Task 1:

a. Observability

- **The Three Pillars of Observability**

○ Logs
○ Metrics (Telemetry)
○ Traces

- **Blackbox vs. Whitebox monitoring**

White box monitoring is the monitoring of applications running on a server.
This could be anything from the number of HTTP requests your web server is getting to the response codes generated by your application.
Monitoring MySQL queries running on a database server
Looking at the number of users utilizing a web application throughout the day, and alerting if this goes above a predefined threshold.

Black box monitoring refers to the monitoring of servers with a focus on areas such as disk space, CPU usage, memory usage, load averages, etc. These are what most in the industry would deem as the standard system metrics to monitor.

We've moved away from implementing purely black box solutions to implementing white box solutions alongside them

- **Alerting**

Alerting is the responsive component of a monitoring system that performs actions based on changes in metric values. Alerts definitions are composed of two components: a metrics-based condition or threshold, and an action to perform when the values fall outside of the acceptable conditions.

Alerting based on monitoring data
Event Failures should immediately provide visibility of impact.
All alerts and signals that generate them should be actionable.

- **Monitoring Signals:**

○ USE Metrics
■ Free Memory (Utilization)
■ CPU queue length (Saturation)
■ Device errors (Errors)
○ RED Metrics
■ Request rate
■ Error rate
■ Duration of requests

○ Golden Signals (minimum viable signals)
■ Latency
■ Errors
■ Traffic
■ Saturation

- **Why do we need logs?**

○ Determining what happened - Audit trail
○ Intrusion Detection
○ Incident Containment
○ Forensic Analysis
○ Proactive Analysis
○ Real-time alerts
○ Determining the health of a network
■ Debugging/Trubbleshooting
■ Proactive maintenance

- **What is the difference between logs and metrics?**

A log is an event that happened and a metric is a measurement of the health of a system.

In essence, logs will tell the “story” for what happened in a system that got it to the issue you’re troubleshooting.

A log message is a system generated set of data when an event has happened to describe the event. In a log message is the log data. Log data are the details about the event such as a resource that was accessed, who accessed it, and the time. Each event in a system is going to have different sets of data in the message.

They specify five different general categories of logs: informational, debug, warning, error, or alert. Informational are typically “benign” events; debug are used during troubleshooting code or systems; warning are things like something may be missing but won’t directly impact the system; error are messages to convey a problem has occurred; and alert are that something important has happened and are largely around security use cases.

While logs are about a specific event, metrics are a measurement at a point in time for the system. This unit of measure can have the value, timestamp, and identifier of what that value applies to (like a source or a tag). Logs may be collected any time an event takes place, but metrics are typically collected at fixed-time intervals.

- **What is Tracing ?**

Tracing
○ A Trace is a representation of a services of causally related distributed events that represents the end-to-end request flow.
○ Identify a specific point in request flow.
○ Zipkin & Jaeger are the most popular OpenTacing-compliant open source

distributed tracing solutions

Distributed Tracing
○ Tracing Infrastructure attaches Contextual metadata to each requests.
○ Trace points in code, records events to be annotated.
○ Events are tagged with context and causality references generated by previous events.

b. Service Mesh

- **What is a Service Mesh?**

A service mesh is a configurable, low-latency infrastructure layer designed to handle a high volume of network-based interprocess communication among application infrastructure services using application programming interfaces (APIs). A service mesh ensures that communication among containerized and often ephemeral application infrastructure services is fast, reliable, and secure. The mesh provides critical capabilities including service discovery, load balancing, encryption, observability, traceability, authentication and authorization, and support for the circuit breaker pattern.

- **How does enable it's data plane?**

The part of a service mesh application that manages the network traffic between instances is called the data plane. Generating and deploying the configuration that controls the data plane's behavior is done using a separate control plane. The control plane typically includes, or is designed to connect to, an API, a command-line interface, and a graphical user interface for managing the app.

- **mTLS**

Mutual TLS is a common security practice that uses client TLS certificates to provide an additional layer of protection, allowing to cryptographically verify the client information.

In most cases when you try to access a secured HTTPS/TLS endpoint, you experience only the client-side check of the server certificate. The purpose of this check is to ensure that no fraud is involved and the data transfer between the client and server is encrypted. In fact, the TLS standard allows specifying the client certificate as well, so the server can accept connections only for clients with certificates registered with the server certificate authority, or provide additional security checks based on the information stored in the client certificate. This is what we call "Mutual TLS" - when both sides of the connection verify certificates.

- **Circuit Breaking**

Circuit breaking is an important pattern for creating resilient microservice applications. Circuit breaking allows you to write applications that limit the impact of failures, latency spikes, and other undesirable effects of network peculiarities.
Circuit breaking enforces limits on requests to a particular service, such as the maximum number of connections or maximum number requests after which requests are prevented from reaching the service to protect the service from degrading further.

```yaml
apiVersion: config.istio.io/v1alpha2
kind: DestinationPolicy
metadata:
  name: productpage-cb
  namespace: default
spec:
  destination:
    name: productpage
    labels:
      version: v1
  circuitBreaker:
    simpleCb:
      maxConnections: 1
      httpMaxRequests: 1
      httpMaxPendingRequests: 1
      httpMaxRequestsPerConnection: 1
      sleepWindow: 5m
      httpDetectionInterval: 1s
      httpConsecutiveErrors: 1
      httpMaxEjectionPercent: 100
```

- **Traffic splitting/steering**

Traffic splitting splits traffic based on weight. For example, you can send 99% of traffic to one service instance and 1% to another service instance. Traffic splitting is typically used for deploying new versions, A/B testing, service migration, and similar processes.
Traffic steering directs traffic to service instances based on content in HTTP request headers. For example, if a user's device is an Android device, with user-agent:Android in the request header, that traffic is sent to service instances designated to receive Android traffic, and traffic that does not have user-agent:Android is sent to instances that handle other devices.

- **Fault Injection**

Fault injection enables you to test the resiliency of services by simulating service failure scenarios, such as delays and aborted requests.

- **Rolling Update**

A rolling update is the process of updating an application — whether it is a new version or just updated configuration — in a serial fashion. By updating one instance at a time, you are able to keep the application up and running. If you were to just update all instances at the same time, your application would likely experience downtime. In addition, performing a rolling update allows you to catch errors during the process so that you can rollback before it affects all of your users.

c. Serverless

○ **What are some benefits and drawbacks of adopting serverless architecture?**

Benefits	Drawbacks
Reduced time to market and quicker software release. Lower operational and development costs.	Serverless is not efficient for long-running applications. In certain cases, using long tasks can be much more expensive than,

A smaller cost to scale – there is no need for developers to implement code to scale and administrators do not need to upgrade existing servers or add additional ones. Works with agile development and allows developers to focus on code and to deliver faster. Fits with microservices, which can be implemented as functions. Reduces the complexity of software. Simplifies packaging and deployment and requires no system administration.	for example, running a workload on a dedicated server or virtual machine. Vendor lock-in. Your application is completely dependent on a third-party provider. You do not have a full control of your application. Most likely, you cannot change your platform or provider without making significant changes to your application. Also, you are dependent on platform availability, and the platform's API and costs can change. Needles to say, the existing FaaS implementations are not compatible with each other. Serverless (and microservice) architectures introduce additional overhead for function/microservice calls. There are no "local" operations; you cannot assume that two communicating functions are located on the same server. To utilize its resources more efficiently, a service provider may run software for several different customers on the same physical server (this is also known as "multitenancy"). Even if customers' workloads are isolated in virtual machines or containers, there can be different bugs in the early stages of FaaS offerings. This can be a major security issue for your application if you work with sensitive data. Such potential bugs in a platform or failures in one customer's code (a "noisy neighbor") can affect the performance or availability of your application. In practice, it takes some time for a scalable serverless platform to handle a first request by your function. This problem is know as "cold start"—a platform needs to initialize internal resources (AWS Lambda, for example, needs to start a container). The platform may also release such resources (such as stopping the container) if there have been no requests to your function for a long time. One option to avoid the cold start is to make sure your function remains in an active state by sending periodic requests to your function. Some FaaS implementations—such as AWS Lambda—do not provide out-of-the-box tools to test functions locally (assuming that a developer will use the

	same cloud for testing). This is not an ideal decision, especially considering that you will pay for each function invocation. As a result, several third-party solutions are trying to fill this gap, enabling you to test functions locally. Different FaaS implementations provide different methods for logging in functions. For example, AWS Lambda allows writing logs to AWS CloudWatch using the standard frameworks for the specific language (Log4j for Java, console methods for Node.js, logging module for Python). It is completely up to the developer how to implement more advanced logging.

Task 2 : Declare how the following solutions solves the following challenges.

a. Monitoring

- **Grafana**

Grafana is an open-source platform for data visualization, monitoring and analysis. Grafana allows users to create dashboards with panels, each representing specific metrics over a set time-frame. Every dashboard is versatile, so it could be custom-tailored for a specific project or any development and/or business needs.

Grafana Notions
A Panel is the basic visualization building block presented per the metrics selected. Grafana supports graph, singlestat, table, heatmap, and freetext panels, as well as integration with official and community-built plugins (like world map or clock) and apps that could be visualized, too. Each panel can be customized in terms of style and format; all panels could be dragged, dropped, resized, and rearranged.

A Dashboard is a set of individual panels arranged on a grid with a set of variables (like server, application and sensor name). By changing variables, you can switch the data being displayed in a dashboard (for instance, data from two separate servers). All dashboards could be customized and sliced and diced depending on the user needs. Grafana has a large community of contributors and users, so there is a large ecosystem of ready-made dashboards for different data types and sources.

Dashboards can utilize annotations to display certain events across panels. An annotation is added by custom requests to Elasticsearch; it shows as a vertical red line on the graph.

When hovering over an annotation, you can get event description and tags, for instance, to track when server responds with 5xx error code or when the system restarts. This way, it is easy to correlate with a time, specific event and its consequences in an application and investigate system behaviour.

- **Prometheus**

Prometheus is a free software application used for event monitoring and alerting. It records real-time metrics in a time series database (allowing for high dimensionality) built using a HTTP pull model, with flexible queries and real-time alerting. The project is written in Go and licensed under the Apache 2 License, with source code available on GitHub, and is a graduated project of the Cloud Native Computing Foundation, along with Kubernetes and Envoy.

A typical monitoring platform with Prometheus is composed of multiple tools:

+ Multiple exporters that typically run on the monitored host to export local metrics.
+ Prometheus to centralize and store the metrics.
+ Alertmanager to trigger alerts based on those metrics.
+ Grafana to produce dashboards.
+ PromQL is the query language used to create dashboards and alerts.

- **Sentry**

Sentry provides self-hosted and cloud-based error monitoring that helps all software teams discover, triage, and prioritize errors in real-time.
Sentry provides:

+ Context, uncovered: Source code, error filters, stack locals — Sentry enhances application performance monitoring with stack traces.
+ All Issues, one place: See all Issues across your entire organization or select a handful of projects to surface correlated trouble spots.
+ Trail of events, discovered: Breadcrumbs make application development a little easier by showing you the trails of events that lead to the error(s).
+ Version changes, highlighted: Whether you're using JavaScript, PHP, or anything in between, Releases provide visibility to which errors were addressed and which were introduced for the first time.
+ Control, given: The software development cycle can be riddled with ambiguity. Issue Owners put control back in the hands of developers to fix what's broken in their code.
+ Queries, customized: Real-time monitoring means data, in real-time. Query raw event data across your organization with Discover, Sentry's query builder.
+ Data, visualized: Dashboards add a visual element to our application monitoring.

- **Kibana:**

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. You use Kibana to search, view, and interact with data stored in Elasticsearch

indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps.

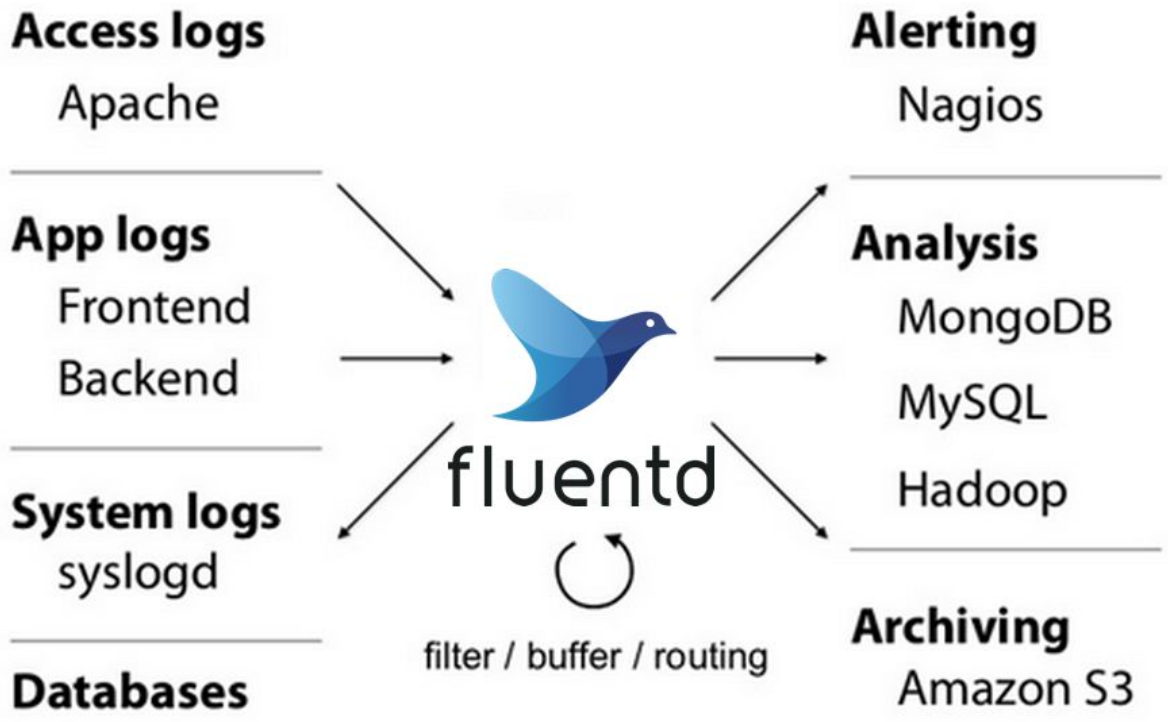
Kibana makes it easy to understand large volumes of data. Its simple, browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time.

Setting up Kibana is a snap. You can install Kibana and start exploring your Elasticsearch indices in minutes — no code, no additional infrastructure required.

b. Logging

- **Fluentd**

Fluentd is an open source data collector for building the unified logging layer. Once installed on a server, it runs in the background to collect, parse, transform, analyze and store various types of data.



Fluentd treats logs as JSON, a popular machine-readable format. It is written primarily in C with a thin-Ruby wrapper that gives users flexibility.
Fluentd has some characteristics below:

+ Unified Logging with JSON: Fluentd tries to structure data as JSON as much as possible: this allows Fluentd to unify all facets of processing log data: collecting, filtering, buffering, and outputting logs across multiple sources and destinations (Unified Logging Layer). The downstream data processing is much easier with JSON, since it has enough structure to be accessible while retaining flexible schemas.

+ Pluggable Architecture: Fluentd has a flexible plugin system that allows the community to extend its functionality. Our 500+ community-contributed plugins connect dozens of data sources and data outputs. By leveraging the plugins, you can start making better use of your logs right away.
+ Minimum Resources Required: Fluentd is written in a combination of C language and Ruby, and requires very little system resource. The vanilla instance runs on 30-40MB of memory and can process 13,000 events/second/core. If you have tighter memory requirements (-450kb), check out Fluent Bit, the lightweight forwarder for Fluentd.
+ Built-in Reliability: Fluentd supports memory- and file-based buffering to prevent inter-node data loss. Fluentd also supports robust failover and can be set up for high availability. 2,000+ data-driven companies rely on Fluentd to differentiate their products and services through a better use and understanding of their log data.

- **Elasticsearch and Logstash**

Elasticsearch:

Elasticsearch is a distributed, open source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Elasticsearch is built on Apache Lucene and was first released in 2010 by Elasticsearch N.V. (now known as Elastic). Known for its simple REST APIs, distributed nature, speed, and scalability, Elasticsearch is the central component of the Elastic Stack, a set of open source tools for data ingestion, enrichment, storage, analysis, and visualization. Commonly referred to as the ELK Stack (after Elasticsearch, Logstash, and Kibana), the Elastic Stack now includes a rich collection of lightweight shipping agents known as Beats for sending data to Elasticsearch.

It can be used for a number of use cases:

+ Application search
+ Website search
+ Enterprise search
+ Logging and log analytics
+ Infrastructure metrics and container monitoring
+ Application performance monitoring
+ Geospatial data analysis and visualization
+ Security analytics
+ Business analytics

How does Elasticsearch work:
Raw data flows into Elasticsearch from a variety of sources, including logs, system metrics, and web applications. Data ingestion is the process by which this raw data is parsed, normalized, and enriched before it is indexed in Elasticsearch. Once indexed in Elasticsearch, users can run complex queries against their data and use aggregations to retrieve complex summaries of their data. From Kibana, users can create powerful visualizations of their data, share dashboards, and manage the Elastic Stack

Why use Elasticsearch:
Elasticsearch is fast. Because Elasticsearch is built on top of Lucene, it excels at full-text search. Elasticsearch is also a near real-time search platform, meaning the latency from the time a document is indexed until it becomes searchable is very short — typically one second. As a result, Elasticsearch is well suited for time-sensitive use cases such as security analytics and infrastructure monitoring.

Elasticsearch is distributed by nature. The documents stored in Elasticsearch are distributed across different containers known as shards, which are duplicated to provide redundant copies of the data in case of hardware failure. The distributed nature of Elasticsearch allows it to scale out to hundreds (or even thousands) of servers and handle petabytes of data.

Elasticsearch comes with a wide set of features. In addition to its speed, scalability, and resiliency, Elasticsearch has a number of powerful built-in features that make storing and searching data even more efficient, such as data rollups and index lifecycle management.

The Elastic Stack simplifies data ingest, visualization, and reporting. Integration with Beats and Logstash makes it easy to process data before indexing into Elasticsearch. And Kibana provides real-time visualization of Elasticsearch data as well as UIs for quickly accessing application performance monitoring (APM), logs, and infrastructure metrics data.

Logstash:

Logstash, one of the core products of the Elastic Stack, is used to aggregate and process data and send it to Elasticsearch. Logstash is an open source, server-side data processing pipeline that enables you to ingest data from multiple sources simultaneously and enrich and transform it before it is indexed into Elasticsearch.

c. Tracing

- **Jaeger**

Jaeger, inspired by Dapper and OpenZipkin, is a distributed tracing system released as open source by Uber Technologies. It is used for monitoring and troubleshooting microservices-based distributed systems, including:

+ Distributed context propagation
+ Distributed transaction monitoring
+ Root cause analysis
+ Service dependency analysis
+ Performance / latency optimization

Features:

+ OpenTracing compatible data model and instrumentation libraries in Go, Java, Node, Python and C++
+ Uses consistent upfront sampling with individual per service/endpoint probabilities
+ Multiple storage backends: Cassandra, Elasticsearch, memory.
+ Adaptive sampling (coming soon)

+ Post-collection data processing pipeline (coming soon)

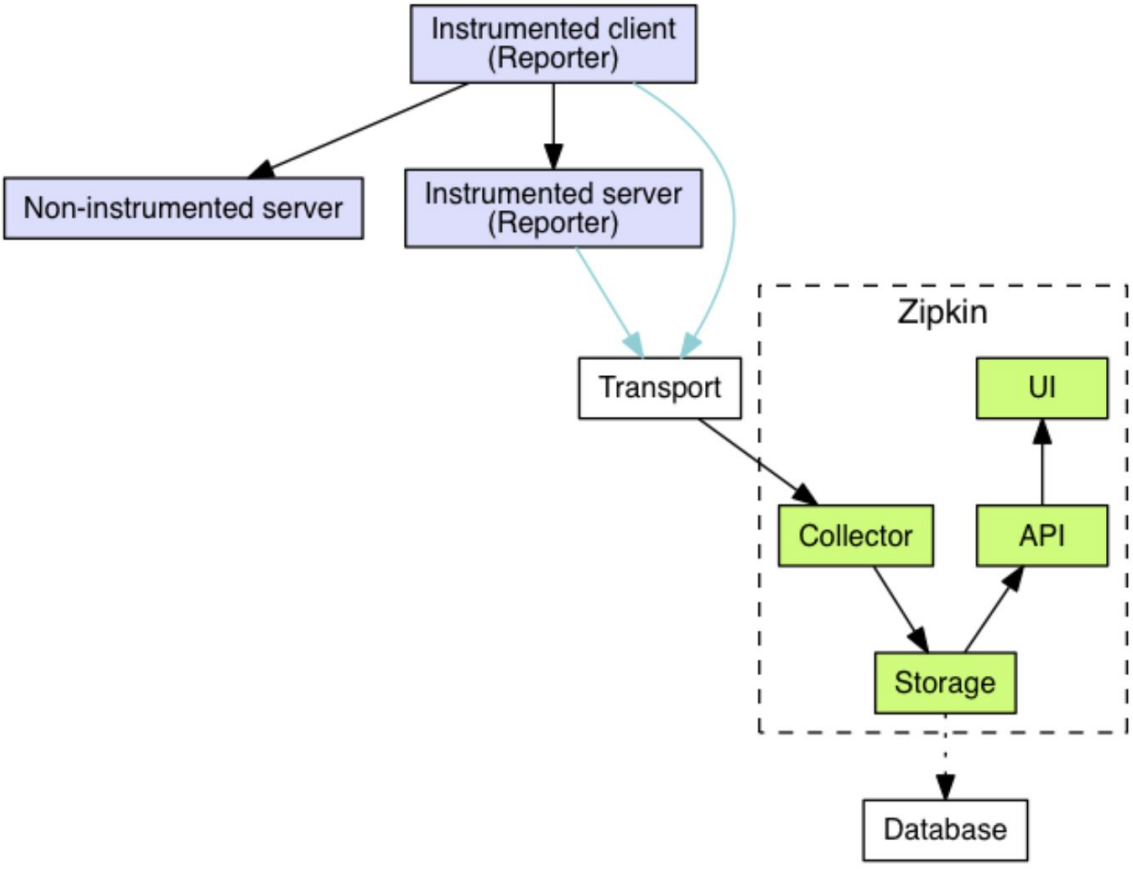
Technical Specs

+ Backend components implemented in Go
+ React/Javascript UI
+ Supported storage backends:
+ Cassandra 3.4+
+ Elasticsearch 5.x, 6.x
+ Kafka
+ memory storage

- **Zipkin:**

Zipkin was one of the first tracing systems developed according to Google's Dapper, and its architecture is quite simple. Every operation performed in the application that is being traced is usually generated on the client-side and starts with an outgoing http request. Several headers are added to this request in order to create unique IDs that are traceable along with the application. The Reporter is the component that is implemented inside each host of the tracked application and is in charge of sending the data to Zipkin. Reporters then send all traces to Zipkin collectors, which persist the data to databases.

You can use either Cassandra or Elasticsearch for a scalable storage backend, as Zipkin supports them both. This storage is queried by the API used by the Zipkin UI.



The Zipkin architecture. Source: zipkin.io

Zipkin supports every programming language out there, with dedicated libraries–with native OpenTracing instrumentation support or extension points for tracing capabilities–for almost all of their frameworks: C, C++, C#, Java, Javascript, Python, Scala, Go, and more. It also comes with native support for the major cloud providers: AWS, Azure, and Google Cloud.

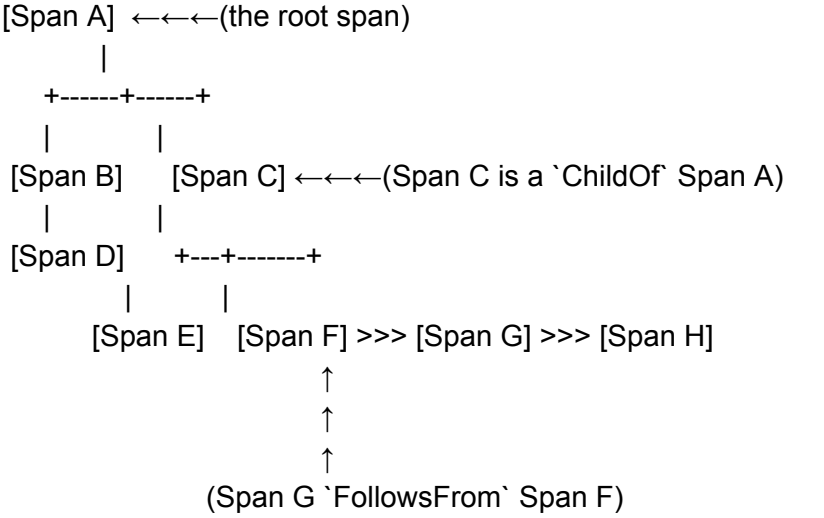
In addition, Zipkin has an open-source community, OpenZipkin, that continuously publishes new APIs, data formats, and libraries. This allows users to replace their existing Zipkin backend to process the same type of data sent to the regular Zipkin server.

- **OpenTracing:**

Distributed tracing, also called distributed request tracing, is a method used to profile and monitor applications, especially those built using a microservices architecture. Distributed tracing helps pinpoint where failures occur and what causes poor performance.

OpenTracing is not a download or a program. Distributed tracing requires that software developers add instrumentation to the code of an application, or to the frameworks used in the application. OpenTracing is not a standard. The Cloud Native Computing Foundation (CNCF) is not an official standards body. The OpenTracing API project is working towards creating more standardized APIs and instrumentation for distributed tracing.

OpenTracing is comprised of an API specification, frameworks and libraries that have implemented the specification, and documentation for the project. OpenTracing allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.



This allows us to model how our application calls out to other applications, internal functions, asynchronous jobs, etc. All of these can be modeled as Spans, as we'll see below.

For example, if I have a consumer website where a customer places orders, I make a call to my payment system and my inventory system before asynchronously acknowledging the order. I can trace the entire order process through every system with an OpenTracing library and can render it like this:

```
—|———————|———————|———————|———————|———————|———————|———————|–> time
 [Place Order·····································]
   [Receive Payment·····] [Fulfill Order··] [Email Order…]
```

d. Observability Stacks

- **Istio**

Cloud platforms provide a wealth of benefits for the organizations that use them. However, there's no denying that adopting the cloud can put strains on DevOps teams. Developers must use microservices to architect for portability, meanwhile operators are managing extremely large hybrid and multi-cloud deployments. Istio lets you connect, secure, control, and observe services.

At a high level, Istio helps reduce the complexity of these deployments, and eases the strain on your development teams. It is a completely open source service mesh that layers transparently onto existing distributed applications. It is also a platform, including APIs that let it integrate into any logging platform, or telemetry or policy system. Istio's diverse feature set lets you successfully, and efficiently, run a distributed microservice architecture, and provides a uniform way to secure, connect, and monitor microservices.

Core features:

+ Traffic management

Istio's easy rules configuration and traffic routing lets you control the flow of traffic and API calls between services. Istio simplifies configuration of service-level properties like circuit breakers, timeouts, and retries, and makes it a breeze to set up important tasks like A/B testing, canary rollouts, and staged rollouts with percentage-based traffic splits.

With better visibility into your traffic, and out-of-box failure recovery features, you can catch issues before they cause problems, making calls more reliable, and your network more robust – no matter what conditions you face.

+ Security

Istio's security capabilities free developers to focus on security at the application level. Istio provides the underlying secure communication channel, and manages authentication, authorization, and encryption of service communication at scale. With Istio, service communications are secured by default, letting you enforce policies consistently across diverse protocols and runtimes – all with little or no application changes.

While Istio is platform independent, using it with Kubernetes (or infrastructure) network policies, the benefits are even greater, including the ability to secure pod-to-pod or service-to-service communication at the network and application layers.

Refer to the Security concepts guide for more details.

+ Policies

Istio lets you configure custom policies for your application to enforce rules at runtime such as:
Rate limiting to dynamically limit the traffic to a service
Denials, whitelists, and blacklists, to restrict access to services
Header rewrites and redirects
Istio also lets you create your own policy adapters to add, for example, your own custom authorization behavior.

+ Observability

Istio's robust tracing, monitoring, and logging features give you deep insights into your service mesh deployment. Gain a real understanding of how service performance impacts things upstream and downstream with Istio's monitoring features, while its custom dashboards provide visibility into the performance of all your services and let you see how that performance is affecting your other processes.

Istio's Mixer component is responsible for policy controls and telemetry collection. It provides backend abstraction and intermediation, insulating the rest of Istio from the implementation details of individual infrastructure backends, and giving operators fine-grained control over all interactions between the mesh and infrastructure backends.

All these features let you more effectively set, monitor, and enforce SLOs on services. Of course, the bottom line is that you can detect and fix issues quickly and efficiently.

+ Platform support

Istio is platform-independent and designed to run in a variety of environments, including those spanning Cloud, on-premise, Kubernetes, Mesos, and more. You can deploy Istio on Kubernetes, or on Nomad with Consul. Istio currently supports:

Service deployment on Kubernetes
Services registered with Consul
Services running on individual virtual machines

+ Integration and customization

The policy enforcement component of Istio can be extended and customized to integrate with existing solutions for ACLs, logging, monitoring, quotas, auditing, and more.

- **Elastic Stack**

Elastic Stack is a group of open source products from Elastic designed to help users take data from any type of source and in any format and search, analyze, and visualize that data in real time. The product group was formerly known as ELK Stack, in which the letters in the name stood for the products in the group: Elasticsearch, Logstash and Kibana. A fourth product, Beats, was subsequently added to the stack, rendering the potential acronym

unpronounceable. Elastic Stack can be deployed on premises or made available as Software as a Service (SaaS).

Elastic Stack components:

+ Elasticsearch is a RESTful distributed search engine built on top of Apache Lucene and released under an Apache license. It is Java-based and can search and index document files in diverse formats.
+ Logstash is a data collection engine that unifies data from disparate sources, normalizes it and distributes it. The product was originally optimized for log data but has expanded the scope to take data from all sources.
+ Beats are “data shippers” that are installed on servers as agents used to send different types of operational data to Elasticsearch either directly or through Logstash, where the data might be enhanced or archived.
+ Kibana is an open source data visualization and exploration tool from that is specialized for large volumes of streaming and real-time data. The software makes huge and complex data streams more easily and quickly understandable through graphic representation.

Elastic Stack presents a steeper learning curve than some comparable products, as well as more set up, owing in part to its open source nature. In return for the extra work, however, the sysadmin is rewarded with a deeper understanding of the software’s underlying structure.

Elastic was founded in Amsterdam in 2012 to support the development of Elasticsearch and related commercial products and services.