International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2020): 7.803

The Role of Observability in Modern Software Development Lifecycle

Amreth Chandrasehar

Abstract: ***Observability, basically meaning "better monitoring" is fundamental for applications to perform optimally while deployed in any environment. The modern applications use distributed architecture, which makes it complex for the developers, SRE and DevOps engineers to debug, analyze, isolate and fix issues. Enabling Observability from development to production helps the better quality, understand how systems behave in different scenarios, reliability, discoverability, controllability and performance of modern applications. This paper focus on enabling Observability in software development lifecycle, compares statistics on developer and operations productivity before and after enabling Observability in applications.***

Keywords: Observability, SRE, Software Development, SDLC, Productivity

1. Introduction

Today's systems are more and more complex, with microservices being distributed over the network and scaling dynamically, resulting in many more ways of failure, ways we can't always predict. Believing that the perfect systemcan be built, will lead to a false sense of security. Investing in observability gives the ability to ask questions to the systems, things never thought about before. Some of the tools that can be used for this are metrics, tracing, structured and correlated logging. [1]

Observability is about data exposure and easy access to information which is critical when you need a way to see when communications fail, do not occur as expected or occur when they shouldn't. The way services interact with each other at runtime needs to be monitored, managed and controlled. This begins with observability and the ability to understand the behavior of your microservice architecture. [2]

Observability is the level of visibility that the system grants to an outside observer. It's a property of a system, just like usability, availability, and scalability. Monitoring is for operating software/systems Instrumentation is for writing software Observability is for understanding systems Investing in observability means to be prepared to spend the time on instrumenting systems, cope for the unknowns that come in production. It can be very simple at the start, such as some basic health - checks. Withmetrics, tracing, logging, correlations, structured logging, events; combined together it just brings a really powerful solution. When things may go wrong, Observability helps to react and recover faster.

2. Basics of Observability

Observability is based on three main pillars: logging, metrics, and traces.

- Metrics: Metrics are quantitative measurements of the state of a system and used to track CPU utilization, memory usage, disk space usage, and response times.
- Logs: Logs are textual records of events that occur in a system and used to troubleshoot problems, identify performance bottlenecks, and audit system activity.
- Traces: Traces are records of the interactions between different components of a system and used to identify root cause of issues in distributed systems.

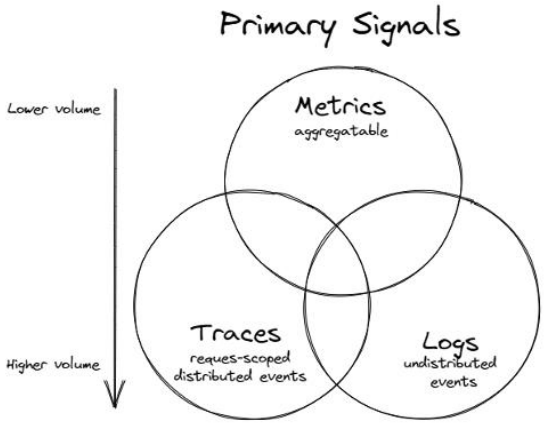

Figure [1]: Logs, Metrics and Traces – 3 Pillars of Observability

Metrics are very efficient at aggregating data over low - granularity dimensions, which makes them a good choice for service level monitoring type of things. However, metrics don't scale well at high cardinality, which is what is needed for debugging and exploration.

Good logs, especially structured logs, play a huge role in increasing our understanding of how applications behave. Logs provides sufficient context to understand how events occurred. Correlation of logs is also one of the first things to incorporate with Request Id following a request through the entire system. Logs can be expensive, logging libraries have often non - trivial overhead, it can be challenging to manage the volume of data, and sampling at logs level is not easy. Enabling correlations using Jager or Zipkin makes it easier to navigate from logs to traces. [3]

Distributed tracing is great for debugging and understanding application's behavior. The key to making sense of all the tracing data is being able to correlate spans from different microservices which are related to a single client request. To achieve this, all microservices in your application should propagate tracing headers.

Most failures in the microservices space occur during the interactions between services, so a view into the transactions will help teams better manage architectures to avoid failures. Observability provided by a service mesh makes it much easier to see what is happening when your services interact with each other, making it easier to build a more efficient, resilient and secure microservice architecture. To put microservices monitoring and observability to a next level and bring the era of the next APM tools, an open, vendor - neutral instrumentation standard would be needed like OpenTracing. This new standard needs to be applied by APM vendors, service providers, and open - source library maintainers as well.

3. Enabling Observability in software applications

To enable Observability, the organization needs to create a central team that is responsible to implement below tasks:

- SDK/ Frameworks to be embedded in the application
- Defining Logging Standards, each application team needs to adhere to
- Enabling APM and Tracing on all services in applications
- Emit, ensure metrics are discoverable to becollected by the agents or tools
- Correlate logs, metrics, traces and events with tags
- Enable Machine Learning models to predict and forecast a problem to avoid an incident
- Continuously asses the implementation and automate process

Some of the challenges organizations face while implementing Observability are:

- Using multiple tools, which creates isolation of data preventing correlation of different datasets. This causes high MTTD, MTTR resulting to more and prolonged incidents.
- Detecting outages manually or from customers – This occurs when data collected is not useful or alerts are not configured, or the on - call support team are overwhelmed due to flood of alerts and lack of proper System Operating Procedures (SOP).
- No central Observability team managing the implementations, leading multiple nonstandard, extreme troubleshooting, siloed data storage, siloed technologies, leading to higher incidents.
- Lack of staffing and skills
- Lack of operations and security process

While challenges exist, the implementers can use below opportunities and tips toimplement observability:

- Collect telemetry data from all the components in the system such as applications, infrastructure, managed services, developer platforms
- Choose an observability platform that supports the telemetry data generated by the applications
- Configure the observability platform to collect and store telemetry data efficiently
- Develop dashboards and alerts to visualize and analyze the collected data.
- Provide training to teams on how to use the observability platform.
- Collect Mean - time metrics, these are the MTTx metrics.
- Implement single, consolidated Observability platform

Observability of a system is not an after - thought after the deployment is done. If not designed, at that level of the deployment lifecycle, successful observability will be by accident, if it’s achievable at all. Without proper observability, enforcing process based, codified desired state is accidental at best. What enables successful observability for a given system are a collection of what already existing different practices encompass. These include monitoring, tracing, log aggregation, and alerting that produce various metrics that provides a view into server performance, service availability, and capacity consumption. Well designed and holistic approach to managing these insights results in an observable system with no blind spots.

4. Collection and storage of Observability data

Once observability has been implemented, data needs to be collected from various sources and stored in a central database or data lake for the data to be accessible to query, building dashboards and creating alerts. Below is the high - level architecture of implementing end to end Observability platform in an organization.

The observability agents need to be deployed on Virtual Machines, Containers, Databases, Managed Services, Vendors, etc and sent to a streaming service (example Kafka). A streaming service helps to collect, store Observability data before the data is pulled by the platform for further analysis. The Observability platform helps to create queries to debug issues, create dashboards and also to apply Machine learning models to predict and forecast issues. Alerts are also created in the platform before being sent to on - call service platforms.

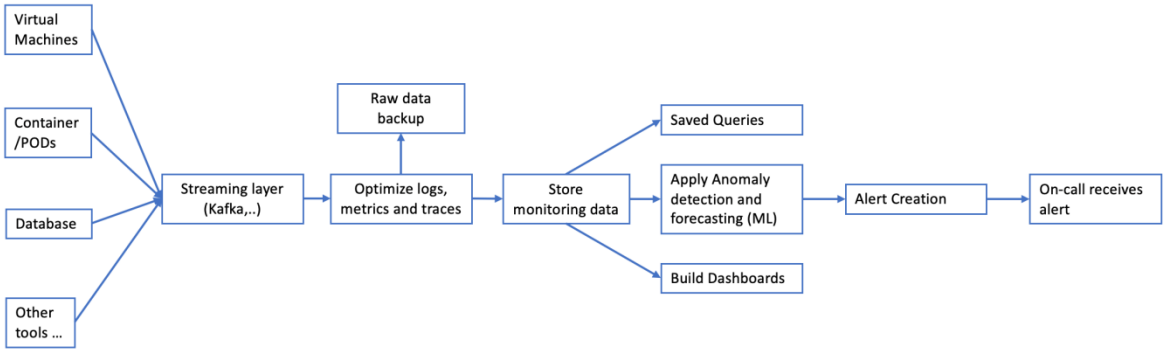

Figure [2]: End to end Observability Platform

5. Using Observability data in Software development lifecycle

Enterprises adopt Observability to avoid downtimes, innovate, increase agility and confidence in organizations and for reliable and faster development.

Reducing manual changes and automating as much as possible is a good practice. Observability as code can be achieved using any templates, SDKs, frameworks of tools that helps to automate the configuration and management of Observability assets such as dashboards, alerts, resources, upgrade, platform changes.

Observability - driven development (ODD) adds another layer to software development by encouraging the development team to think about the application availability and uptime throughout the development process and like unit - testing development, wrap the code with more verbose logging, metrics and KPIs. This gives the IT operations team more data on the application and improves overall observability, allowing an organization to detect predictable and unpredictable permutations of failures that may occur in the future. ODD observes the behavior of a system throughout the entire development cycle in order to learn what behavior is normal and abnormal, and to detect potential weaknesses. [7]

Observability provides everything with a real - time status of a product–this includes software developers, DevOps and any other department that requires it. Each role has different needs. Software developers need to know why a certain feature must be added or developed in order to understand its business value and prioritize it over other features or updates. If a certain feature is likely to require many future updates, this also needs to be taken into consideration in the development phase so that the code can be adjusted easily without negatively affecting DevOps teams when they test and deploy the feature. These are just examples of things that must be taken into consideration during the development process. Organizations must be able to create transparency and effective communication between the different departments and roles. [7]

A strong proactive observability platform can predict and address issues before they occur, thus increasing effectiveness and speed when it comes to updating and tracking changes, as well as releasing new features. It simplifies complexities and management and allows to automate operations so that organizations can maximize efficiency.

Benefits of ODD include:

- Common goals.
- Shared metrics.
- Execute on goals and recognize results.
- Performance guidelines.
- Application governance strategy.
- Alignment between IT and business perspectives.

Employing ODD when developing cloud applications is critical. This is due to the highly distributed nature of cloud architectures, increasing the chances of a failure exponentially compared to traditional systems. As a result, developers should be educated to think and implement observability to create highly available and resilient

To get started with ODD, OpenTelemetry is a complete solution that solves the problem of collecting telemetry metrics. Its mission is to develop an open, industry - wide standard for telemetry data, and to provide reference implementations with universal tools that support metrics, tracing, and logs. OpenTelemetry currently supports Python, Golang, JavaScript, Erlang, Java,. NET, PHP, Rust, C++, Ruby, and Swift.

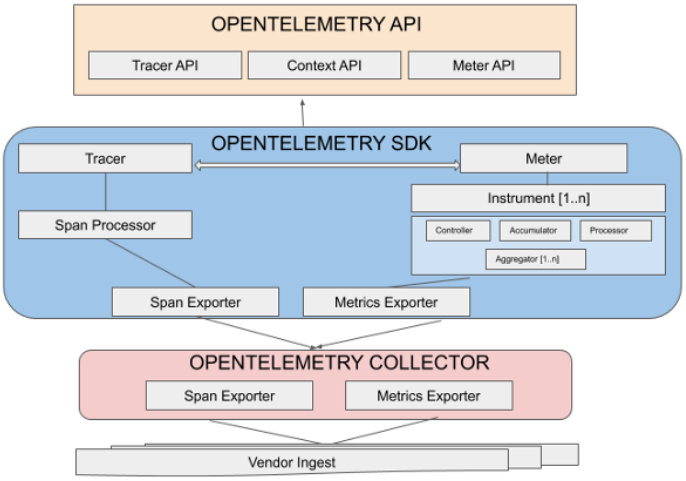

Figure [3]: Open Telemetry SDK, API and Collector architecture

The OpenTelemetry specification demands an API and a SDK for its metrics architecture. The API defines how to capture metric data, while the SDK processes, queries, and exports it. A user can inject our API elements into their application with no compilation issues; however, the API on its own will not be able to generate any useful metric data. After a user installs the SDK, the library aggregates, filters, and distributes API - captured data to any number of visualization backend services. [6]

The API consists of three major components: metric instruments, meter, and meter provider classes. Metric instruments are what a user injects into their code at strategic locations to capture data of interest. The meter is responsible for creating these instruments and managing them in an internal registry. The meter also provides a single endpoint for collecting data from all operational metric instruments. Finally, the meter provider creates a global meter instance and allows users to specify certain aspects of the pipeline. [6]

```
// Retrieve a counter metric instrument from the active meter
std::shared_ptr<Counter<int>> ctr = meter.NewIntCounter("test", "description", "units", true);

// Initialize a list of labels which describe the capture
std::map<std::string, std::string> labels = {{"key", "value"}};
auto labelkv = trace::KeyValueIterableView<decltype(labels)>{labels};

// Update the instrument with the value 57
ctr->add(57, labels);

// In some cases, labels may be constant. We can bind an instrument to a set
// of labels and they will be associated with all future value captures automatically.
auto boundctr = ctr->bindCounter(labelkv);
boundctr->add(46);
boundctr->add(57);
boundctr->unbind();
```

Figure [4]: Instrumenting code and capturing labeled data

6. Measuring impact with and without Observability in Software Development

Implementing Observability helps to reduce outages, faster MTTD, faster MTTR and cost savings from incidents and customer complaints.

Below are the metrics that can help track the impacts. The data is collected over 3 - year period from incidents in organizations. All the data given are an average of incidents with and without Observability.

S. No	Metric	Without Observability	With Observability	Impact
1	MTTR	5 hours 33 minutes	42 minutes	87.4% reduction
2	Total Number of incidents	234	121	48.3 % reduction
3	Customer reported or manually detected incidents	77	12	84.4 $ reduction
4	Number of incidents resolved without customer impact or downtime	133	53	40%
5	MTTD	56 minutes	6 minutes	89.3%
6	Uptime/ Service availability	99.9%	99.95%	0.05%
7	Incident Response Speed	5 minutes	4 minutes	20%

7. Future Work

Observability helps in many aspects in an organization. More analysis needs to be done to capture Customer Satisfaction and Reduction in Churn metrics, false positives in alerts and the 5 pillars of data observability.

8. Conclusion

Observability is key to a successful operation of applications. It enables customers and developers of applications to get more insights into usage patterns, debugging data and maintain higher SLAs. With modern software development becoming more complex with distributed systems, applications hosted in different clouds and regions, it should be mandatory to incorporate Observability from the initial stages of development and take it to Production.

References

[1] https: //www.infoq. com/news/2018/06/observable - distributed - systems/
[2] https: //medium. com/[at]chamilad/a - primer - on - observability - for - dynamic - organizations - part - 1 - 43c577ded1d3
[3] https: //www.infoq. com/news/2018/06/observable - distributed - systems/
[4] https: //www.riverbed. com/blogs/it - performance - management - observability/
[5] https: //www.honeycomb. io/blog/forrester - tei - benefits - observability - roi - 2021
[6] https: //aws. amazon. com/blogs/opensource/aws - adds - observability - metrics - to - the - opentelemetry - c - library/
[7] https: //devops. com/observability - driven - development - from - software - development - to - devops - and - beyond/