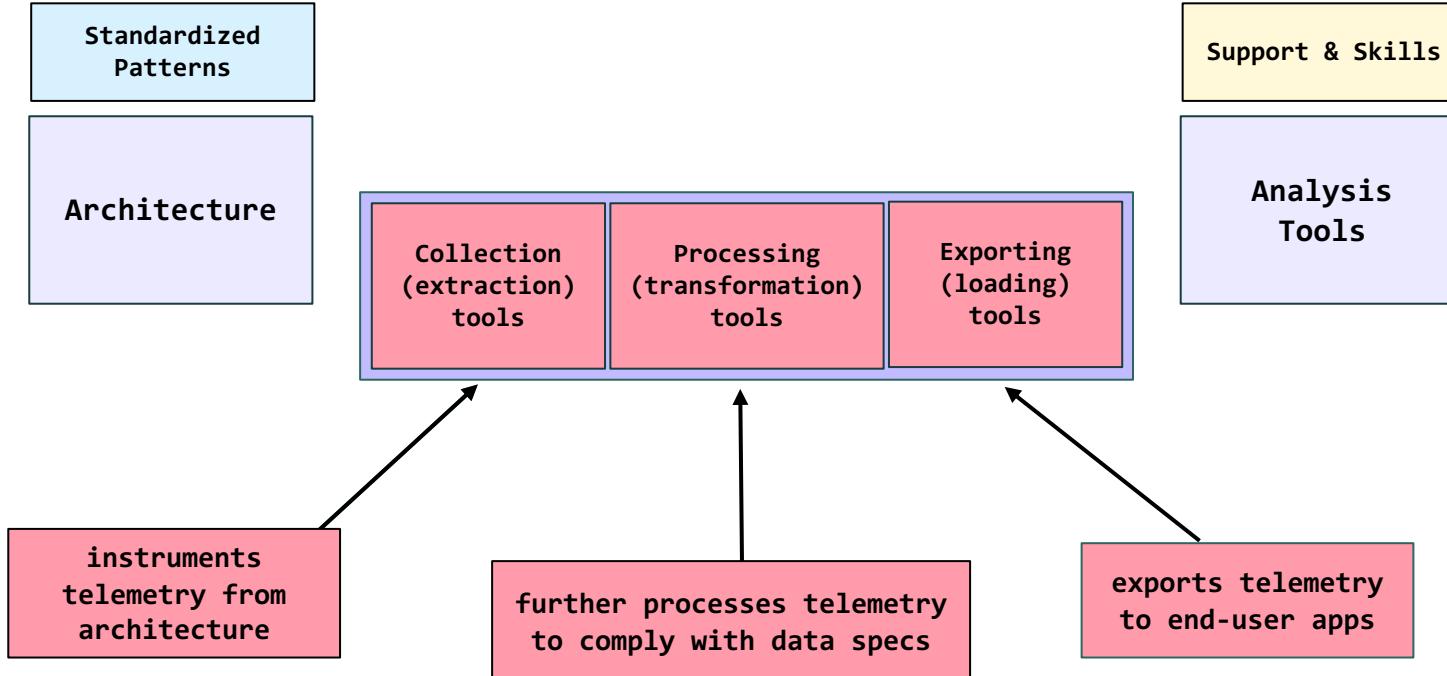
Architecture-to-Engineer Communication



Architecture-to-Engineer Communication



Principles
Support
Architecture



Communication Layer Design Principles



Principles
Support
Architecture

Reliability

Engineers should be able to trust the telemetry emitted from our systems.

Communication Layer Design Principles



Reliability

Engineers should be able to trust the telemetry emitted from our systems.

Richness

Embrace high-dimensional & high-cardinality telemetry data.

Communication Layer Best Practices



What's our shared definition of telemetry?

Communication Layer Best Practices



1. Event-driven telemetry

Provides a standardized yet flexible way of collecting telemetry data.

Communication Layer Best Practices



1. Event-driven telemetry

Provides a standardized yet flexible way of collecting telemetry data.

Communication Layer Best Practices



1. Event-driven telemetry

Provides a standardized yet flexible way of collecting telemetry data.

2. Standardize instrumentation techniques

Provide reliable & rich telemetry data by standardizing the ways we instrument.

Communication Layer Best Practices



1. Event-driven telemetry

Provides a standardized yet flexible way of collecting telemetry data.

2. Standardize instrumentation techniques

Provide reliable & rich telemetry data by standardizing the ways we instrument.

3. Use auto-instrumentation first

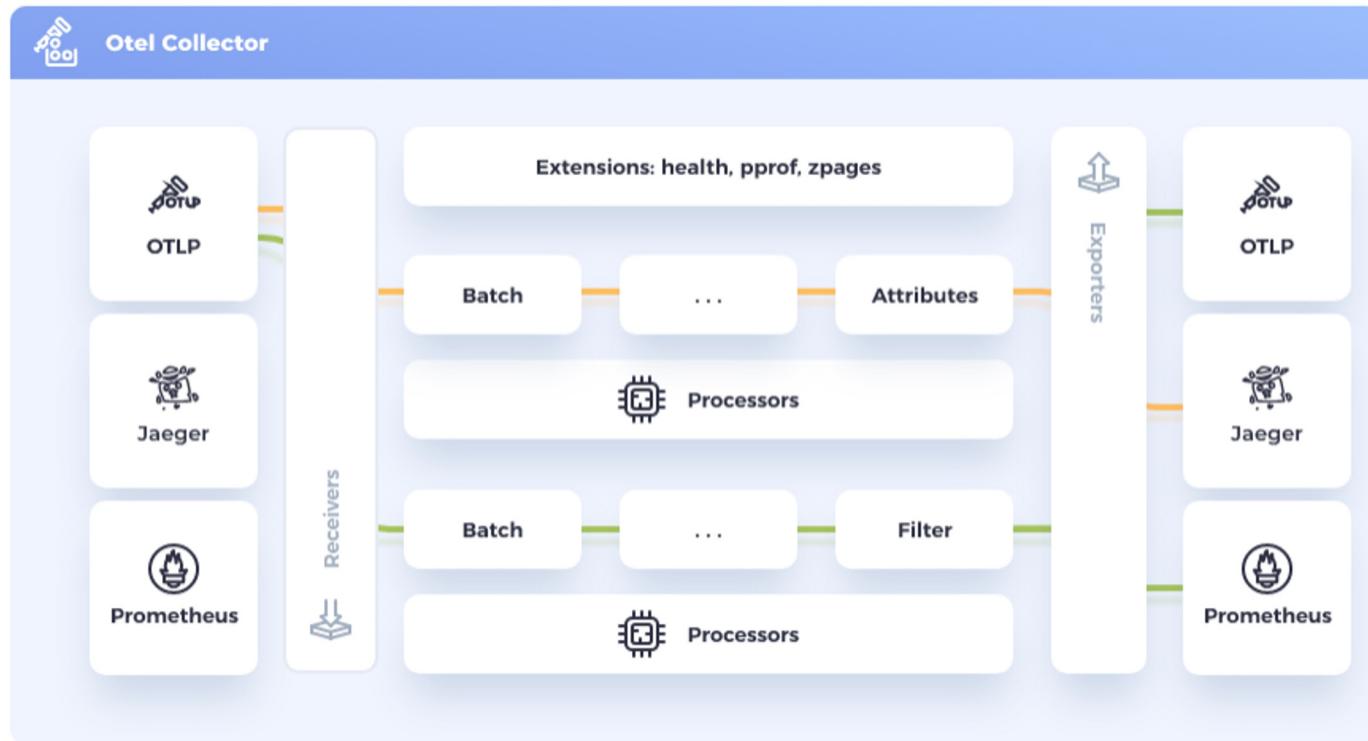
Provides consistent techniques for instrumenting telemetry.

4. Use manual instrumentation strategically

Provides an avenue for addressing inconsistencies with platform standards.



OpenTelemetry Collector





Receivers

Accepts telemetry from
instrumented applications
and sends it to the
observability pipeline.



Receivers

Accepts telemetry from instrumented applications and sends it to the observability pipeline.



Processors

Modifies and enriches telemetry as it flows through the pipeline.



Receivers

Accepts telemetry from instrumented applications and sends it to the observability pipeline.



Processors

Modifies and enriches telemetry as it flows through the pipeline.



Exporters

Sends telemetry from the pipeline to observability backends.

Tool Decisions: Collector



1. Manage your own Collector Service

Maintain your own OpenTelemetry collector by deploying and managing it within your infrastructure.

Tool Decisions: Collector



1. Manage your own Collector Service

Maintain your own OpenTelemetry collector by deploying and managing it within your infrastructure.

2. Vendor Hosted Collector

Use a vendor managed collector solution.

Tool Decisions: Collector



1. Manage your own Collector Service

Maintain your own OpenTelemetry collector by deploying and managing it within your infrastructure.

2. Vendor Hosted Collector

Use a vendor managed collector solution.

3. Cloud Provider Integration

Utilize integrations provided by your Cloud Provider instead of your observability vendor tools.



Demo

OpenTelemetry Collector Example

<https://bit.ly/3MIWVYf>

or

<https://github.com/open-telemetry/opentelemetry-demo>

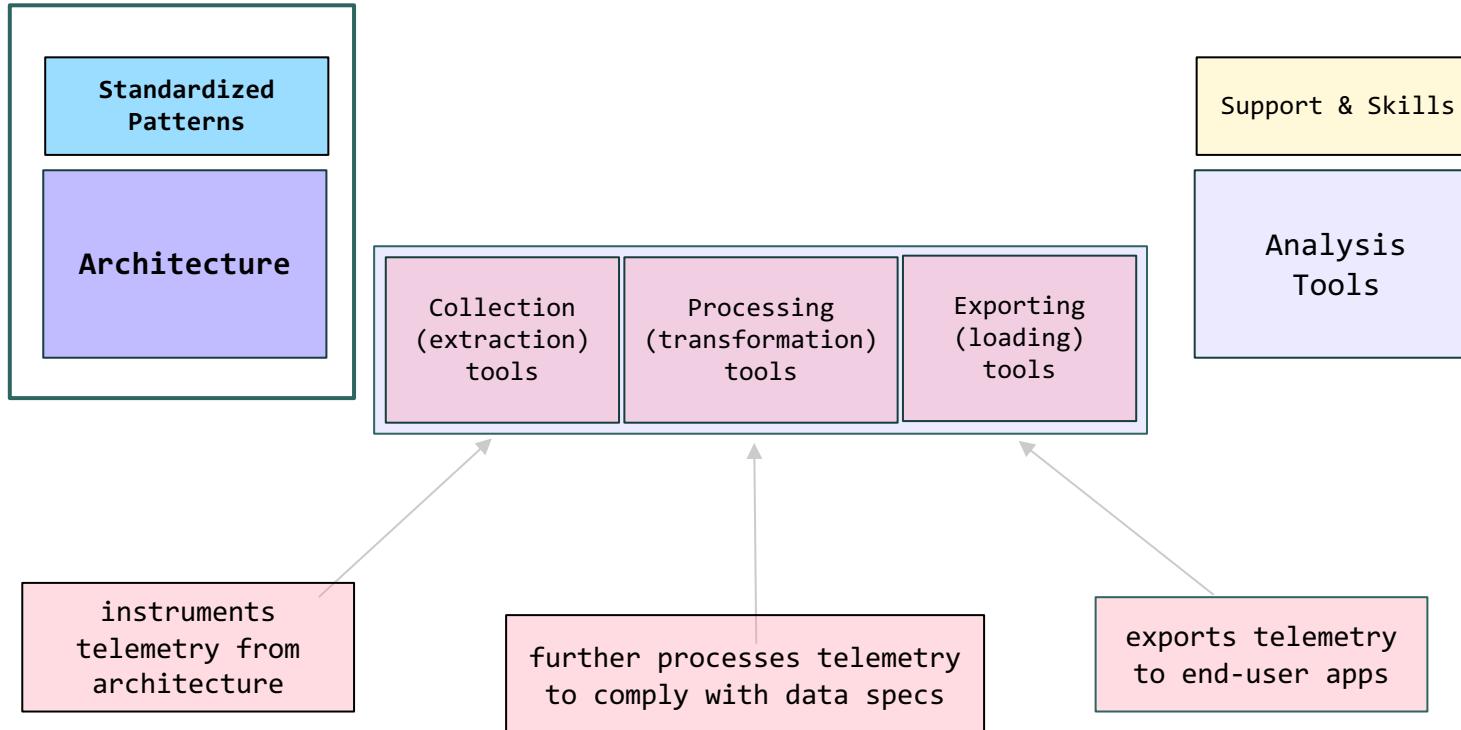


Q&A

Architecture-to-Engineer Communication



Patterns
Support
Stack





Lightstep

from ServiceNow



splunk®



Stackdriver



dynatrace



APPDYNAMICS



AWS CloudWatch



INSTANA



Cribl®



Poll

“Which observability tools are you currently using at work?”

1. *Honeycomb*
2. *New Relic*
3. *DataDog*
4. *Splunk*
5. *CloudWatch or StackDriver*
6. *Other*

Investigation & Analysis



Principles
Support
Architecture

Investigation & Analysis



Principles
Support
Architecture

Investigation & Analysis



1. What observability gap does it fill?

Engineers should have access to functionality that enables them to debug production issues.

Investigation & Analysis



1. What observability gap does it fill?

Engineers should have access to functionality that enables them to debug production issues.

2. How compatible is it with our platform vision?

Optimize for tools that introduce less complexity to your platform.

Investigation & Analysis



1. What observability gap does it fill?

Engineers should have access to functionality that enables them to debug production issues.

2. How compatible is it with our platform vision?

Optimize for tools that introduce less complexity to your platform.

3. How easy is it to use?

Consider the developer experience of your tools.

Investigation & Analysis



1. What observability gap does it fill?

Engineers should have access to functionality that enables them to debug production issues.

2. How compatible is it with our platform vision?

Optimize for tools that introduce less complexity to your platform.

3. How easy is it to use?

Consider the developer experience of your tools.

4. Does it provide our support needs?

Tools without adoption or use don't address organizational challenges.

Investigation & Analysis



1. What observability gap does it fill?

Engineers should have access to functionality that enables them to debug production issues.

2. How compatible is it with our platform vision?

Optimize for tools that introduce less complexity to your platform.

3. How easy is it to use?

Consider the developer experience of your tools.

4. Does it provide our support needs?

Tools without adoption or use don't address organizational challenges.

Investigation & Analysis



1. What observability gap does it fill?

Engineers should have access to functionality that enables them to debug production issues.

2. How compatible is it with our platform vision?

Optimize for tools that introduce less complexity to your platform.

3. How easy is it to use?

Consider the developer experience of your tools.

4. Does it provide our support needs?

Tools without adoption or use don't address organizational challenges.



Exercise

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-oly/
final



Takeaways

- Social challenges of observability – shifting towards a new framework of thinking.
 - Organizational support strategies to address this
- Building Microservice observability platforms with OpenTelemetry.
- Brief System Architecture overview
 - Walkthrough each component & discuss architecture challenges
 - Client side, async, serverless, and full stack challenges.
- Observability Architecture via the OpenTelemetry Collector
 - Overview & implementation of Receivers, Processors, & Exporters.
- Investigation & analysis best practices.



Q&A

O'REILLY®



```
// w3c trace context spec
{
  "traceId": "1234",
  "tracestate": "somestringhere"
  "traceFlags": 1,
}
```



```
// b3 trace context spec
{
  "X-B3-TraceId": "1234",
  "X-B3-Sampled": "1",
  "X-B3-Flags": "1"
}
```

Trace Context

The necessary metadata that enables tracers to manage, correlate, and assemble spans.

Tracers



```
// w3c trace & span context spec
{
  "traceId": "1234",
  "spanId": "4940"
}
```



```
// b3 trace context spec
{
  "X-B3-TraceId": "1234",
  "X-B3-SpanId": "4940",
}
```

Span Context

Additional metadata that enables tracers to manage, correlate, and assemble spans.

Tracers



Discussion

“How would you define an event in the context of a client-side rendered front-end service vs a server-side rendered service?”



```
// w3c trace & span context spec
{
  "traceId": "1234",
  "spanId": "4940"
  "parentSpanId": "8080",
}
```

```
// b3 trace context spec
{
  "X-B3-TraceId": "1234",
  "X-B3-SpanId": "4940",
  "X-B3-parentSpanId": "8080",
}
```

Span Context

Additional metadata that enables tracers to manage, correlate, and assemble spans.



Tracers



Exercise 3

bit.ly/3pZiOF8

or

github.com/clesleycode/oreilly-olly/exercise3



Discussion

“What metadata standards would support building observable API Gateways?”



Exercise 4

bit.ly/3pZiOF8

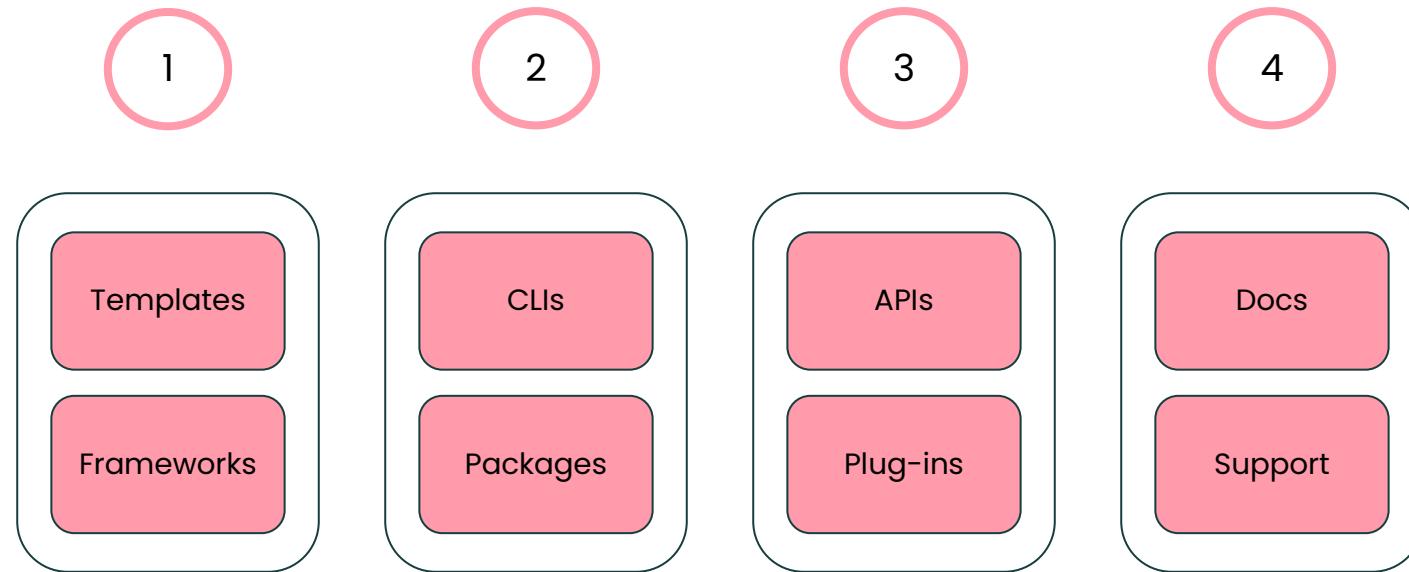
or

github.com/clesleycode/oreilly-olly/exercise4

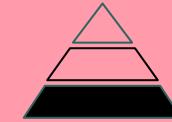
Platform “Recipe”



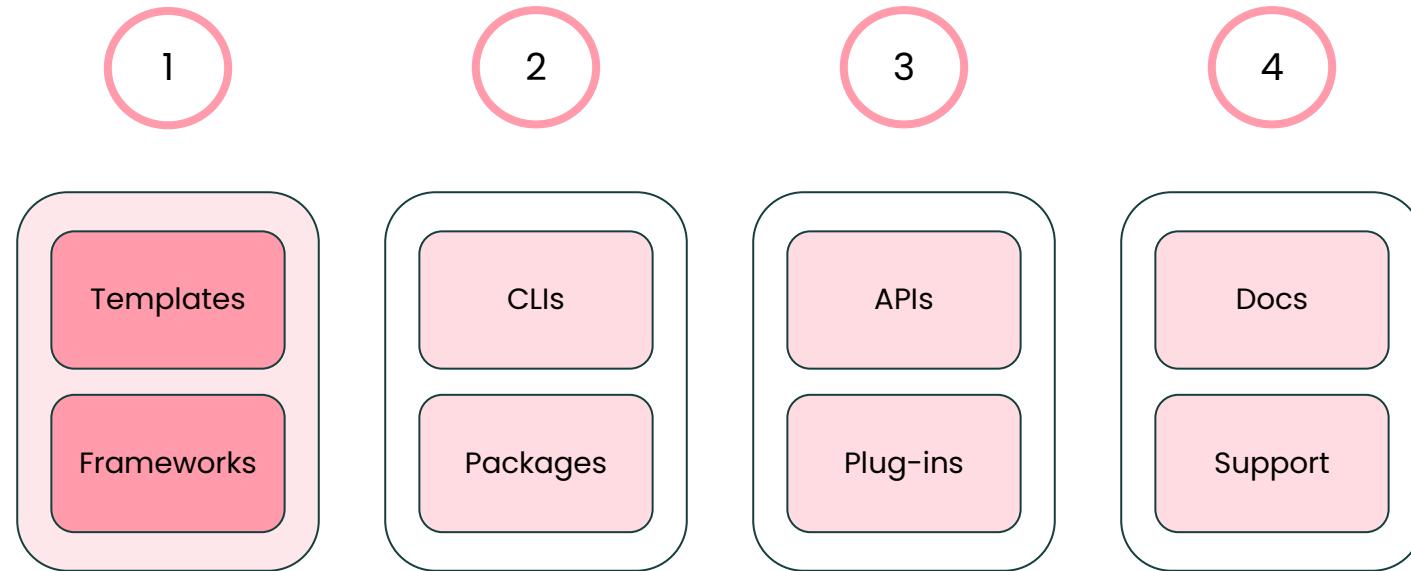
Practices
Support
Architecture



Platform “Recipe”



Practices
Support
Architecture

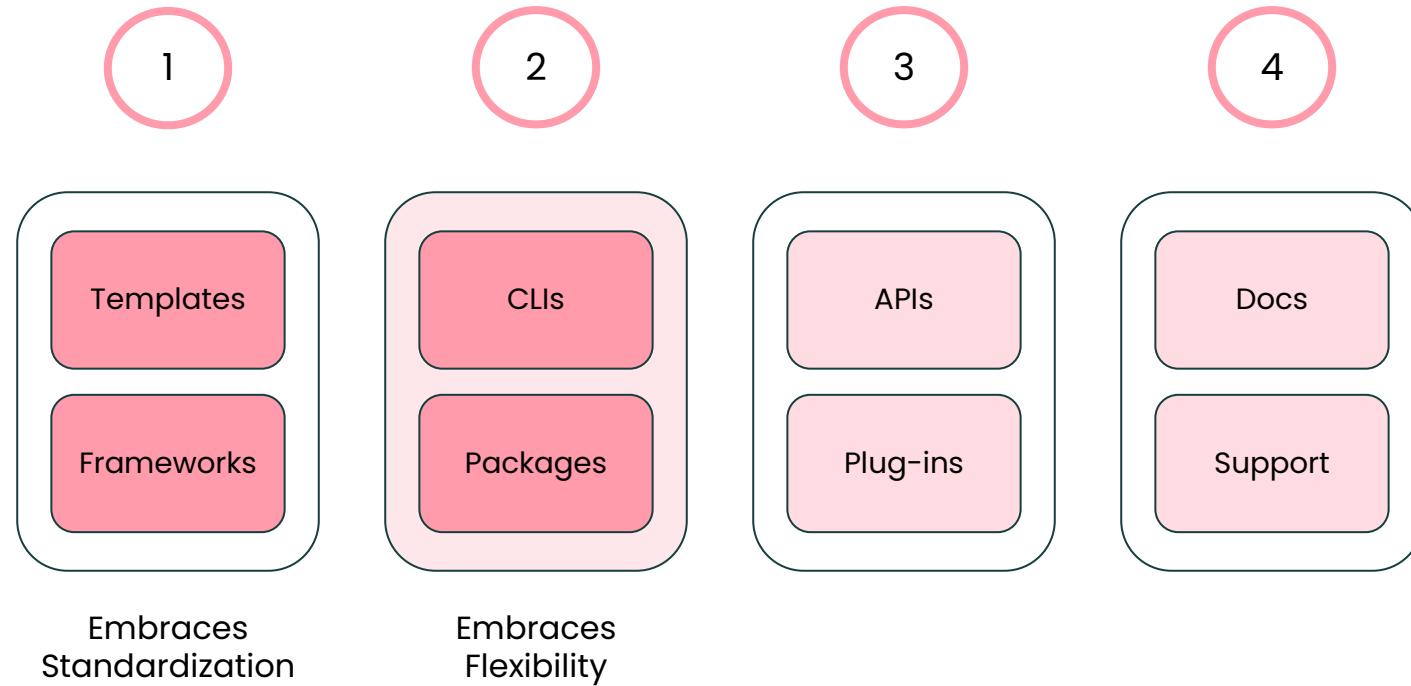


Embraces
Standardization

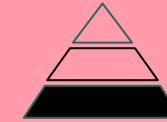
Platform “Recipe”



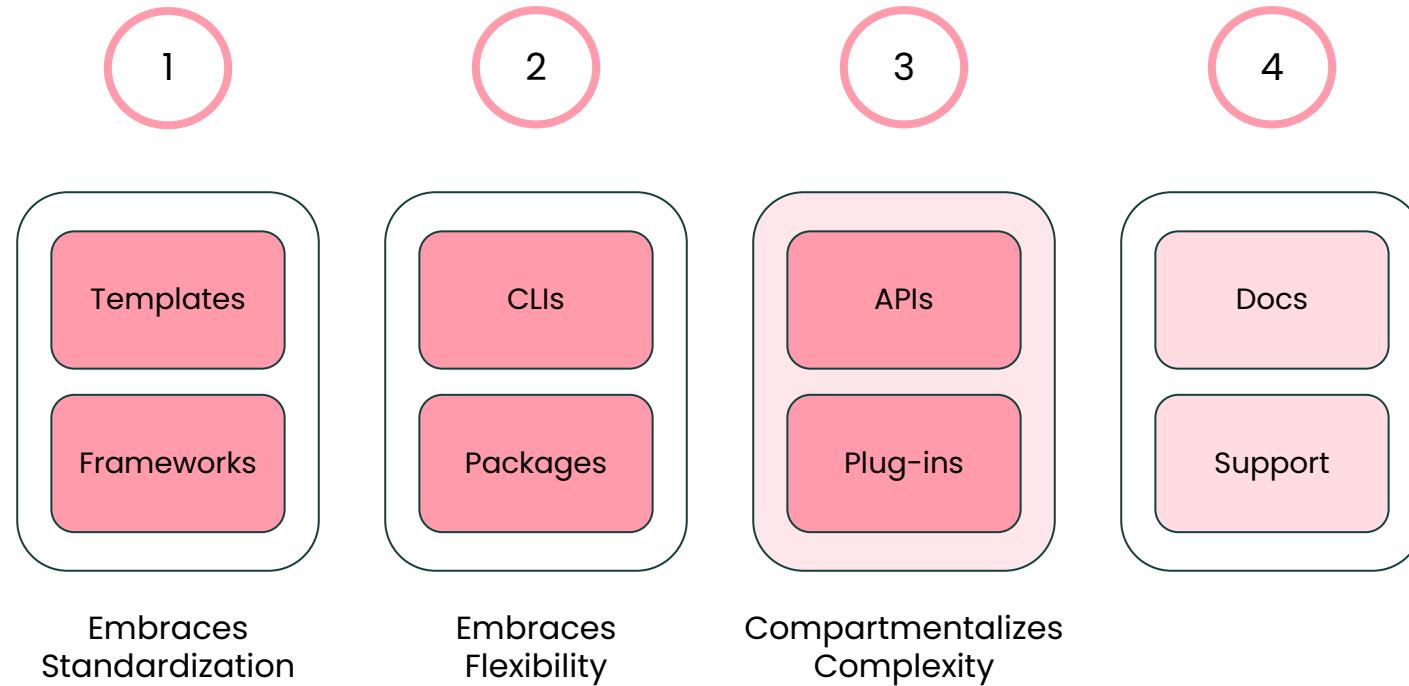
Practices
Support
Architecture



Platform “Recipe”



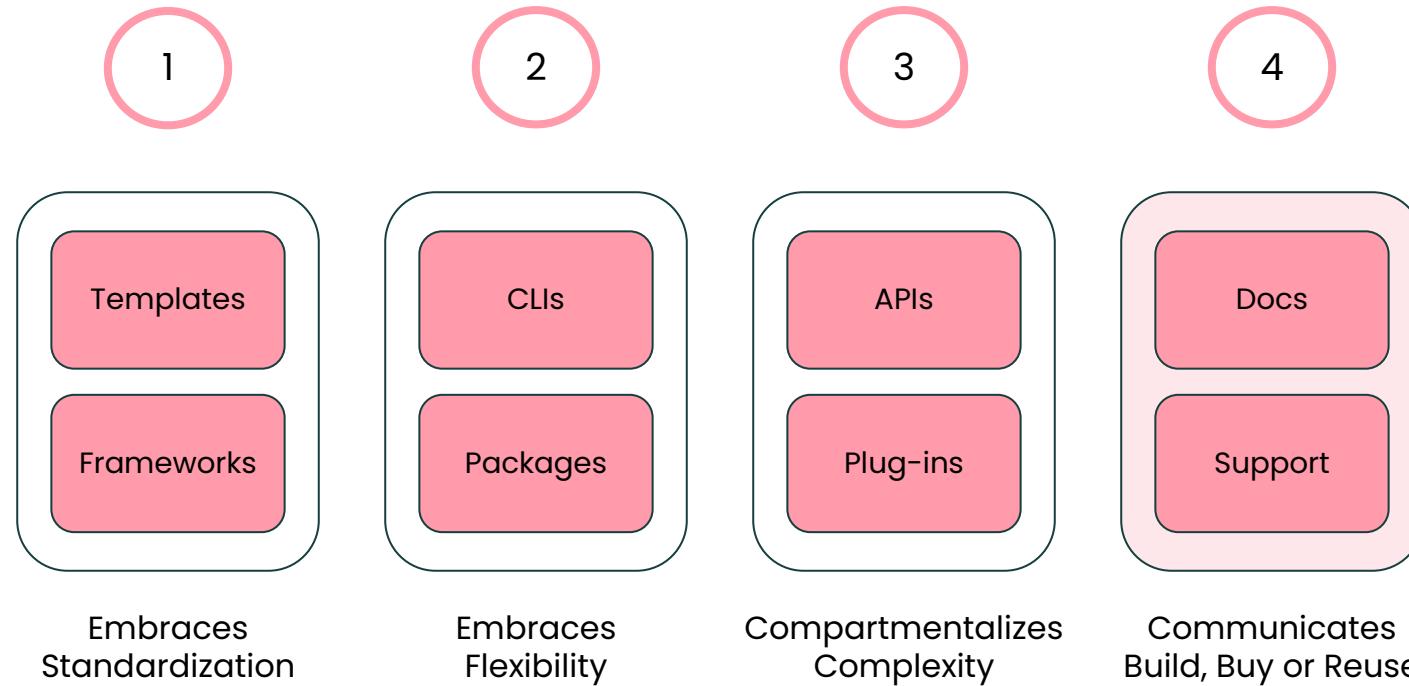
Practices
Support
Architecture



Platform “Recipe”



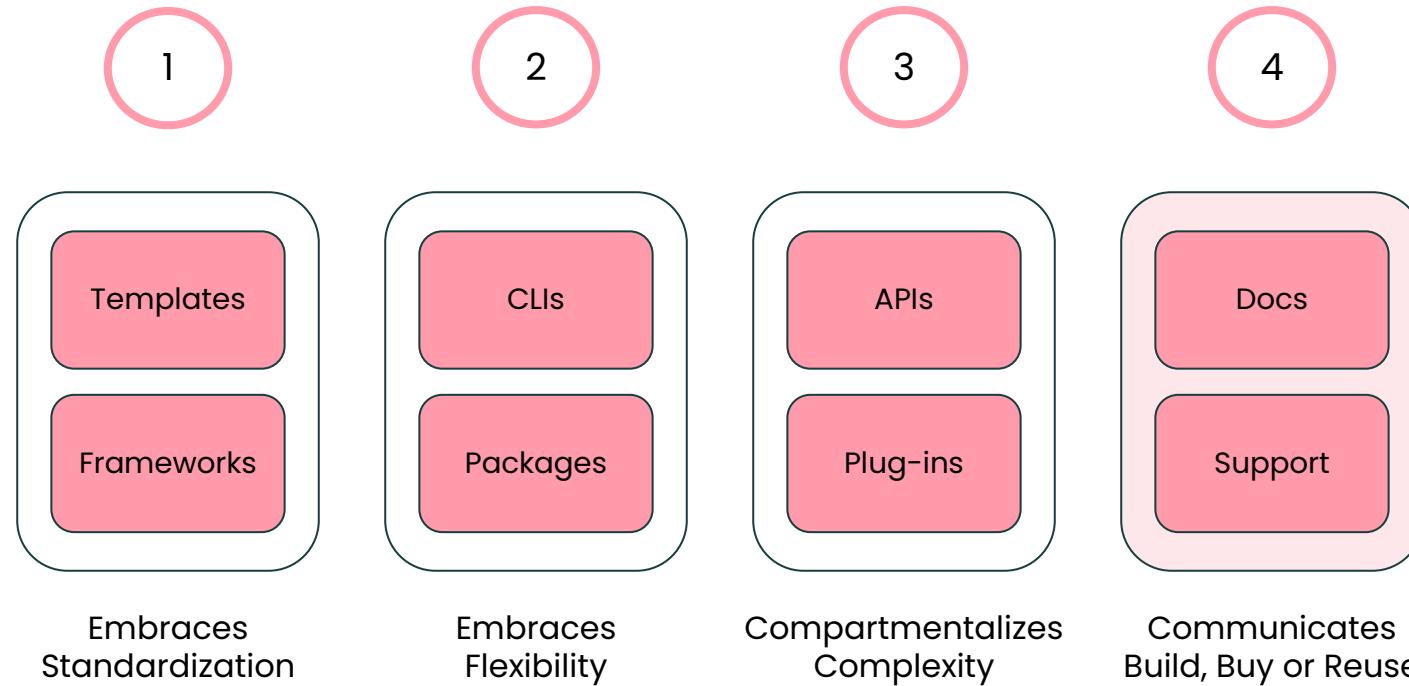
Practices
Support
Architecture



Platform “Recipe”



Practices
Support
Architecture

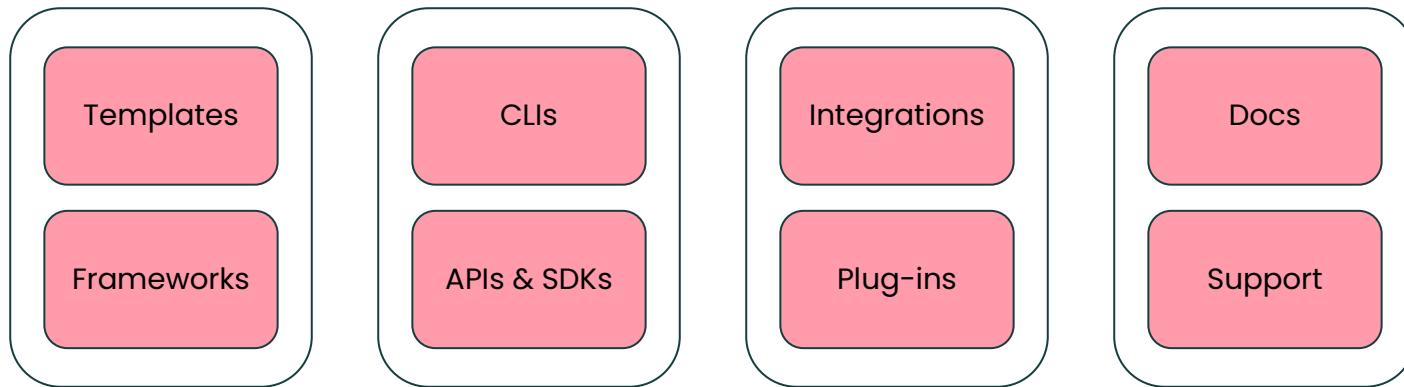


Platform Visibility



Practices
Support
Architecture

Developer Portal

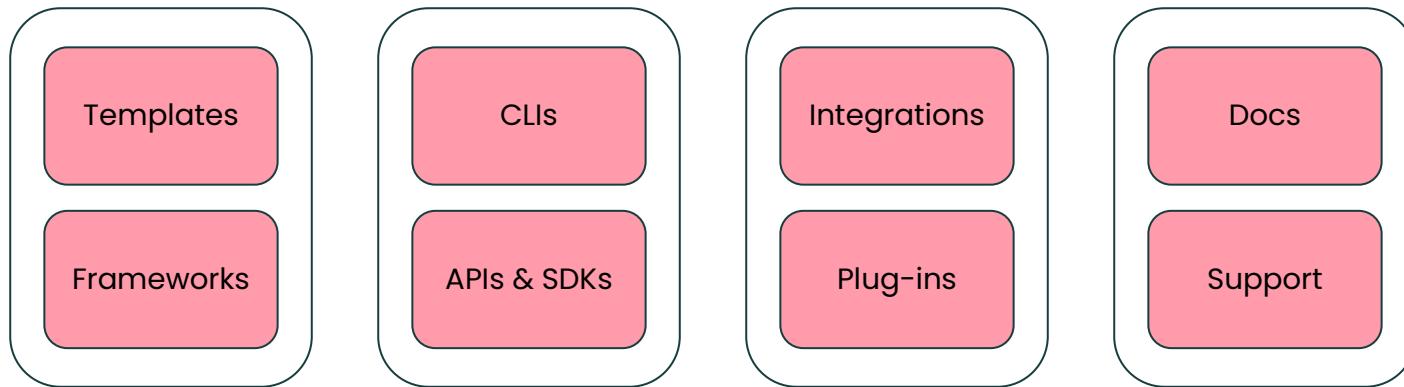


Platform Visibility



Practices
Support
Architecture

Developer Portal

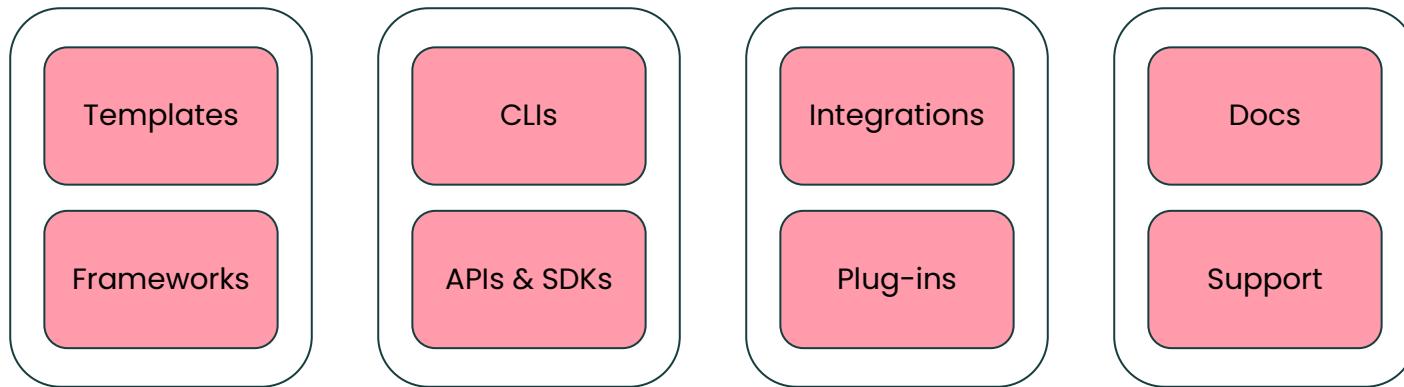


Platform Visibility



Practices
Support
Architecture

Developer Portal



Observability Architecture Challenges



1. Client-side rendering

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Event-driven architectures

The set of rules for how context is transferred between different services.

Observability Architecture Challenges



1. Client-side rendering

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Event-driven architectures

The set of rules for how context is transferred between different services.

3. Serverless

Provide reliable & rich telemetry data by standardizing the ways we instrument.

Observability Architecture Challenges



1. Client-side rendering

Defines telemetry data definitions that drive consistency in how telemetry is collected.

2. Event-driven architectures

The set of rules for how context is transferred between different services.

3. Serverless

Provide reliable & rich telemetry data by standardizing the ways we instrument.

4. Full stack observability

The interfaces for tools that enable collecting, processing, or analyzing telemetry data.



Discussion

“Are there any topics from today that you’d like to dive deeper into tomorrow?”

For example, metrics best practices, more logging types, context propagation, manual instrumentation, agents, etc.

Feature Development Cycle



Alert

Partnering with product management to define feature requirements.

Monitor

Designing the software that will enable your release.

Telemetry

The actual implementation of your feature.

Prod Readiness

Preparing for the challenges that might come from production.

Delivery

Finally, release! 🎉

Maintenance

The ongoing cost of maintaining the feature in production.

Actionable Observability



Lesley Cordero, @clesleycode
Staff Engineer & Tech Lead, DV Observability

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Defining Observability

“the ability to understand what’s happening inside of your software systems to debug problems you’ve never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It’s also not just about a specific tool, it’s about a team’s ability to analyze that telemetry data.”

- Liz Fong-Jones, Honeycomb

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

Actionable Observability

The practice of **utilizing telemetry** to identify, debug, and address technical issues with the goal of improving application reliability.

Actionable Observability

The practice of utilizing telemetry to **identify, debug, and address technical issues** with the goal of improving application reliability.

Actionable Observability: Incident Management Cycle

0

(Prerequisite)
Collect



1

Identify



2

Debug



3

Address



Actionable Observability

The practice of utilizing telemetry to identify, debug, and address technical issues with the goal of improving application reliability.

Defining Observability

"the ability to understand what's happening inside of your software systems to debug problems you've never seen before just using the telemetry (traces, logs, & metrics) emitted by your applications. It's also not just about a specific tool, it's about a team's ability to analyze that telemetry data."

- Liz Fong-Jones, Honeycomb

The role of Automation

The role of Automation

Consolidate efforts

Automation reduces drift
by centralizing efforts into
software that performs this
work for you.

The role of Automation

Consolidate efforts

Automation reduces drift by centralizing efforts into software that performs this work for you.

Reduce cognitive load

Reducing our cognitive load enables us to focus on skills that can't be automated away.

The role of Automation

Consolidate efforts

Automation reduces drift by centralizing efforts into software that performs this work for you.

Reduce cognitive load

Reducing our cognitive load enables us to focus on skills that can't be automated away.

Make better decisions

By reducing the work and cognitive load needed to manage an incident, we'll be able to make more intentional decisions.

Actionable Observability: Incident Management Cycle

0

Collect



1

Identify



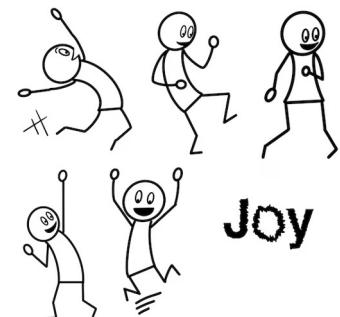
2

Debug



3

Address



Actionable Observability: Incident Management Cycle

0

Collect



1

Identify



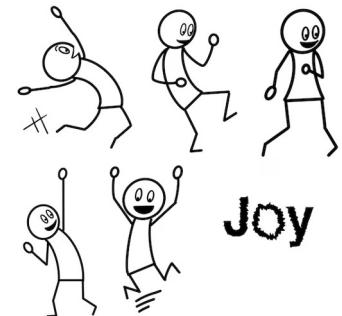
2

Debug



3

Address



Monitoring

Automating the discovery of application issues to aid incident management in scalable ways.

Actionable Observability: Incident Management Cycle

0

Collect



1

Identify



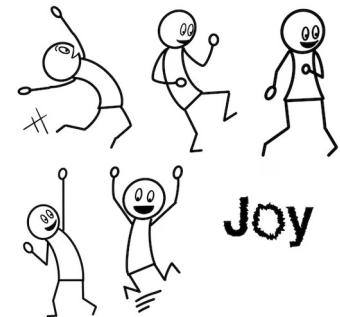
2

Debug

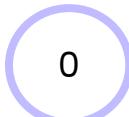


3

Address



Actionable Observability: Incident Management Cycle



Collect



Monitor



Debug



Address



Joy

Actionable Observability: Incident Management Cycle

0

Collect



1

Monitor



2

Debug



3

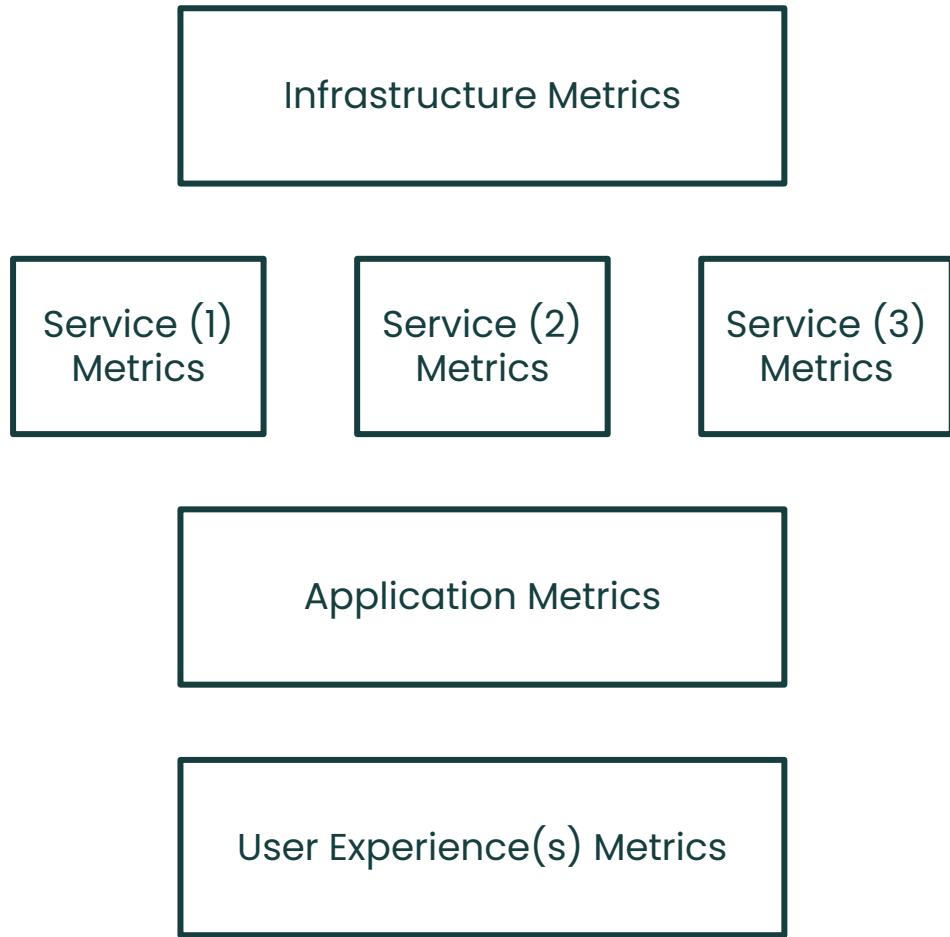
Address



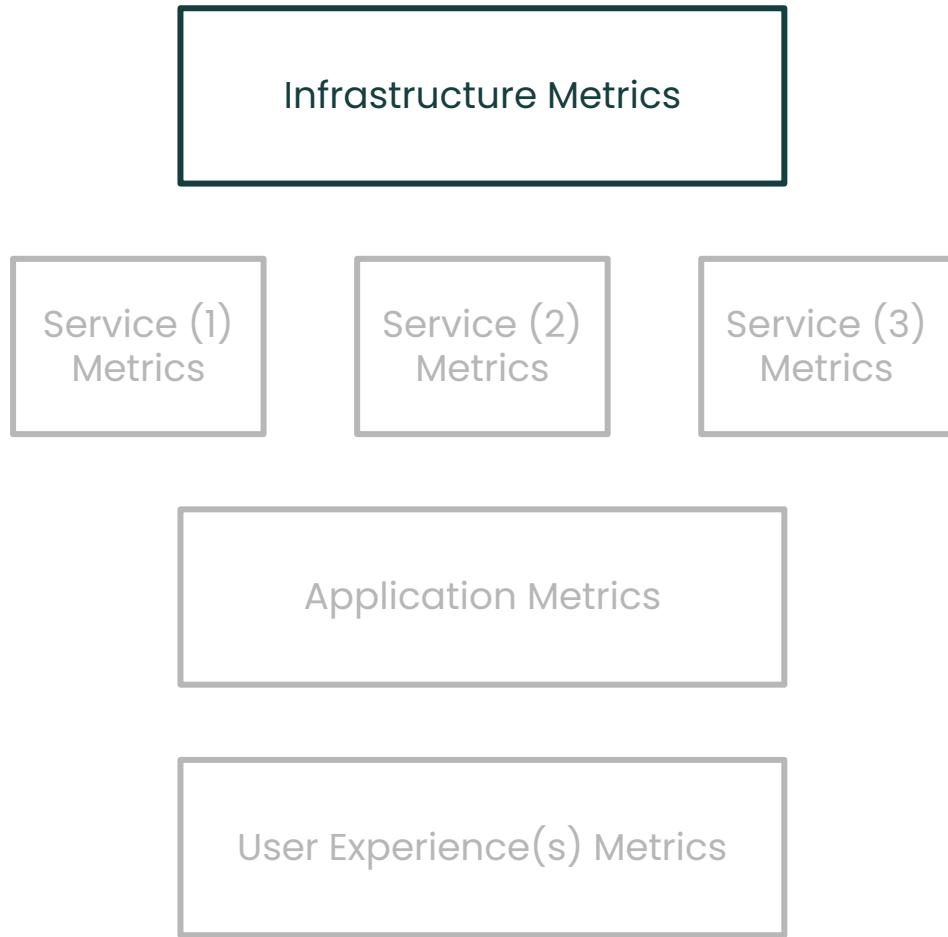
Monitoring & Alerting Strategy

The approach guiding how we choose metrics and alerts that represent failures impacting user experiences.

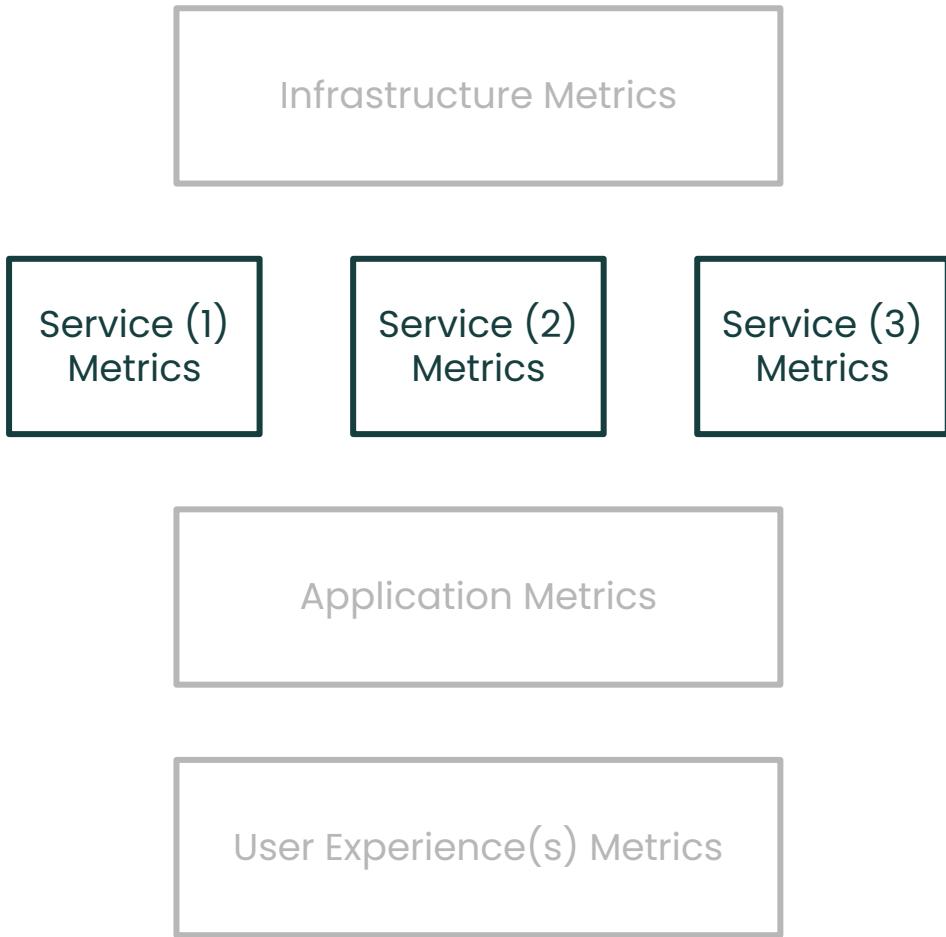
Metric Categories



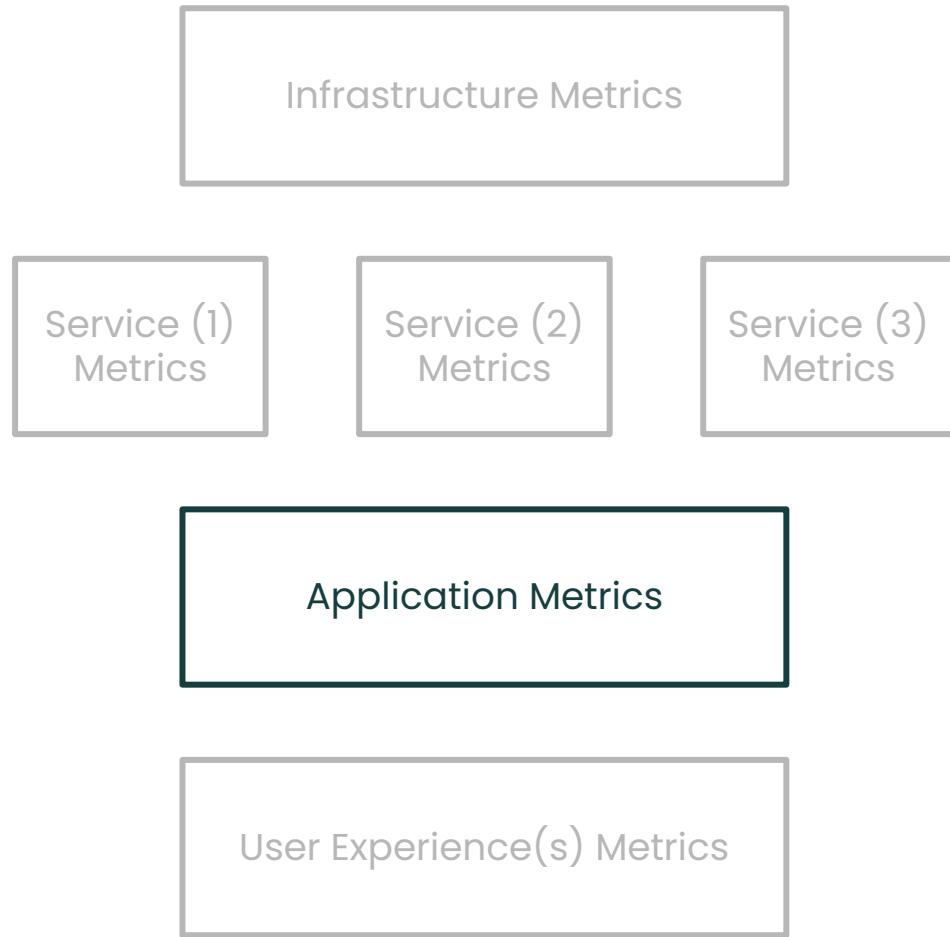
Metric Categories: Infra Layer



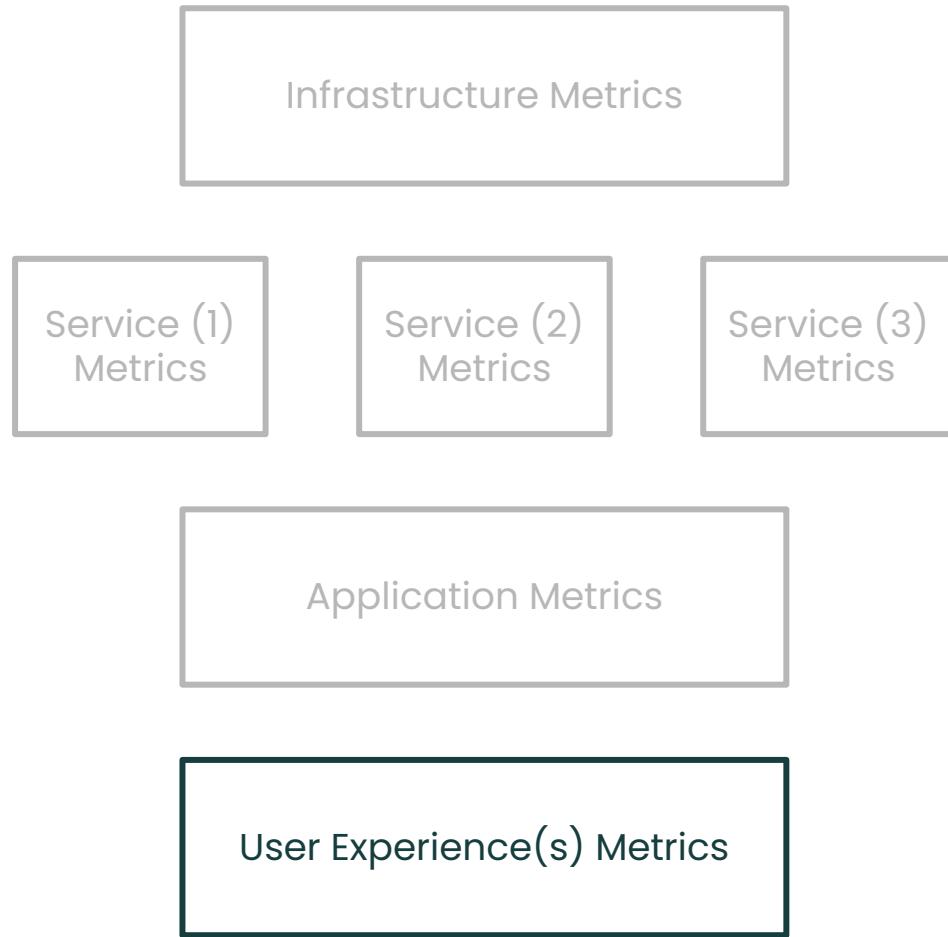
Metric Categories: Service Layer



Metric Categories: App Layer



Metric Categories: User Experience



Monitors & Alerts

Monitors represent the metrics that would indicate a user experience has degraded, whereas Alerts define the conditions that should indicate a degradation.

Monitors & Alerts

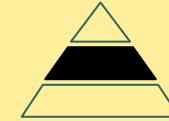


Patterns
Support
Stack

1. Define your user* journeys

What parts of our software are crucial enough to warrant an interrupt?

Monitors & Alerts



Patterns
Support
Stack

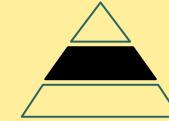
1. Define your user journeys

What parts of our software are crucial enough to warrant an interrupt?

2. Define your metrics

Choose metrics that represent the reliability needs of your users.

Monitors & Alerts



Patterns
Support
Stack

1. Define your user journeys

What parts of our software are crucial enough to warrant an interrupt?

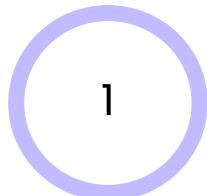
2. Define your metrics

Choose metrics that represent the reliability needs of your users.

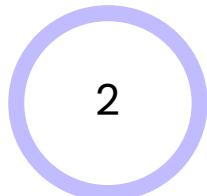
3. Configure the thresholds

Choose values or thresholds that indicate system degradation.

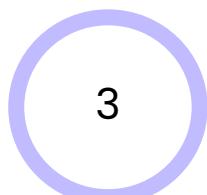
Monitors & Alerts



Define your user* journeys

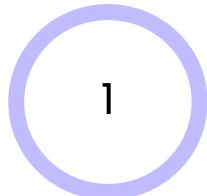


Define your metrics



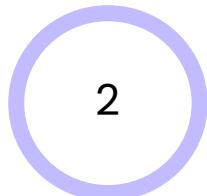
Configure the thresholds

Monitors & Alerts

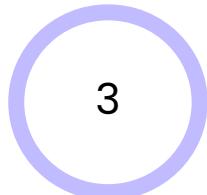


Define your user* journeys

Pub/Sub notifications for content publishing.

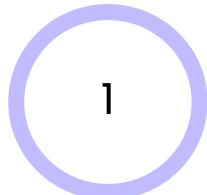


Define your metrics



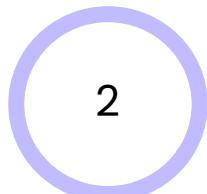
Configure the thresholds

Monitors & Alerts

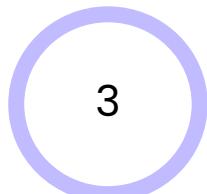


Define your user* journeys

Pub/Sub notifications for content publishing.

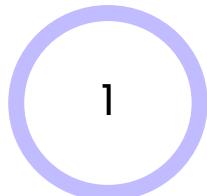


Define your metrics



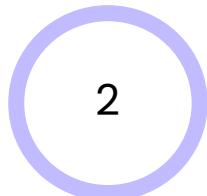
Configure the thresholds

Monitors & Alerts



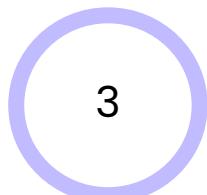
Define your user* journeys

Pub/Sub notifications for content publishing.



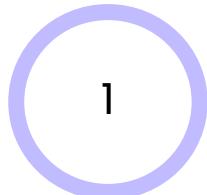
Define your metrics

Delivery Duration as our north star metric.



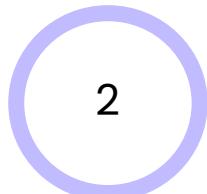
Configure the thresholds

Monitors & Alerts



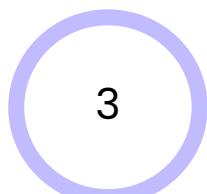
Define your user* journeys

Pub/Sub notifications for content publishing.



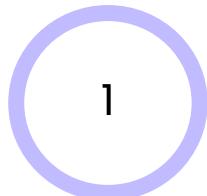
Define your metrics

Delivery Duration as our north star metric.



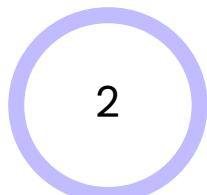
Configure the thresholds

Monitors & Alerts



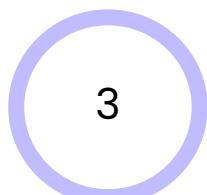
Define your user* journeys

Pub/Sub notifications for content publishing.



Define your metrics

Delivery Duration as our north star metric.



Configure the thresholds

Logging indicator: 250ms, Leading Indicator: 300ms

Service Level Objectives (SLO)

SLOs clarify our collective expectations for what experience we should be providing users by setting reliability targets.

Error Budgets

Error Budgets are a measurement of the overall state of a system.

Service Level Objectives



Patterns
Support
Stack

1. *Prioritize user journeys*

Focus on your most important product workflows.

Service Level Objectives



Patterns
Support
Stack

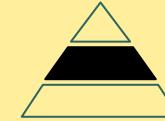
1. Prioritize user journeys

Focus on your most important product workflows.

2. Define your SLI

Choose an SLI that represents the reliability needs of your users.

Service Level Objectives



Patterns
Support
Stack

1. Prioritize user journeys

Focus on your most important product workflows.

2. Define your SLI

Choose an SLI that represents the reliability needs of your users.

3. Define reliability target

Choose a target that indicates the optimal* user experience you're looking to provide.

Service Level Objectives



Patterns
Support
Stack

1. Prioritize user journeys

Focus on your most important product workflows.

2. Define your SLI

Choose an SLI that represents the reliability needs of your users.

3. Define reliability target

Choose a target that indicates the optimal* user experience you're looking to provide.

4. Choose an alerting strategy

Consider the trade-offs of alerting based on consumption or burn rate.

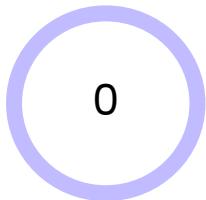
SLOs

0

Prioritize your user journeys

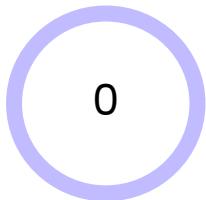
5 critical user journeys, prioritizing “Serving an Ad.”

SLOs



Prioritize your user journeys

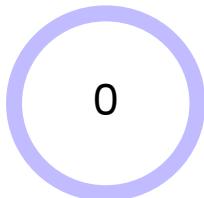
5 critical user journeys, prioritizing “Serving an Ad.”



Define your SLI

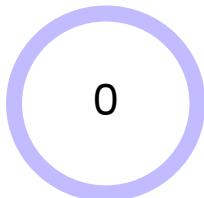
Composite SLI reflecting availability and load time.

SLOs



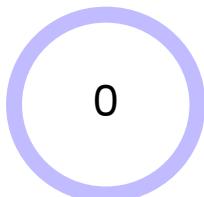
Prioritize your user journeys

5 critical user journeys, prioritizing “Serving an Ad.”



Define your SLI

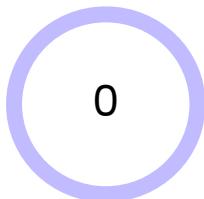
Composite SLI reflecting availability and load time.



Define Reliability time

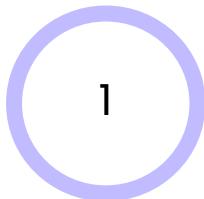
Criticality leads to a target of 99.9%

SLOs



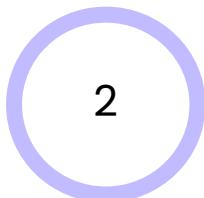
Prioritize your user journeys

5 critical user journeys, prioritizing "Serving an Ad."



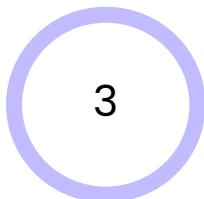
Define your SLI

Composite SLI reflecting availability and load time.



Define Reliability time

Criticality leads to a target of 99.9%



Choose an alerting strategy

Burn rate approach for the sake of precision.

Actionable Observability: Incident Management Cycle

0

Collect



1

Identify



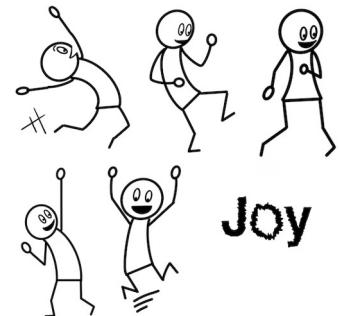
2

Debug



3

Address



Actionable Observability: Incident Management Cycle

0

Collect



1

Monitor



2

Debug



3

Address



Actionable Observability: Incident Management Cycle

0

Collect



1

Monitor



2

Debug



3

Address



Actionable Observability: Incident Management Cycle

0

Collect



1

Identify



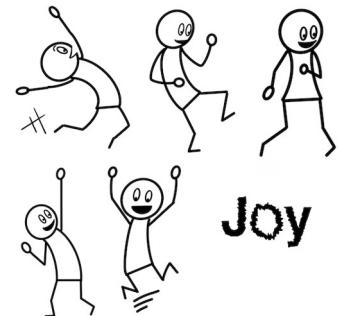
2

Debug



3

Address



Actionable Observability: Incident Management Cycle

0

Collect



1

Monitor



2

Debug

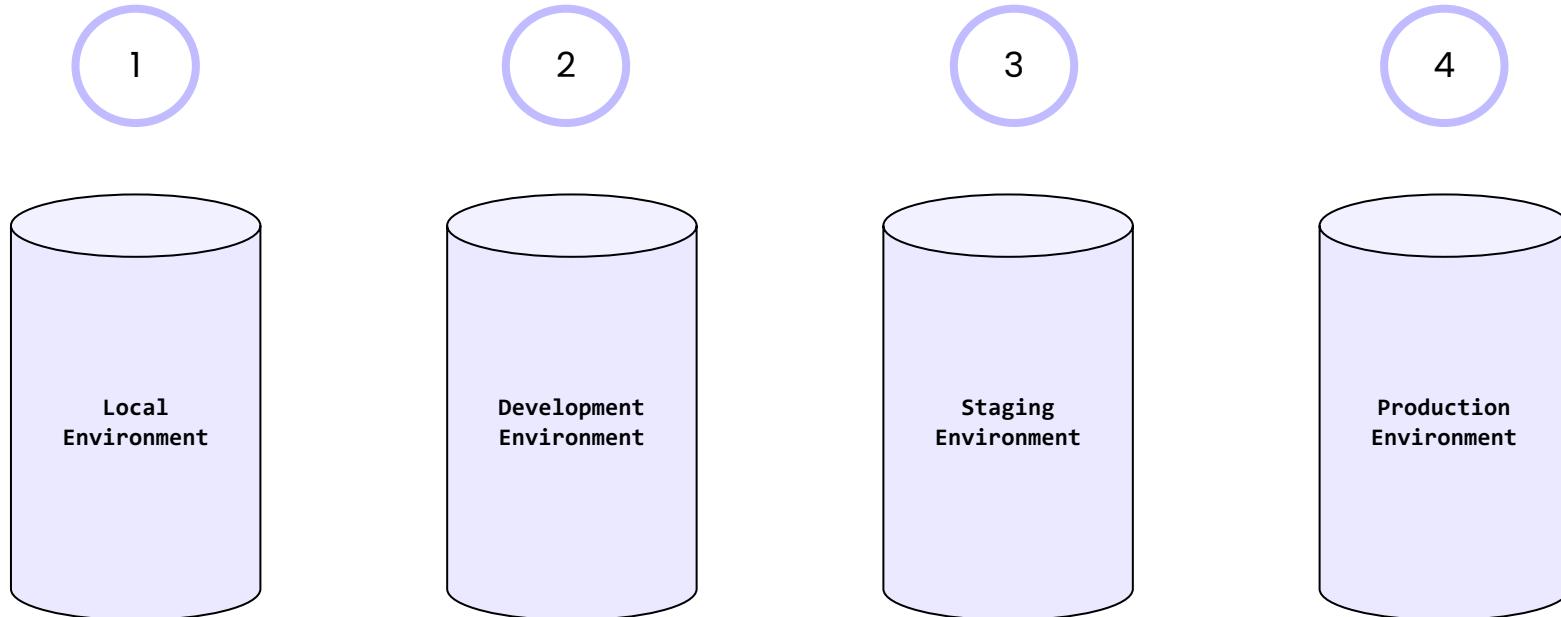


3

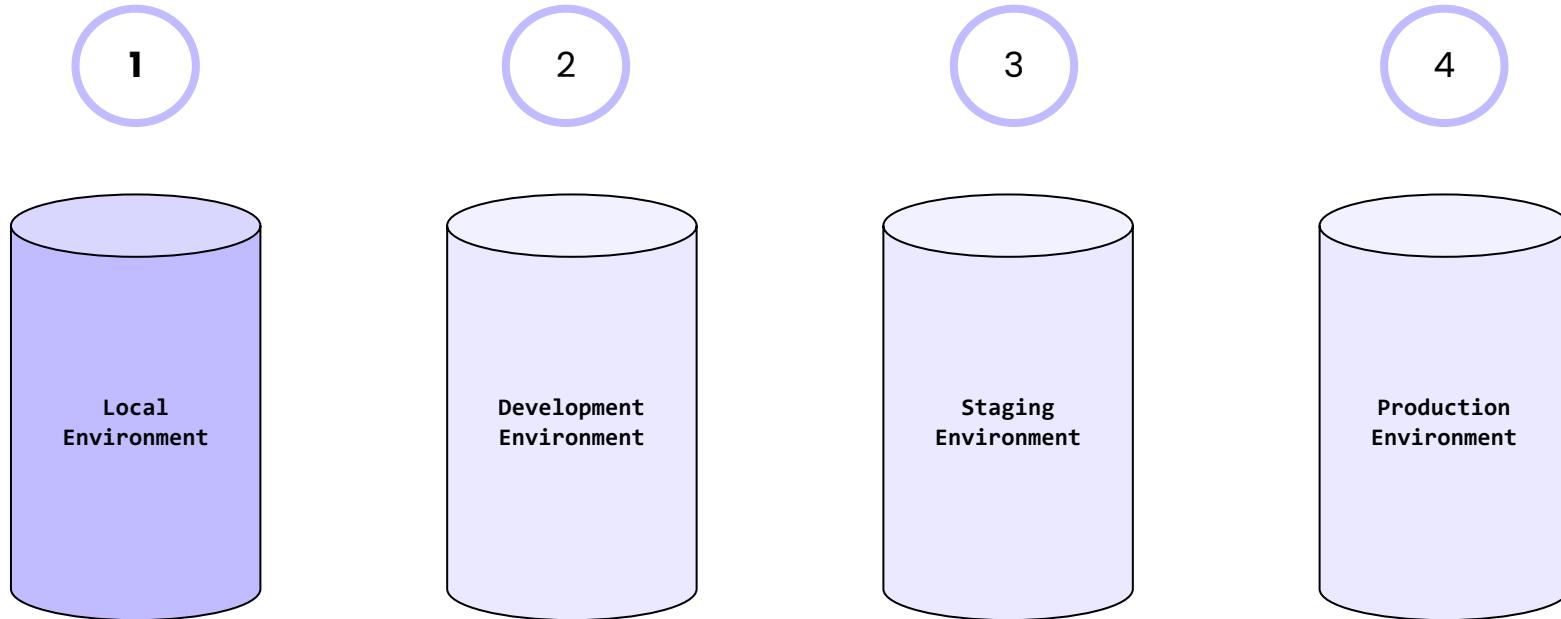
Address



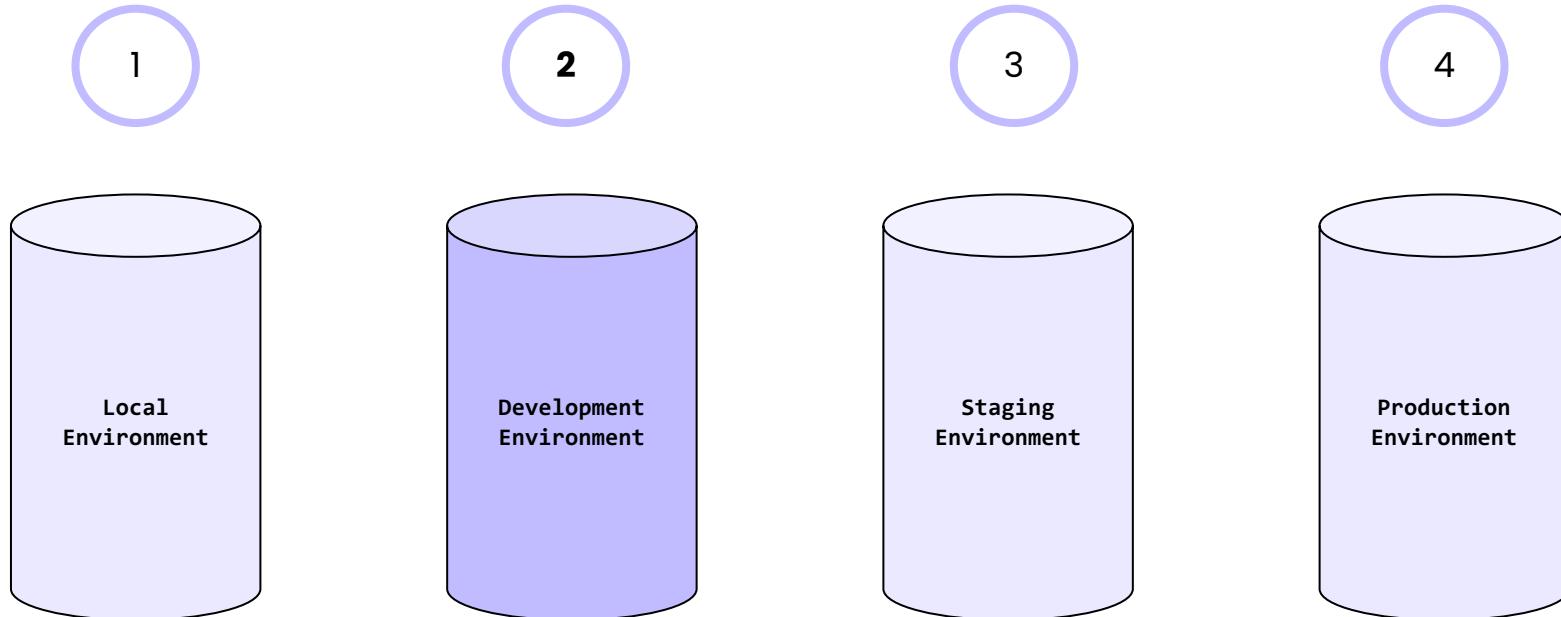
Environments



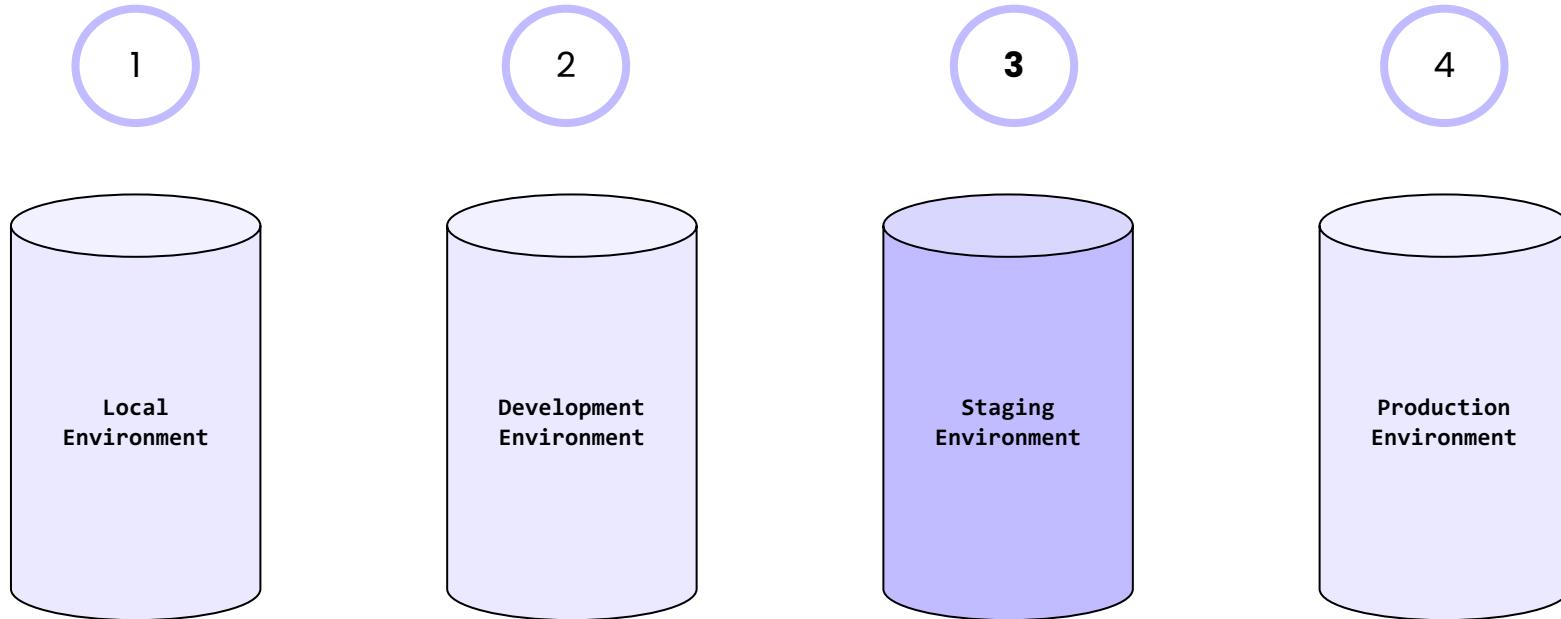
Environments



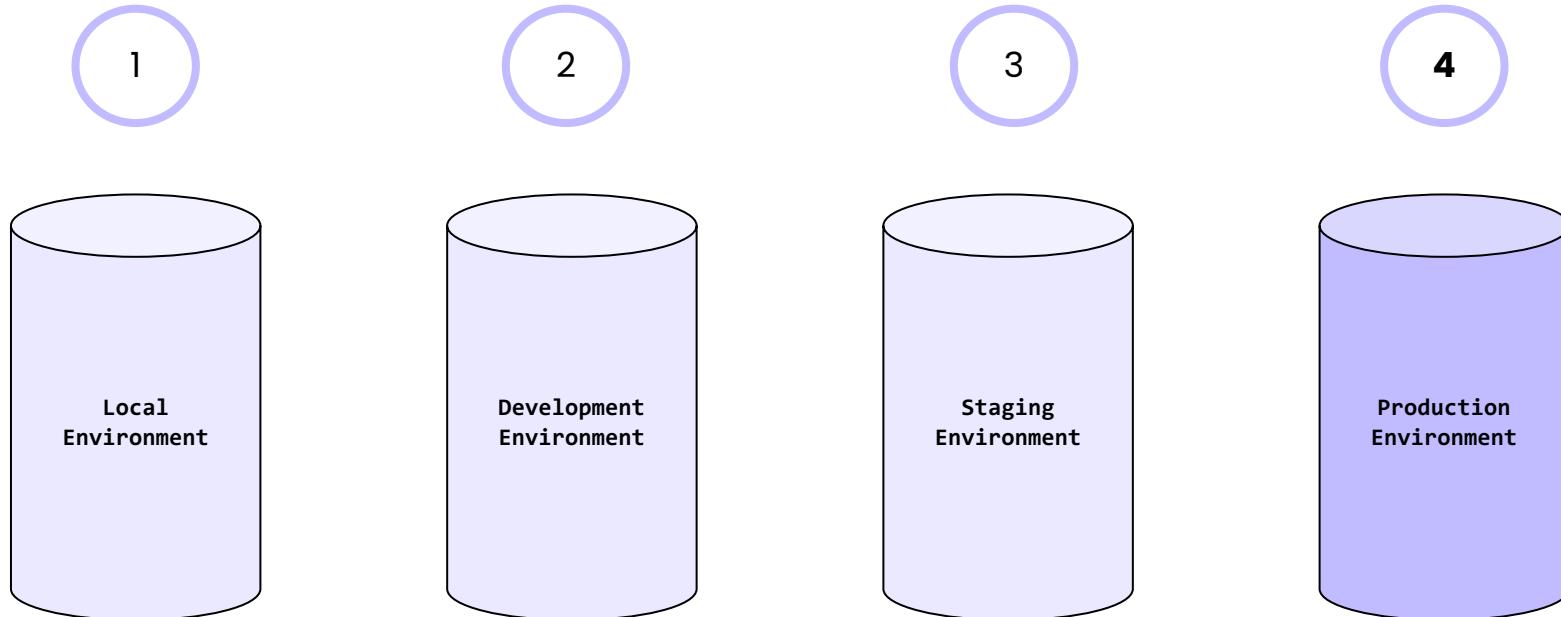
Environments



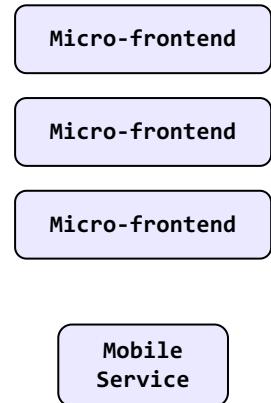
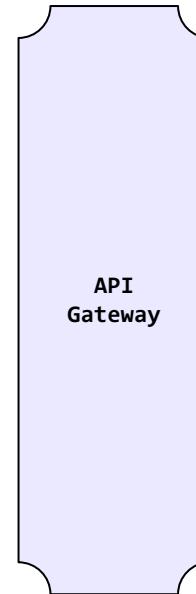
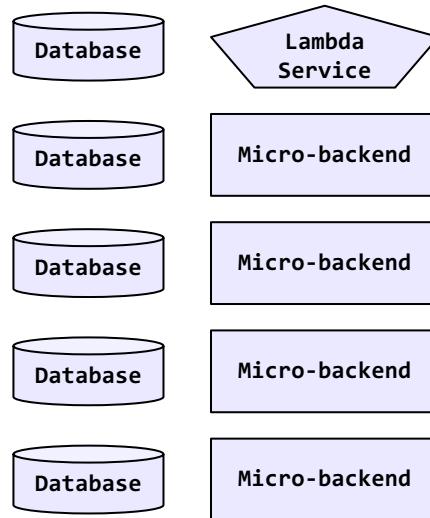
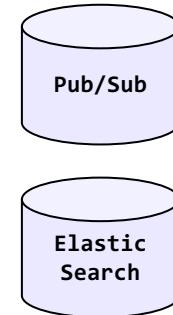
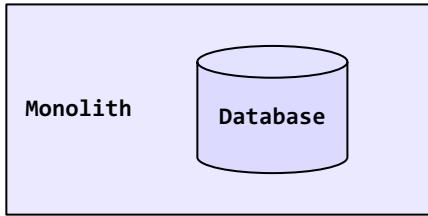
Environments



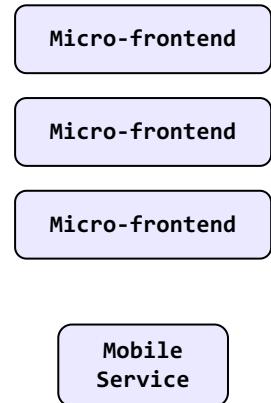
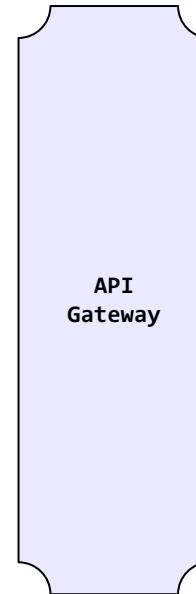
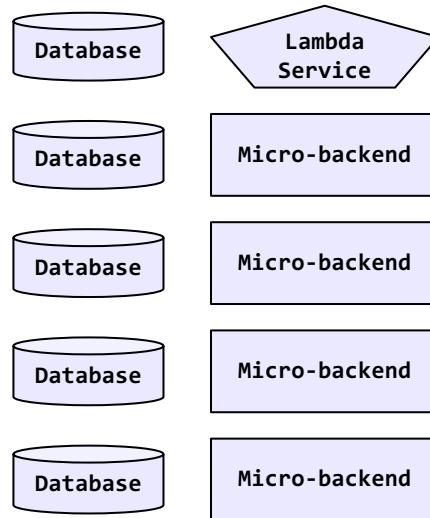
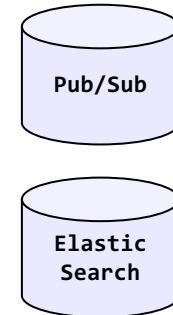
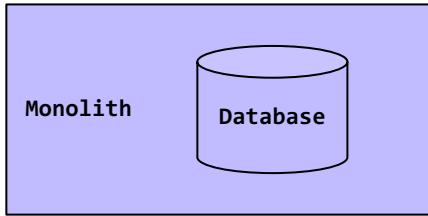
Environments



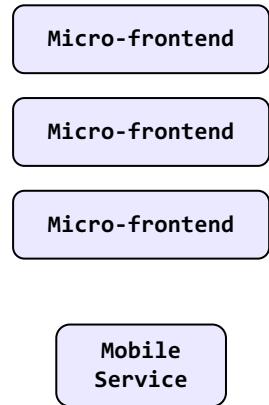
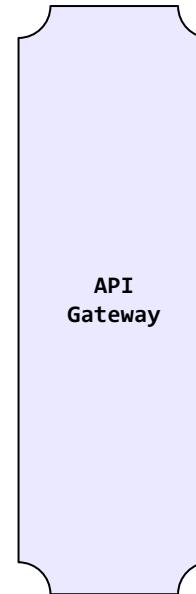
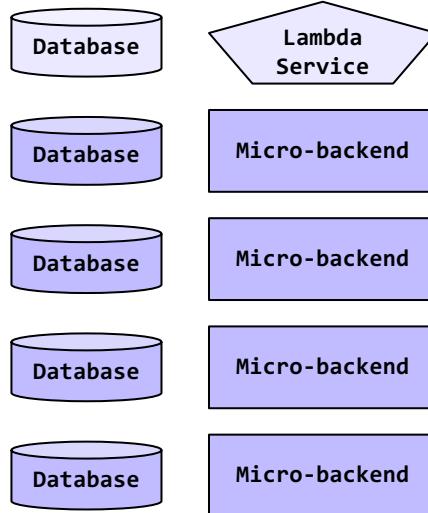
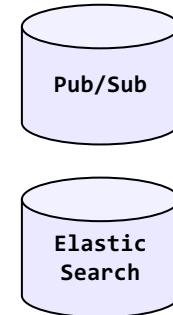
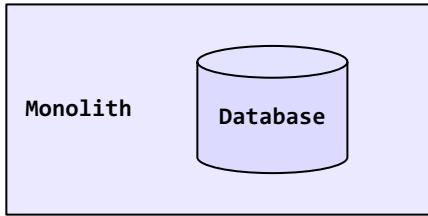
Service Architecture



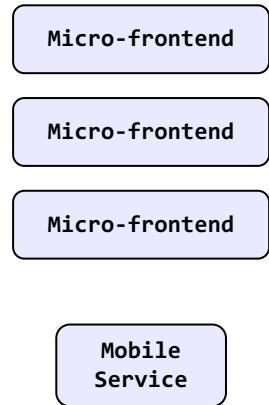
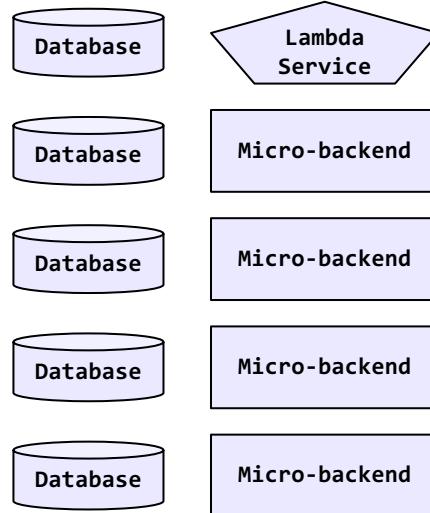
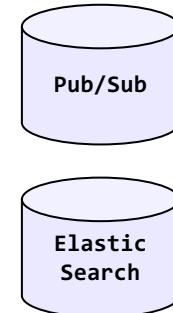
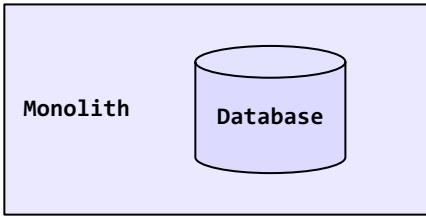
Service Architecture



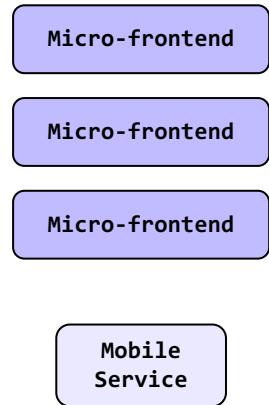
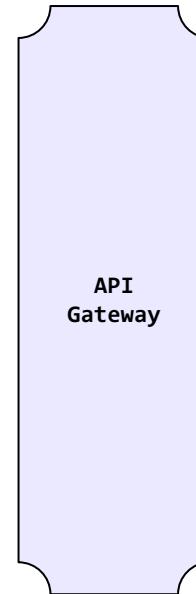
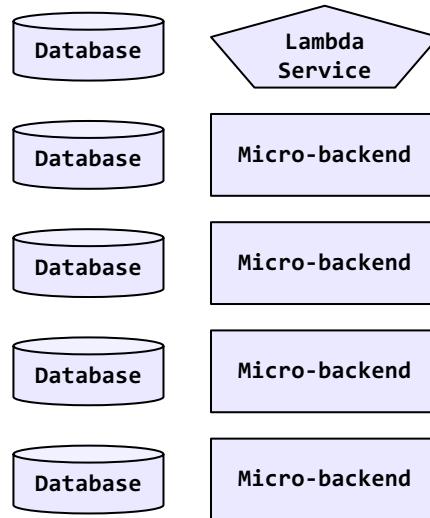
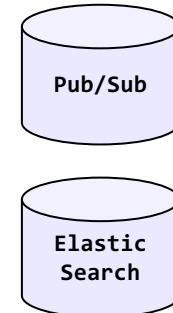
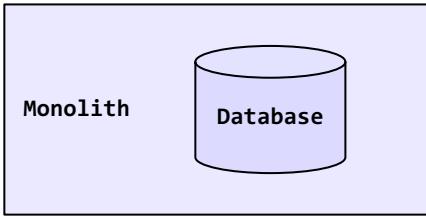
Service Architecture



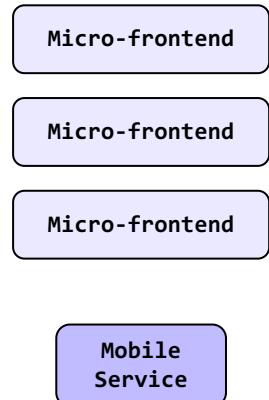
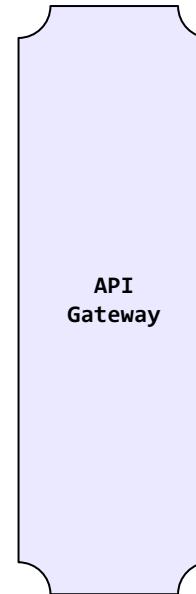
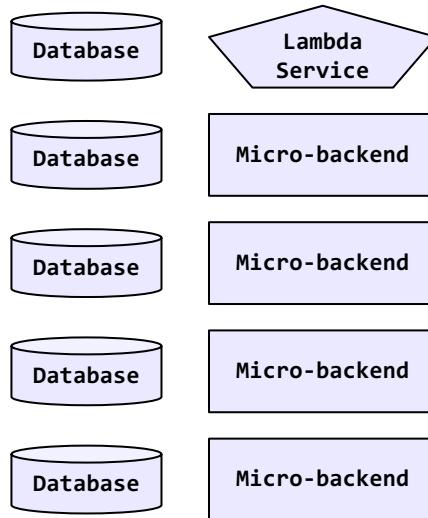
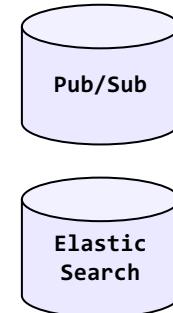
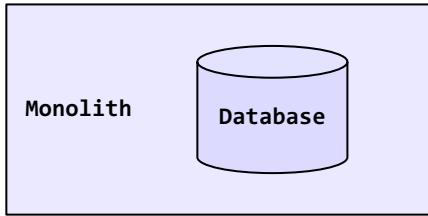
Service Architecture



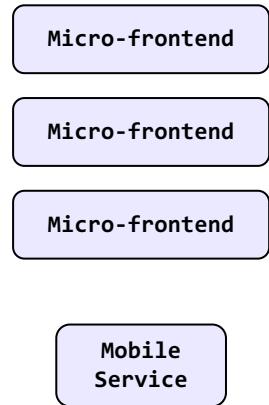
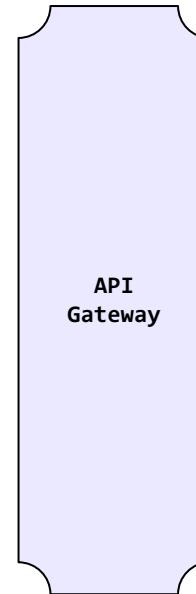
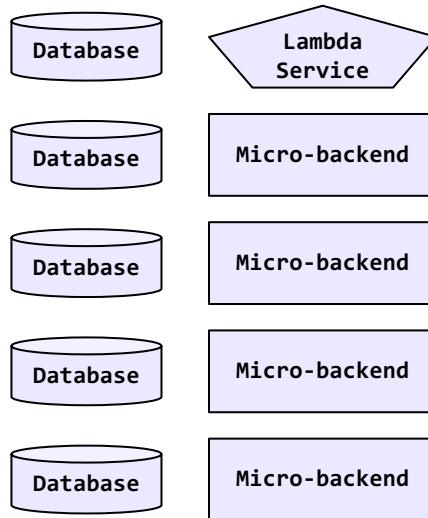
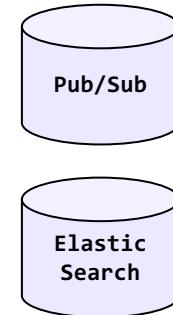
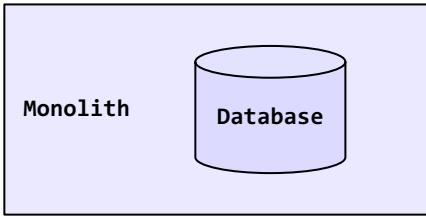
Service Architecture



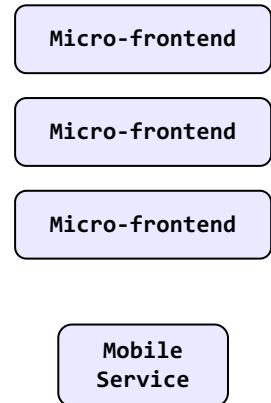
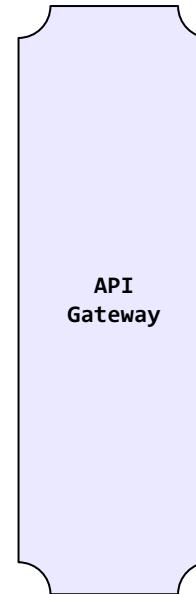
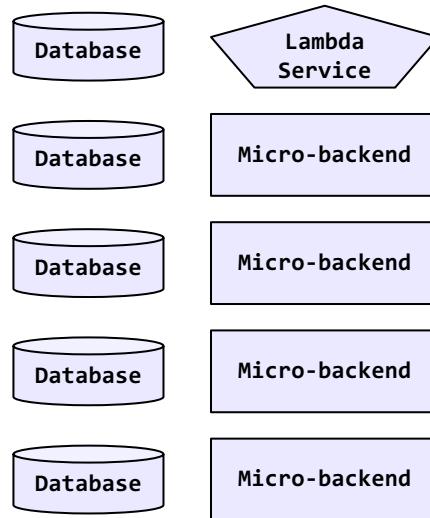
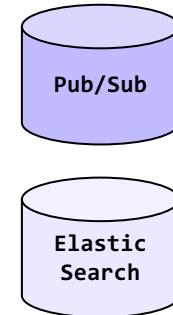
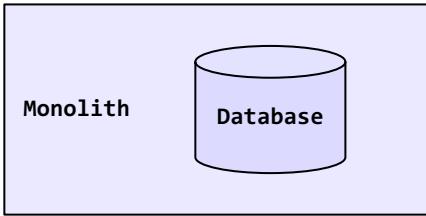
Service Architecture



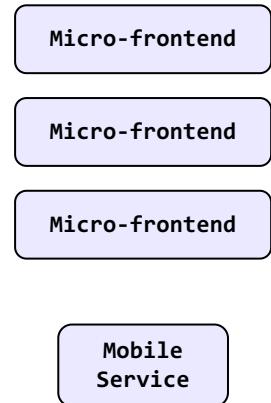
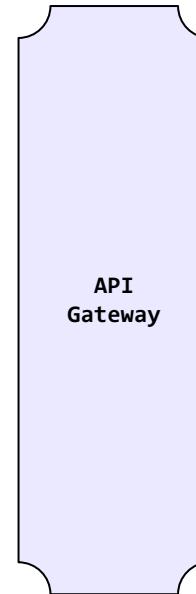
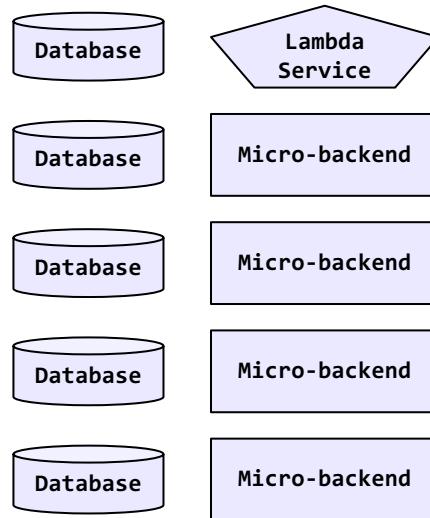
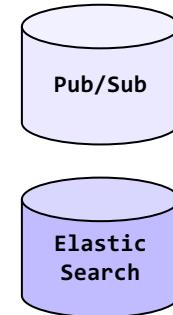
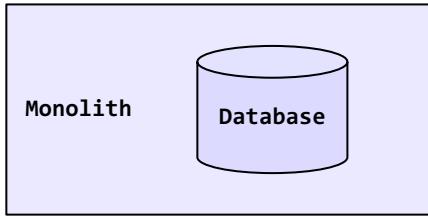
Service Architecture



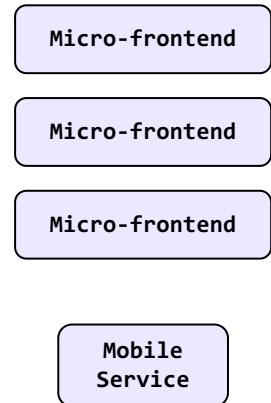
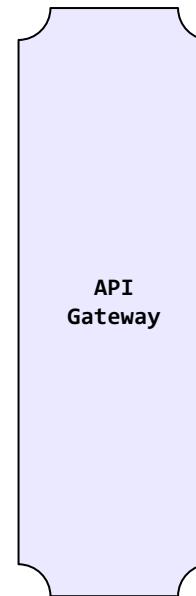
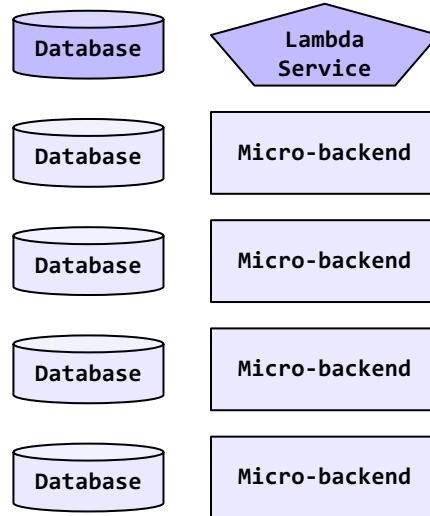
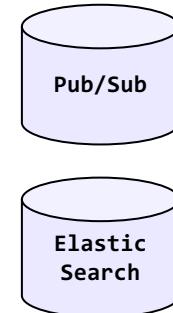
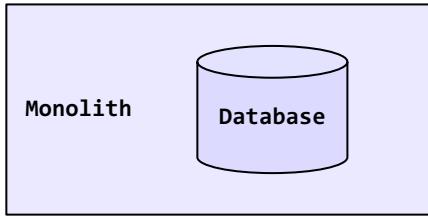
Service Architecture



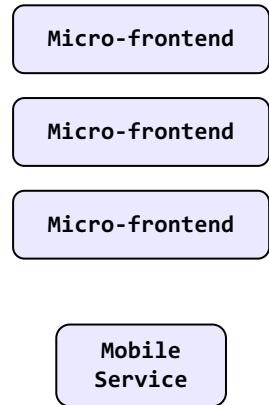
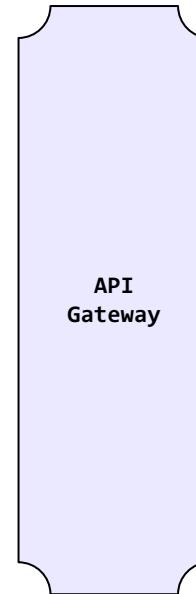
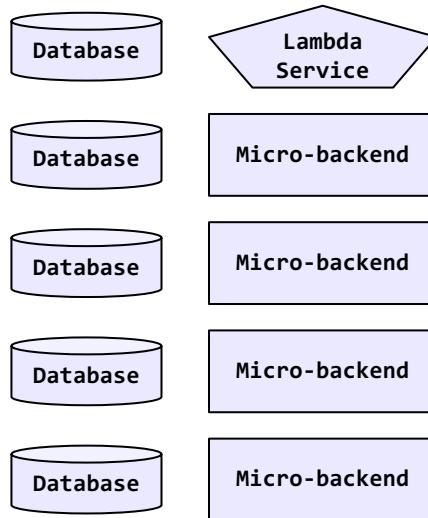
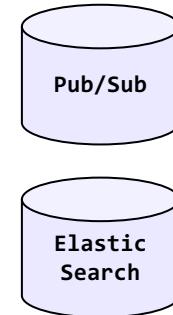
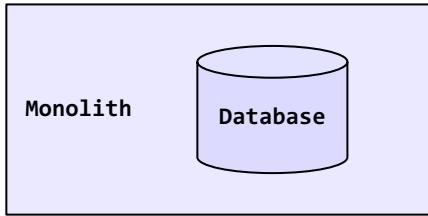
Service Architecture



Service Architecture

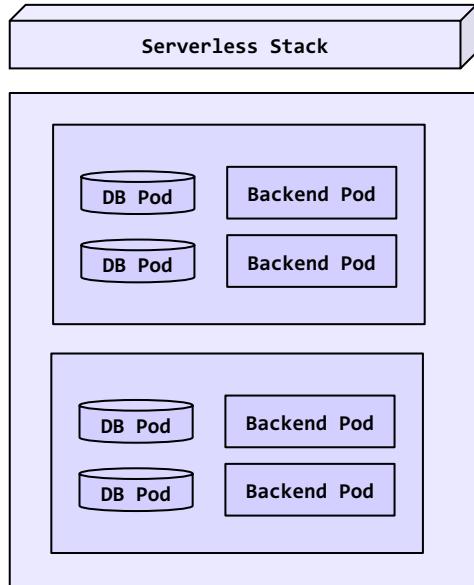


Service Architecture

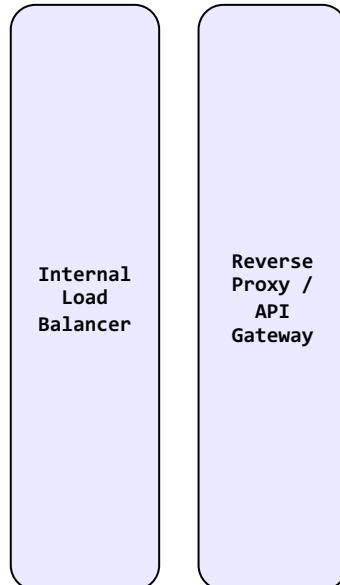


Infrastructure

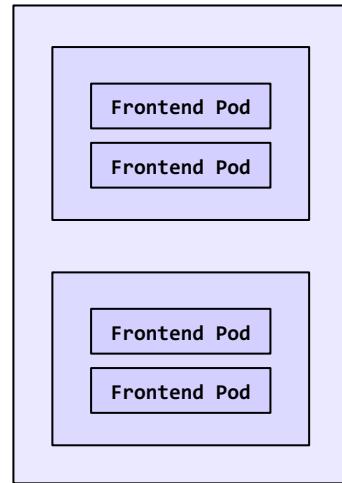
Infrastructure Components



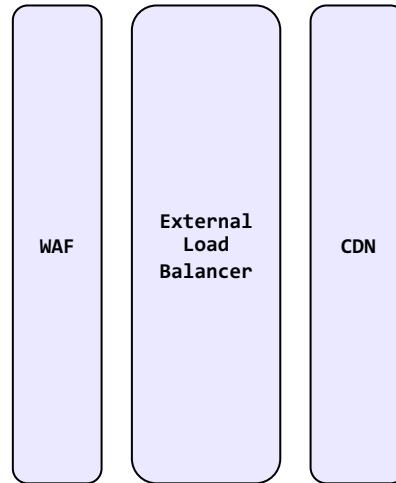
Network Components



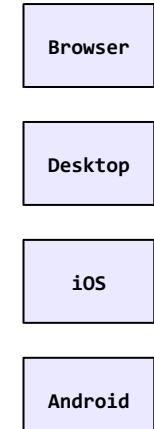
Infrastructure Components



Network Components

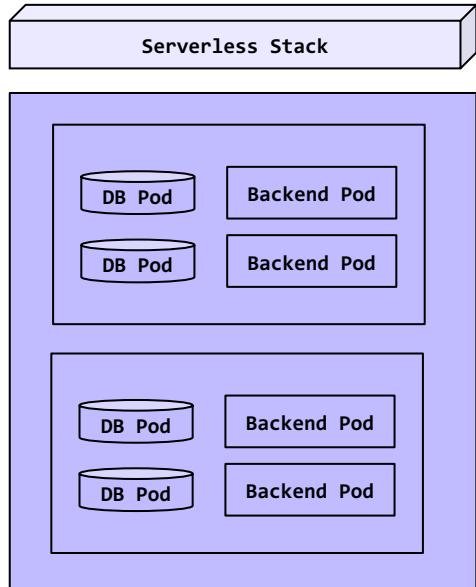


Clients

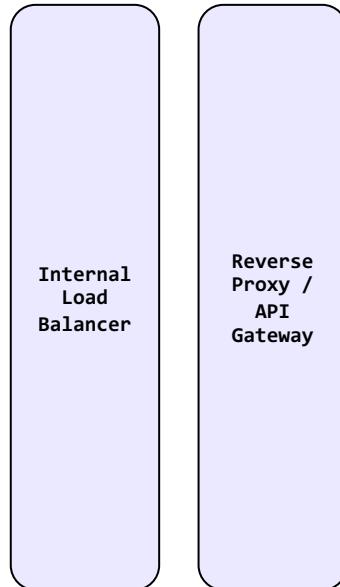


Infrastructure

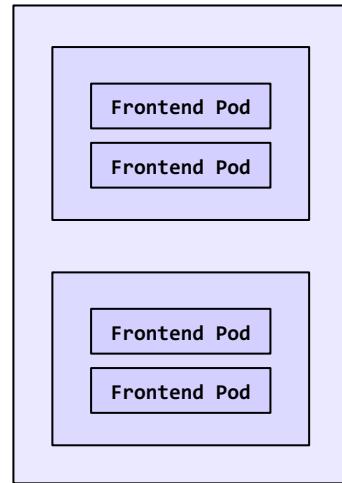
Infrastructure Components



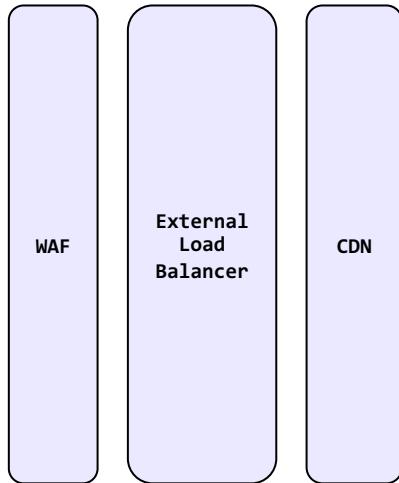
Network Components



Infrastructure Components



Network Components

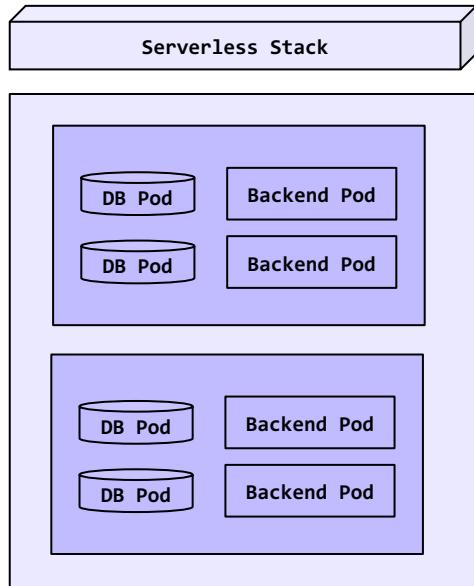


Clients

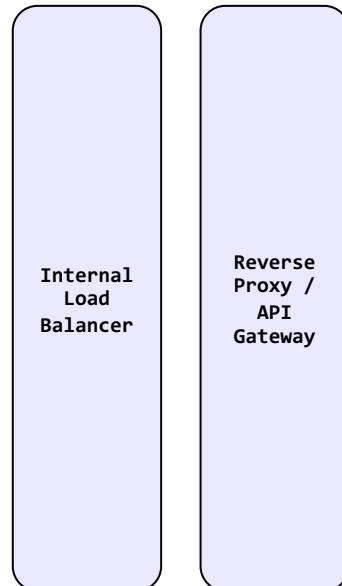


Infrastructure

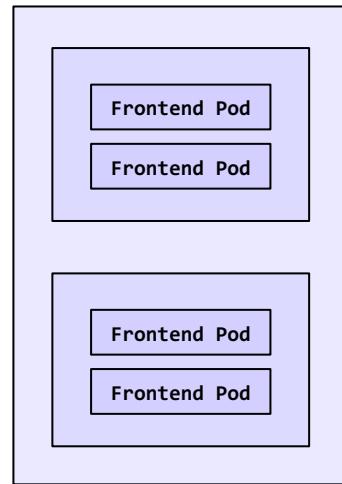
Infrastructure Components



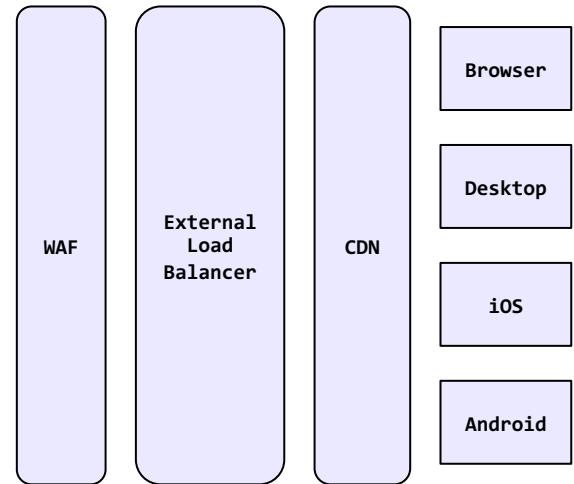
Network Components



Infrastructure Components

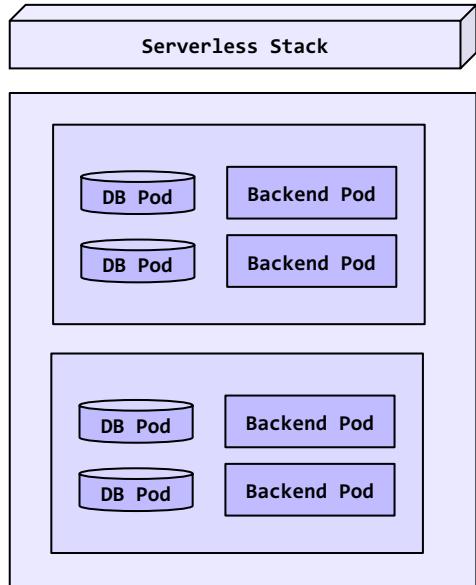


Network Components

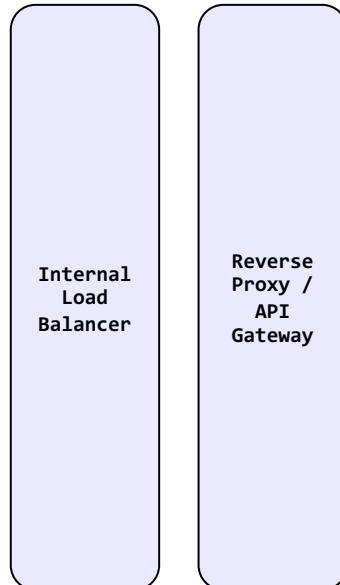


Infrastructure

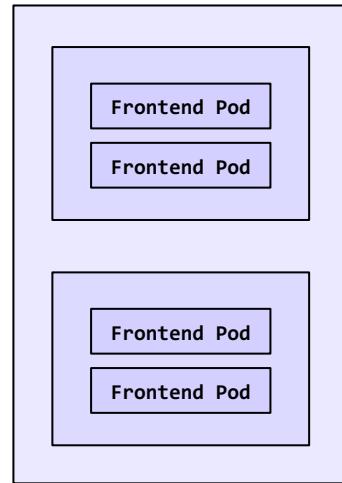
Infrastructure Components



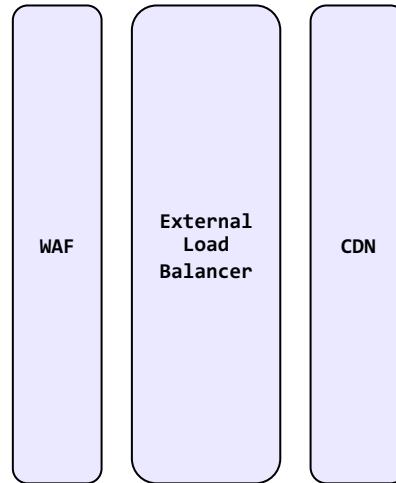
Network Components



Infrastructure Components



Network Components

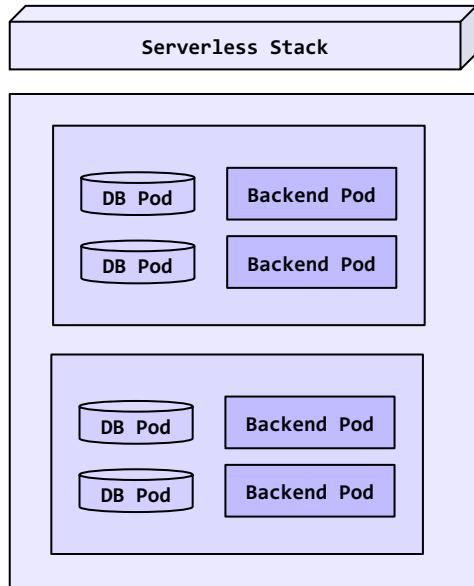


Clients

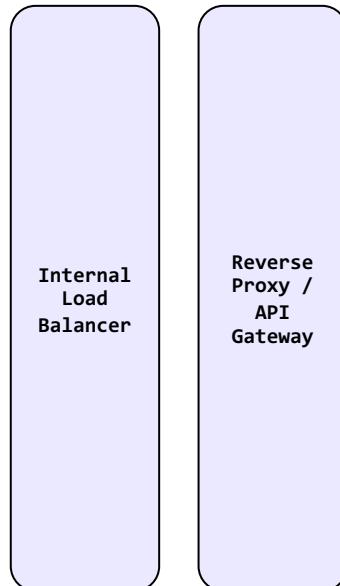


Infrastructure

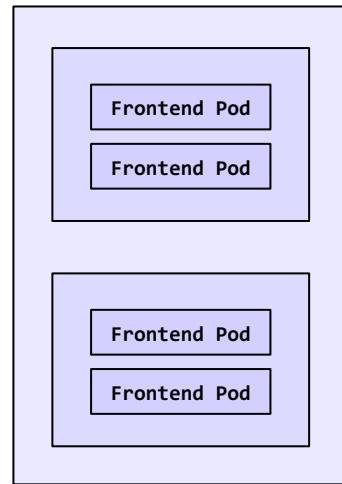
Infrastructure Components



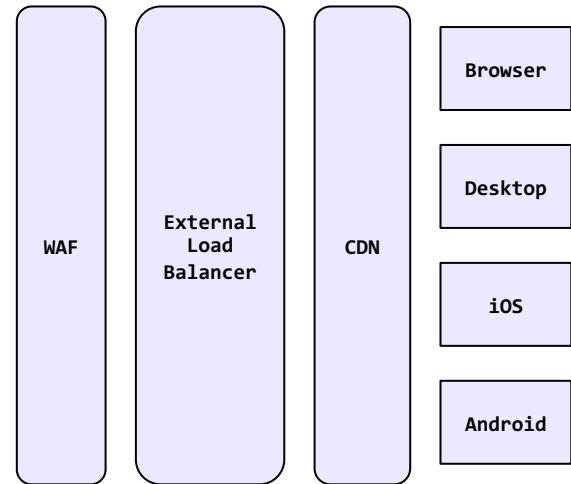
Network Components



Infrastructure Components



Network Components

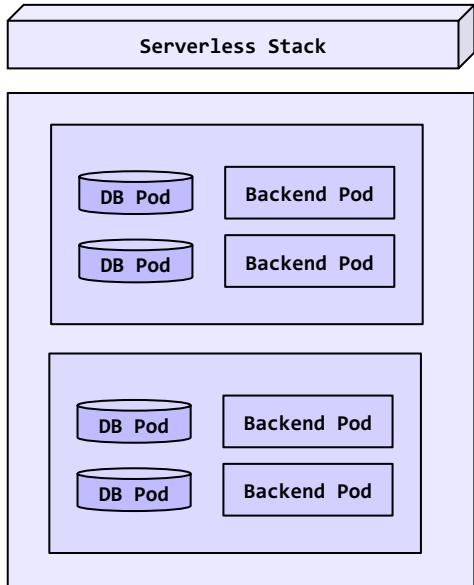


Clients

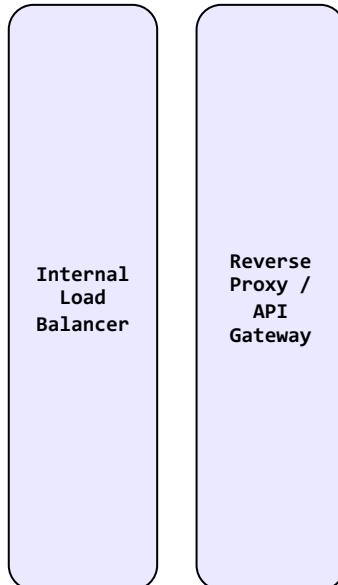


Infrastructure

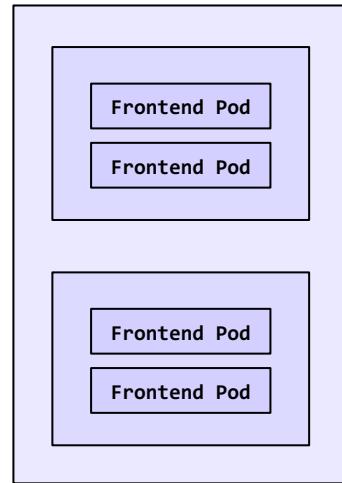
Infrastructure Components



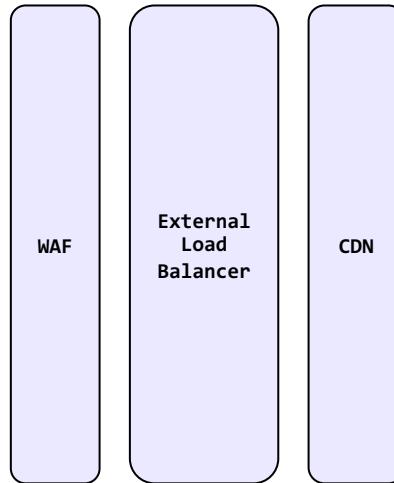
Network Components



Infrastructure Components



Network Components

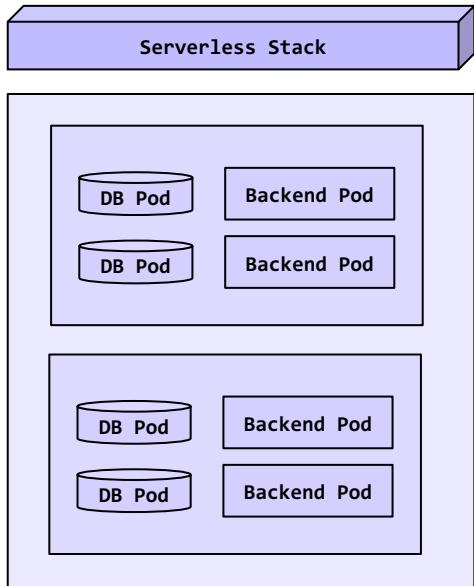


Clients

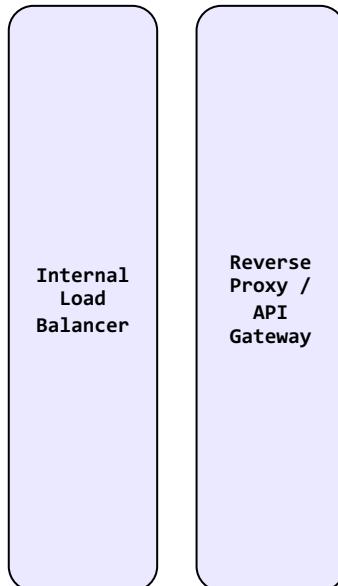


Infrastructure

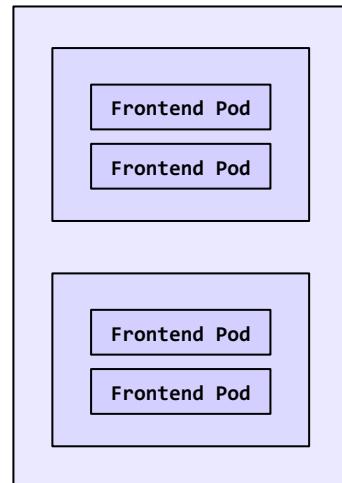
Infrastructure Components



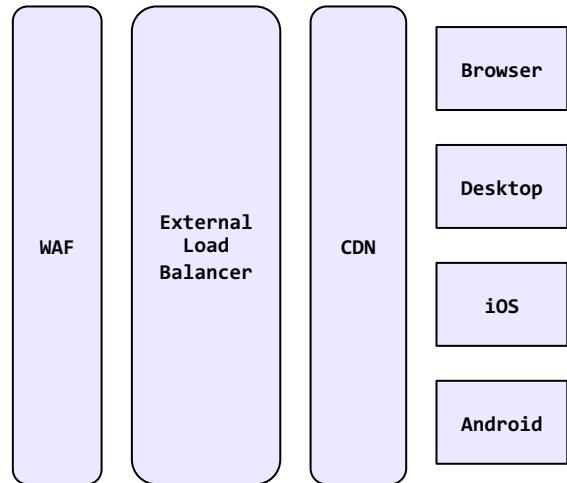
Network Components



Infrastructure Components



Network Components

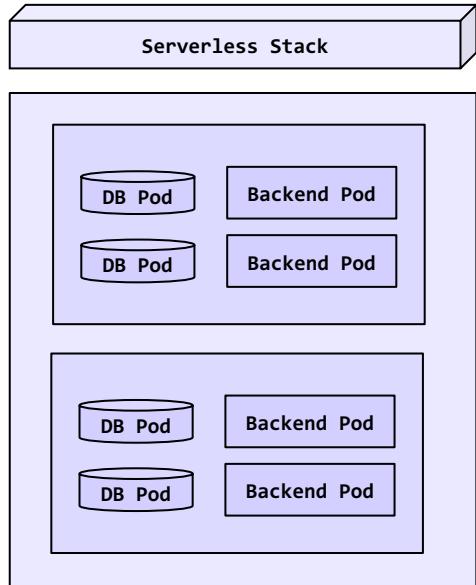


Clients

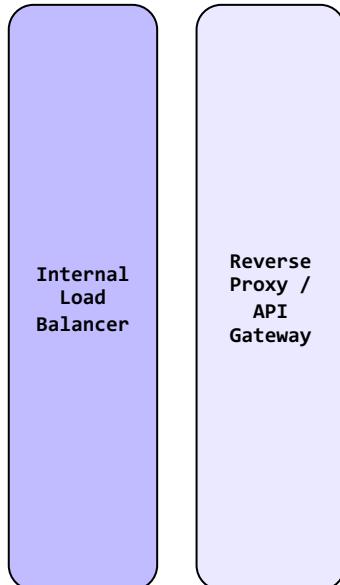


Infrastructure

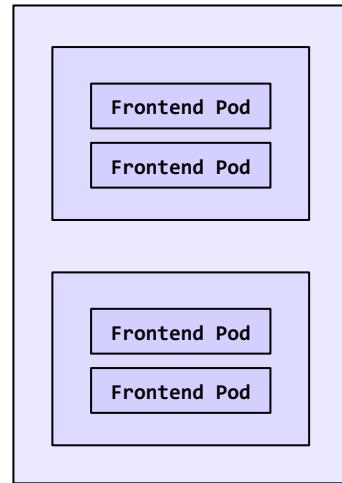
Infrastructure Components



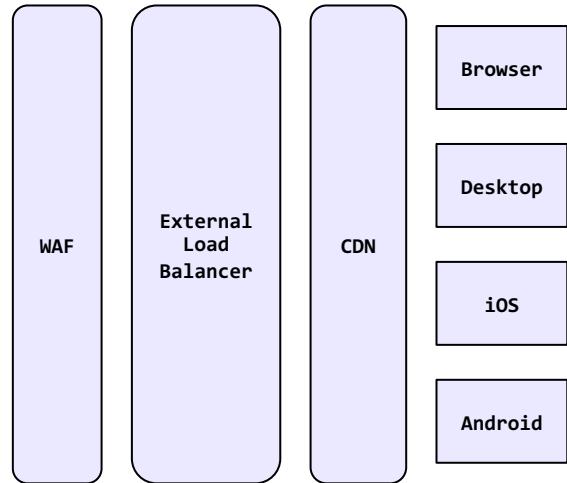
Network Components



Infrastructure Components

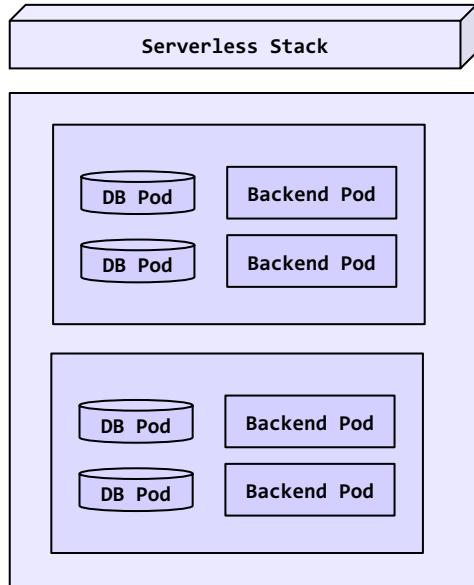


Network Components

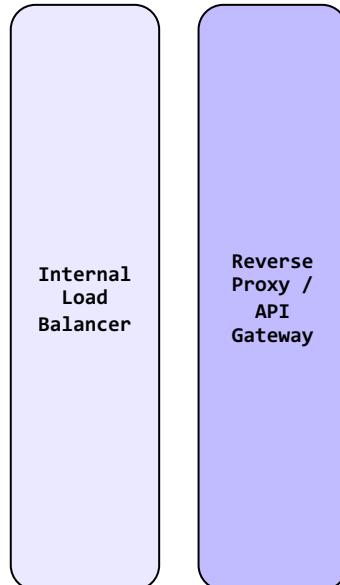


Infrastructure

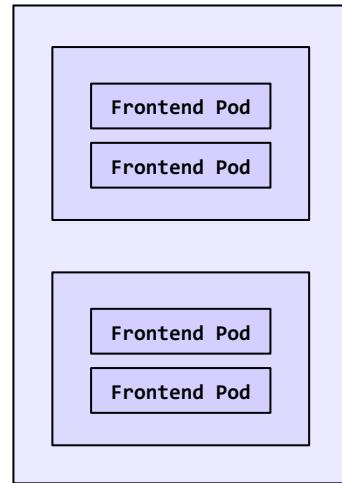
Infrastructure Components



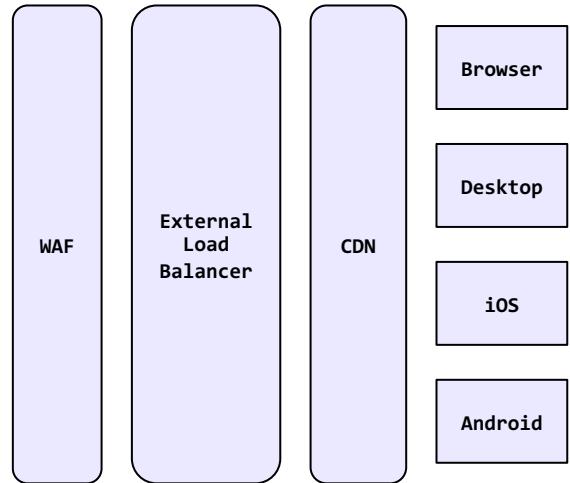
Network Components



Infrastructure Components

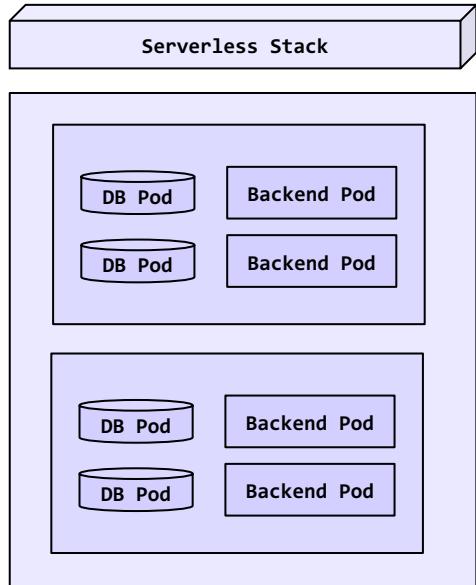


Network Components

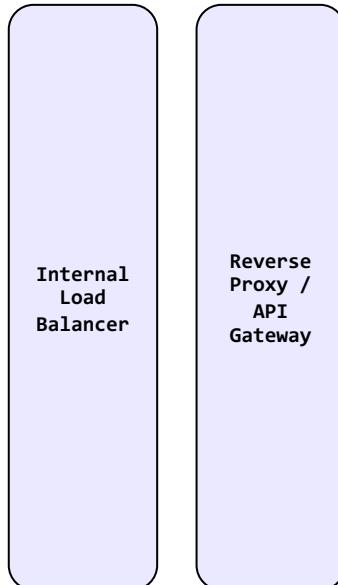


Infrastructure

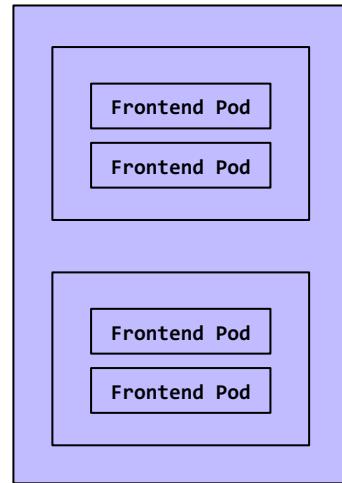
Infrastructure Components



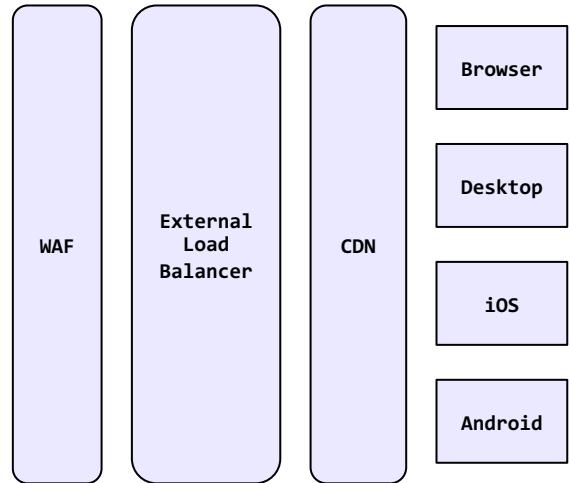
Network Components



Infrastructure Components



Network Components

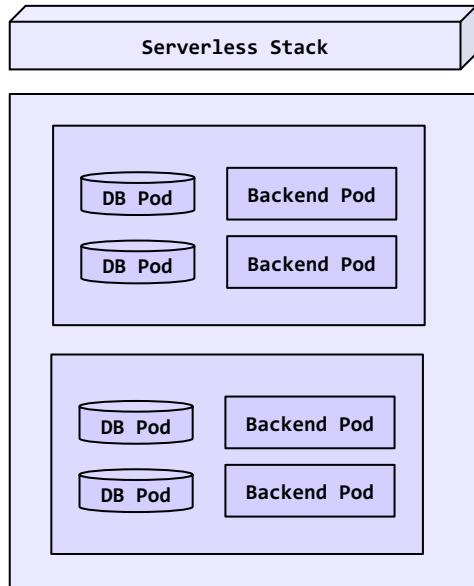


Clients

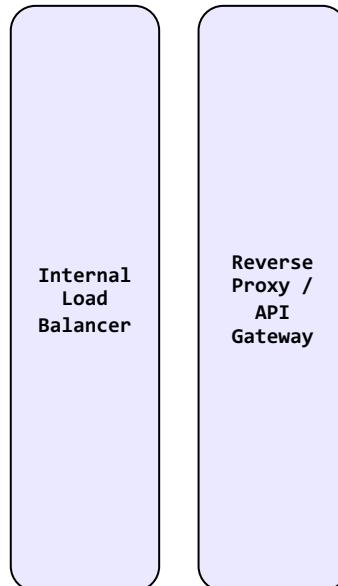


Infrastructure

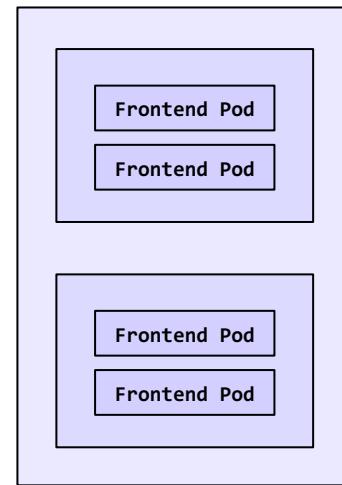
Infrastructure Components



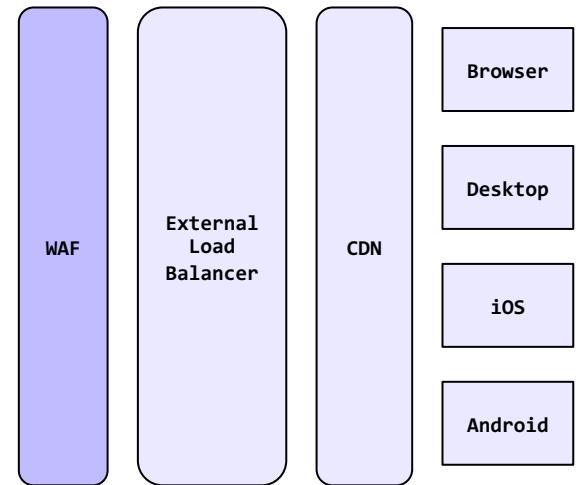
Network Components



Infrastructure Components

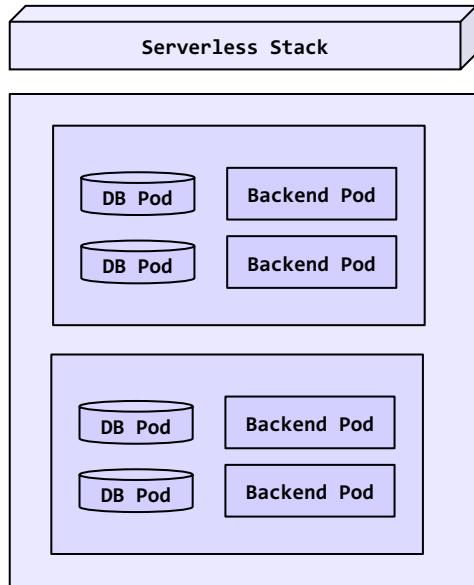


Network Components

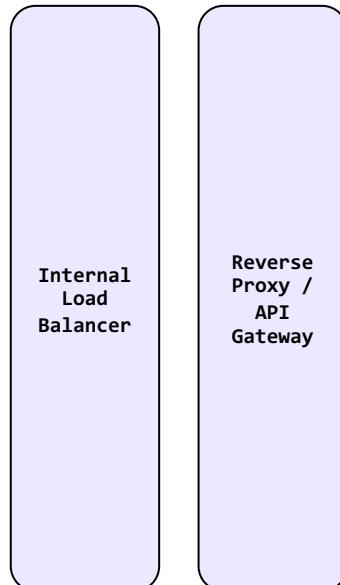


Infrastructure

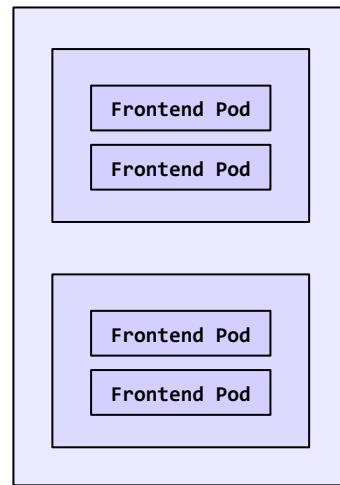
Infrastructure Components



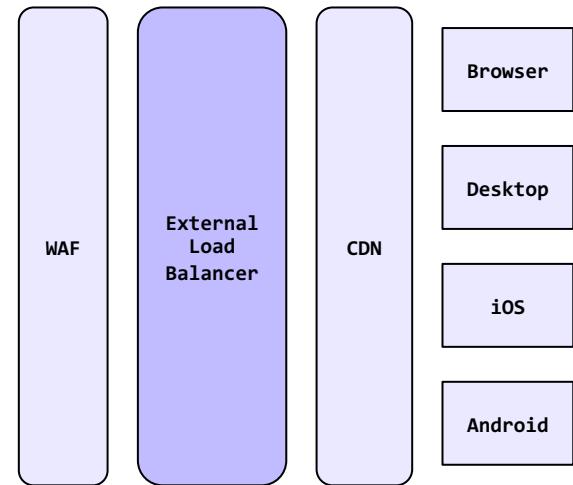
Network Components



Infrastructure Components



Network Components

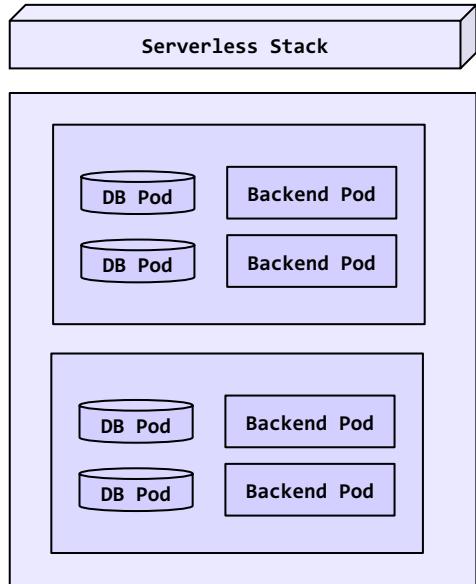


Clients

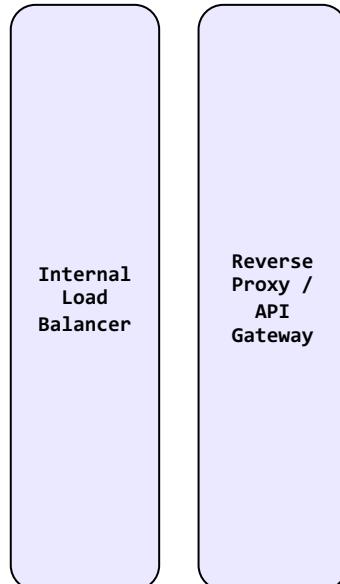


Infrastructure

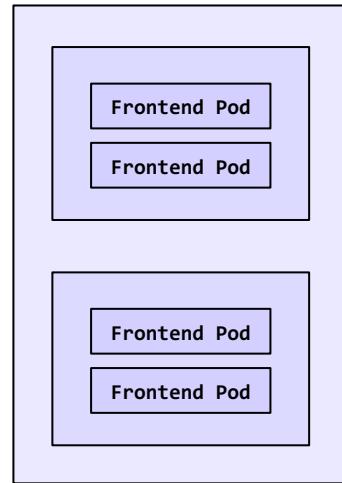
Infrastructure Components



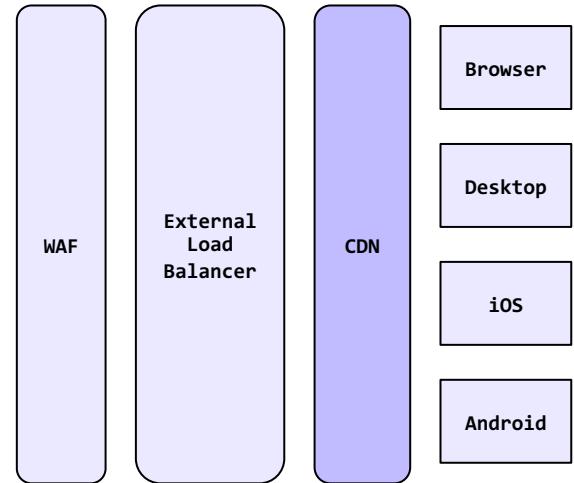
Network Components



Infrastructure Components



Network Components

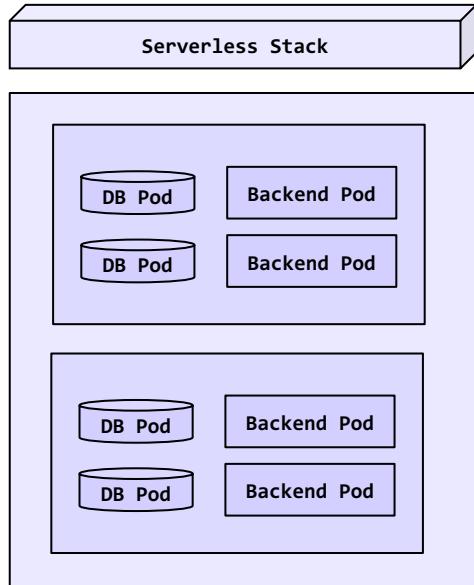


Clients

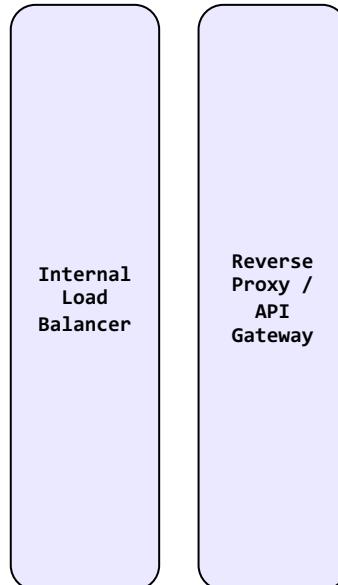


Infrastructure

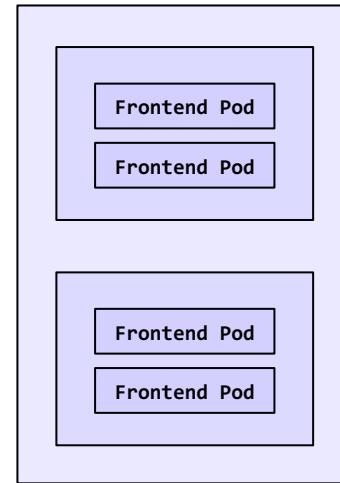
Infrastructure Components



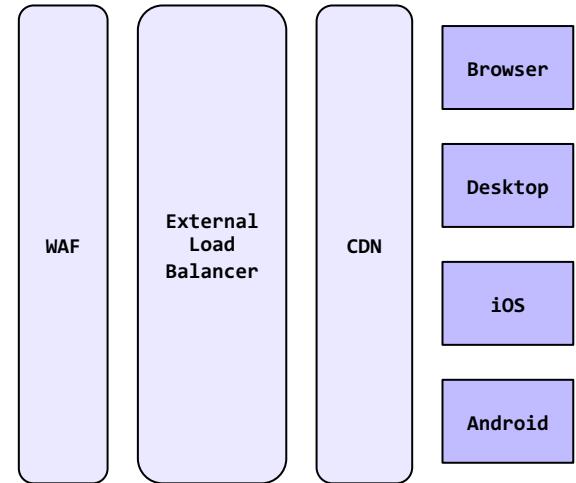
Network Components



Infrastructure Components



Network Components

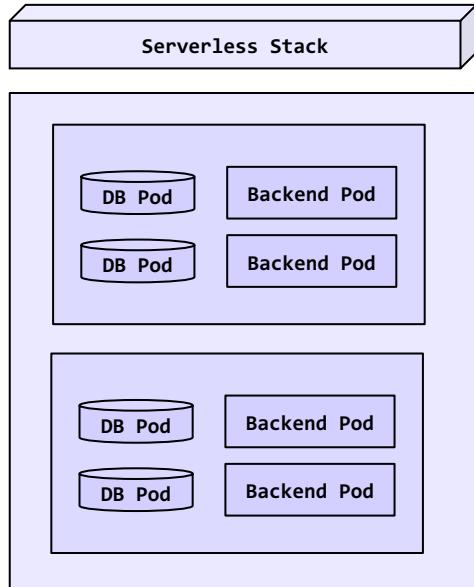


Clients

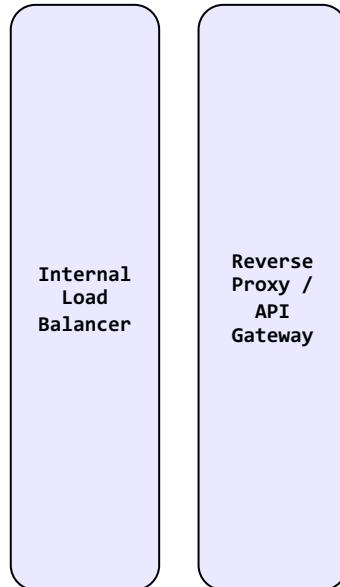


Infrastructure

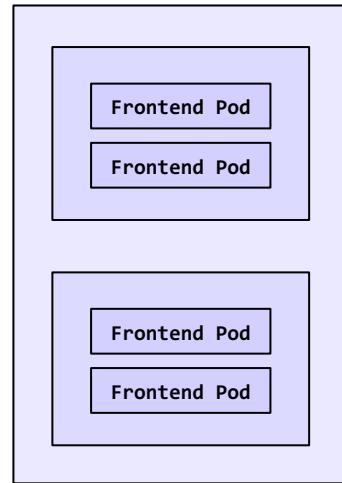
Infrastructure Components



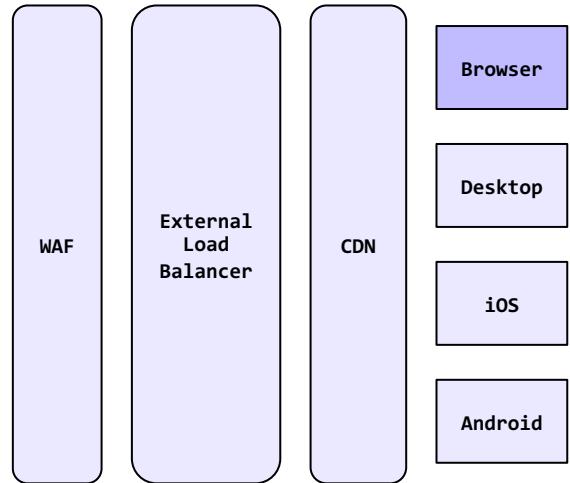
Network Components



Infrastructure Components



Network Components

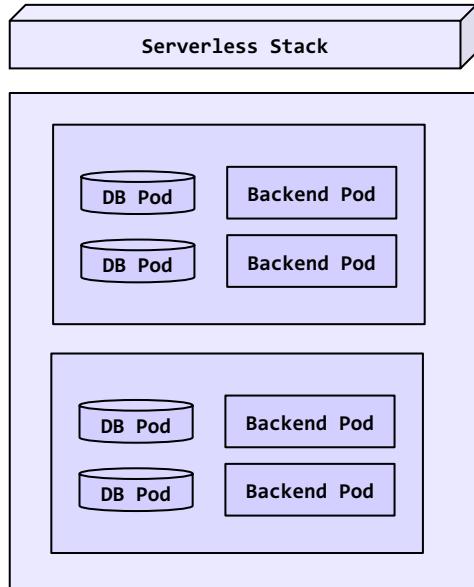


Clients

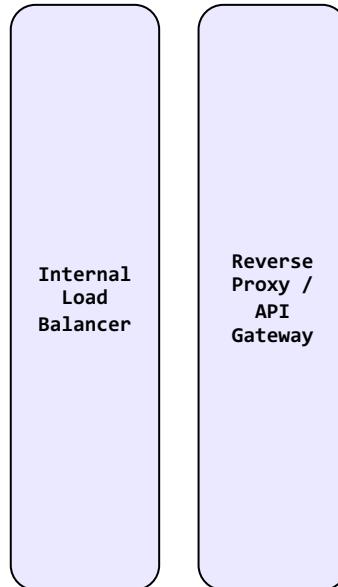


Infrastructure

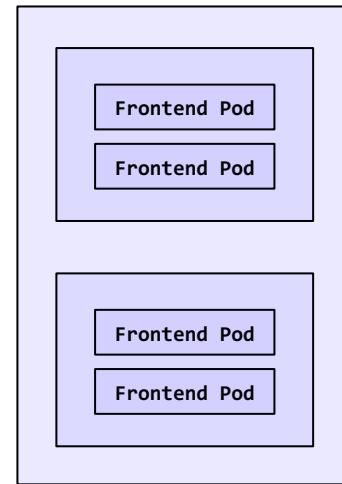
Infrastructure Components



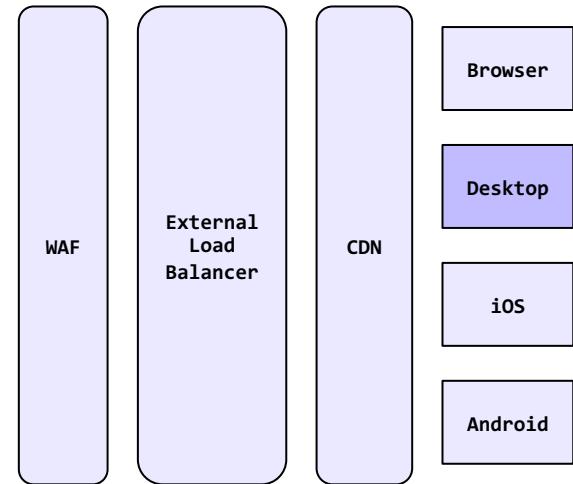
Network Components



Infrastructure Components



Network Components

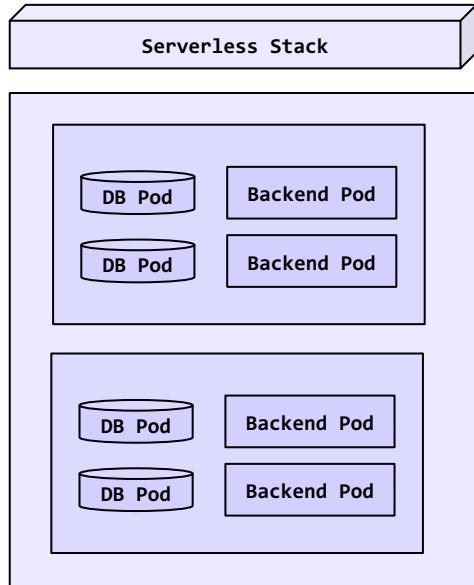


Clients

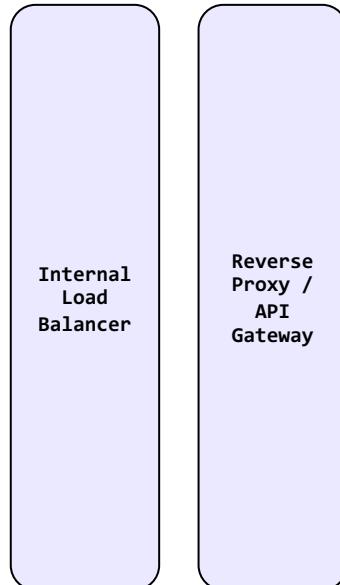


Infrastructure

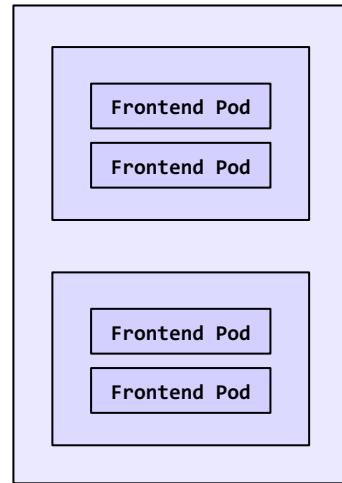
Infrastructure Components



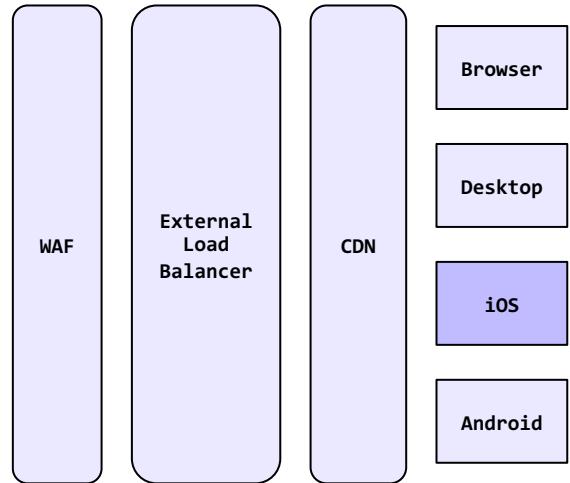
Network Components



Infrastructure Components

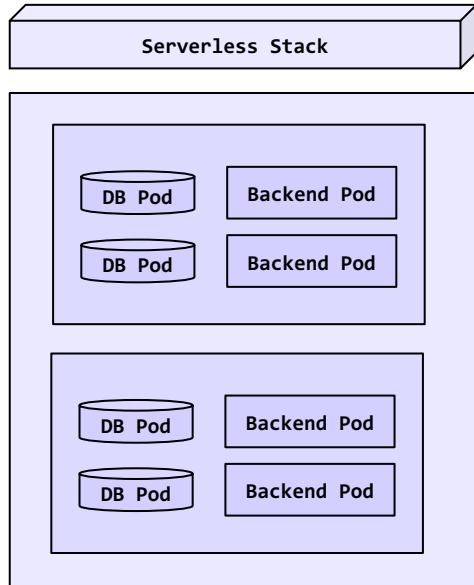


Network Components

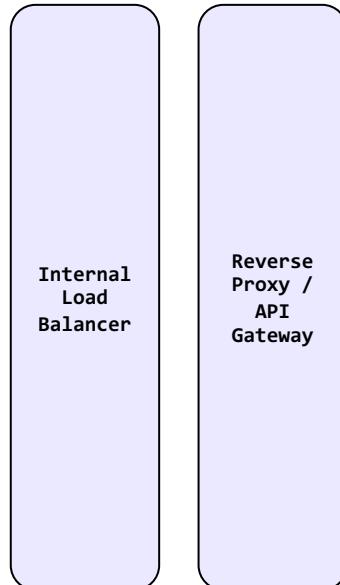


Infrastructure

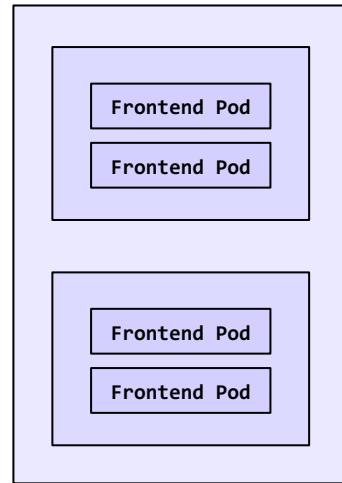
Infrastructure Components



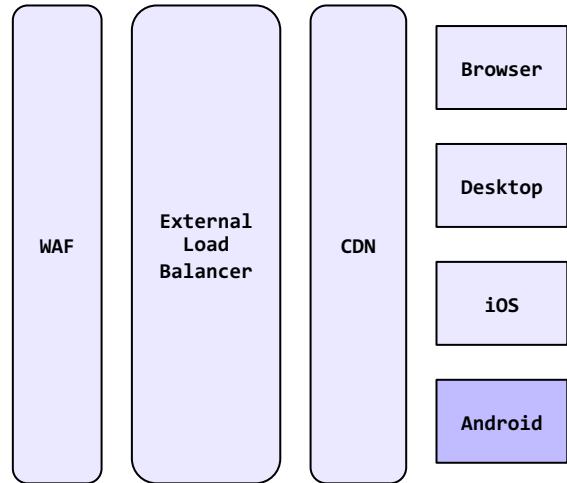
Network Components



Infrastructure Components

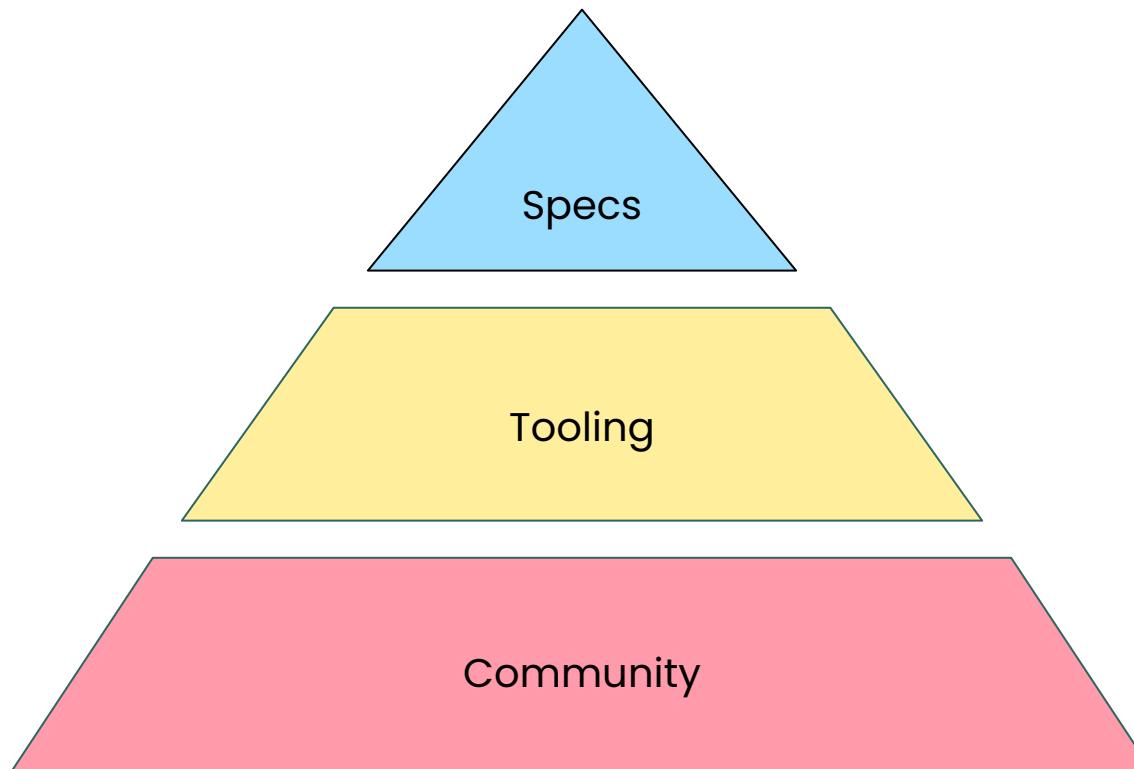


Network Components



Clients

Introducing OpenTelemetry



OpenTelemetry Community



Specs
Tooling
Community

Community Principles



1. Make building observable services easy

"To enable effective observability by making high-quality, portable telemetry ubiquitous."

Community Principles



1. Make building observable services easy

"To enable effective observability by making high-quality, portable telemetry ubiquitous."

2. Support engineers in learning Otel

Support engineers in understanding observability and the Otel standard through scalable methods.

Community Principles



1. Make building observable services easy

"To enable effective observability by making high-quality, portable telemetry ubiquitous."

2. Support engineers in learning Otel

Support engineers in understanding observability and the Otel standard through scalable methods.

3. Embed observability into your the culture

Reinforce the idea that observability is a crucial part of managing services.