Operating System

Week-3

Type your text

Type your text

Type your text

Type your text

Lecturer: Meesam Raza

Type your text

Type your text

Type your text

Type your text

Type your text

Type your text

Type your text

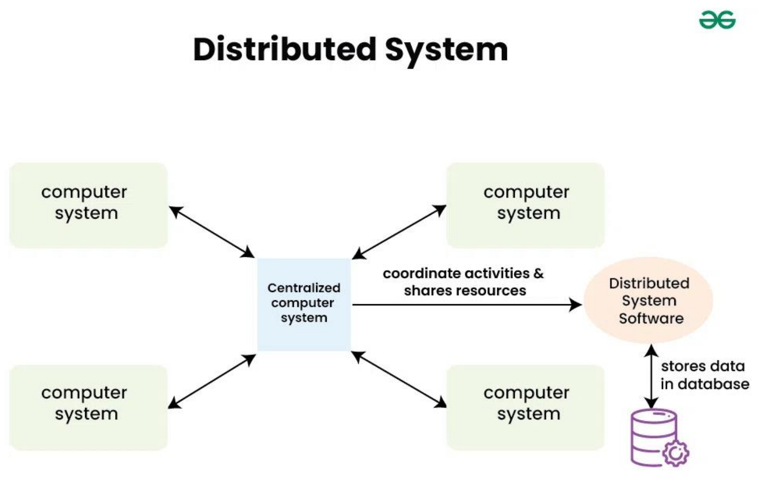
Type your text

Type your text

Type your text

Distributed System

A distributed system is simply any environment where multiple computers or devices are working on a variety of tasks and components, all spread across a network. Components within distributed systems split up the work, coordinating efforts to complete a given job more efficiently than if only a single device ran it.



Today, all types of computing jobs — from database management to video games — use distributed computing. In fact, many types of software, such as cryptocurrency systems, scientific simulations, blockchain technologies and AI platforms, wouldn't be possible at all without these platforms.
Distributed systems are used when a workload is too great for a single computer or device to handle.
Distributed systems are essential in situations when the workload is subject to change, such as e-commerce traffic on Cyber Monday or lots of web traffic in response to news about your organization.

Distributed System

Because they draw on the capabilities of other computing devices and processes, distributed systems can offer features that would be difficult or impossible to develop on a single system.

This includes things like performing an off-site server and application backup — if the master catalog doesn’t see the segment bits it needs for a restore, it can ask the other off-site node or nodes to send the segments. Virtually everything you do now with a computing device takes advantage of the power of distributed systems, whether that’s sending an email, playing a game or reading this article on the web.

Distributed System

Here are some very common examples of distributed systems:

- Telecommunications networks that support mobile and internet networks
- Graphical and video-rendering systems
- Scientific computing, such as protein folding and genetic research
- Airline and hotel reservation systems
- Multiuser video conferencing systems
- Cryptocurrency processing systems (e.g. Bitcoin)
- Peer-to-peer file-sharing systems
- Distributed community compute systems
- Multiplayer video games
- Global, distributed retailers and supply chain management

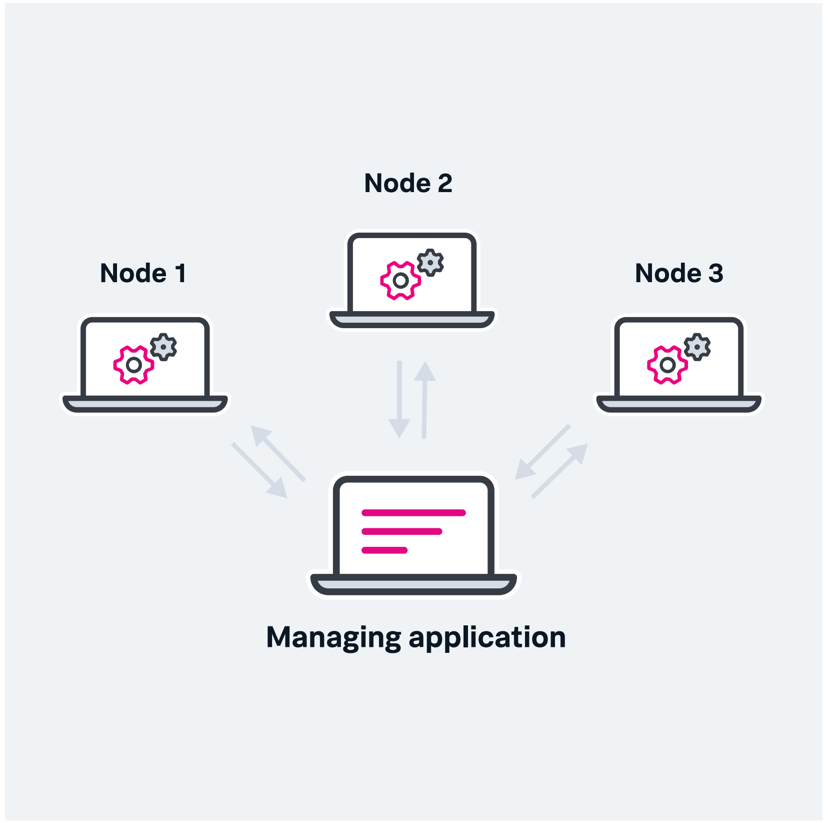
Examples at work

A distributed system begins with a task. Let’s pretend you need to render a video to create a finished product.

The application, or distributed applications, managing this task — like a video editor on a client computer — splits the job into pieces. In this simple example, the algorithm gives one frame of the video to each of a dozen different computers (or nodes) to complete the rendering. Once the frame is complete, the managing application gives the node a new frame to work on. This process continues until the video is finished and all the pieces are put back together.

A system like this doesn’t have to stop at just 12 nodes: the job may be distributed among hundreds or thousands of nodes, turning a task that might have taken days for a single computer to complete into one that is finished in a matter of minutes.

Examples at work



When thinking about the challenges of a distributed computing platform, the trick is to break it down into a series of interconnected patterns. Simplifying the system into smaller, more manageable and more easily understood components helps abstract a complicated architecture. Patterns are commonly used to describe distributed systems, such as:

- Command and query responsibility segregation (CQRS)
- Two-phase commit (2PC)

Different combinations of patterns are used to design distributed systems, and each approach has unique benefits and drawbacks.

Types of DS

Client-Server Architecture
Client-server architecture is a common type of distributed computing architecture. In this model, the system is divided into two types of nodes: clients and servers. Clients request services, and servers provide them. The servers are typically powerful computers that host and manage resources, while the clients are usually less powerful machines that access these resources.
Three-Tier Architecture
Three-tier architecture is a type of client-server architecture where the system is divided into three layers: the presentation layer, the application layer, and the data layer. The presentation layer handles the user interface, the application layer processes the business logic, and the data layer manages the database. By separating these functions, the system can achieve greater scalability, flexibility, and maintainability.
N-Tier Architecture
N-tier architecture is a further extension of the three-tier architecture. In this model, the system is divided into 'n' tiers or layers, where 'n' can be any number greater than three. Each layer is dedicated to a specific function, such as user interface, business logic, data processing, data storage, etc. This division of labor allows for greater modularity, making the system more scalable and easier to manage.
Peer-to-Peer Architecture
Peer-to-Peer (P2P) architecture is a type of distributed computing architecture where all nodes are equal, and each node can function as both a client and a server. In this model, there is no central server; instead, each node can request services from and provide services to other nodes. This decentralization makes P2P architectures highly scalable and resilient, as there is no single point of failure.

Distributed Vs Related Concepts

Distributed Computing vs. Cloud Computing

Distributed computing involves a collection of independent computers connected via a network, working together to perform tasks. Each computer in a distributed system operates autonomously, and the system is designed to handle failures of individual machines without affecting the entire system's functionality.

Cloud computing is a model for delivering computing services over the internet. It provides on-demand access to shared computing resources, such as servers, storage, and applications, without direct active management by the user. While distributed computing is about the system architecture, cloud computing is more about service delivery.

Distributed Computing vs. Grid Computing

Grid computing involves using the unused processing power of computers connected over a network (often the internet), to solve complex computational problems. It is a decentralized form of distributed computing where each node is independent, and there is no central coordinating system.

In contrast, distributed computing can be either centralized or decentralized, depending on the architecture. It involves multiple computers sharing the workload to achieve common goals. The computers in a distributed system may be physically close together and connected by a local network, or they may be geographically distant and connected by a wide area network.

Distributed Computing vs. Parallel Computing

Parallel computing is a type of computation in which many calculations or processes are carried out simultaneously. It breaks down a large problem into smaller, independent parts that can be solved concurrently.

While distributed computing and parallel computing share the objective of processing tasks more quickly, they differ in how they achieve this. In parallel computing, all processors may have access to shared memory to exchange information, while in distributed systems, each node has its own memory.

Technologies and Tools

Software Frameworks in Distributed Computing

Software frameworks are essential in distributed computing. They provide the necessary foundation and structure, enabling developers to focus on the unique aspects of their applications, rather than the complexities of network communication and task synchronization.

One of the most popular software frameworks in distributed computing is Apache Hadoop. This open-source platform allows for the processing of large datasets across clusters of computers. It is designed to scale up from a single server to thousands of machines, each providing local computation and storage. Its robustness comes from its fault-tolerance capability; if a machine fails, the tasks are automatically redirected to other machines to prevent application failure.

Another noteworthy framework is Apache Spark. Spark is known for its speed and ease of use in processing large-scale data. It supports multiple programming languages and offers libraries for machine learning, graph processing, and streaming analytics. Unlike Hadoop, which is disk-based, Spark's in-memory processing capability significantly speeds up computing tasks.

Distributed File Systems

Distributed file systems are another integral part of distributed computing. They facilitate the storage and retrieval of data across multiple machines, providing a unified view of data regardless of where it is physically stored.

Google File System (GFS) is a prominent example of a distributed file system. GFS is designed to provide efficient, reliable access to data using large clusters of commodity hardware. It achieves this through replication – storing multiple copies of data across different machines – thereby ensuring data availability and reliability even in the event of hardware failure.

Hadoop Distributed File System (HDFS) is another popular distributed file system. HDFS is designed to handle large data sets reliably and efficiently and is highly fault-tolerant. It divides large data files into smaller blocks, distributing them across different nodes in a cluster. This allows for efficient data processing and retrieval, as tasks can be performed on multiple nodes simultaneously.

Technologies and Tools

Distributed Databases

Distributed databases are the backbone of many modern applications. They store data across multiple nodes, ensuring high availability, performance, and scalability.

One of the leading distributed databases is Apache Cassandra. Cassandra offers high availability and scalability across many commodity servers, with no single point of failure. It provides a flexible schema and allows for fast writes, making it an excellent choice for applications that need to handle large amounts of data quickly.

Another popular distributed database is Amazon DynamoDB. DynamoDB is a fully-managed NoSQL database service that provides fast and predictable performance with seamless scalability. It offers built-in security, backup and restore, and in-memory caching, making it a robust choice for applications that need reliable, high-performance data access.

Cloud Platforms

Cloud computing platforms offer a vast array of resources and services, enabling businesses to scale and innovate faster based on distributed computing infrastructure.

Amazon Web Services (AWS) is a leading cloud platform. AWS provides a broad set of products and services, including computing power, storage options, networking, and databases, tailored to meet the specific needs of organizations. Its pay-as-you-go approach allows businesses to scale as needed, reducing the cost and complexity of planning and maintaining on-premises infrastructure.

Google Cloud Platform (GCP) is another major player in the cloud space. GCP offers similar services as AWS but is particularly strong in data analytics and machine learning. Its robust data storage and compute services, combined with its cutting-edge machine learning and AI capabilities, make it a compelling choice for businesses looking to leverage data to drive innovation.

Technologies and Tools

Virtualization and Containerization

Virtualization and containerization are key technologies in distributed computing. They allow for the efficient deployment and management of applications across multiple machines.

Virtualization involves creating a virtual version of a server, storage device, or network resource. VMware is a leading provider of virtualization software, offering solutions for server, desktop, and network virtualization. Many distributed computing infrastructures are based on virtual machines (VMs).

Docker is a leading platform for containerization. Docker containers package software into standardized units for development, shipment, and deployment. This ensures that the software runs the same in any environment, making it easy to deploy applications across multiple distributed resources.

Kubernetes is a powerful system for managing containerized applications. It automates the deployment, scaling, and management of applications, making it an excellent choice for businesses looking to scale their distributed applications efficiently.

Technologies and Tools

Distributed Computing Optimization with Run:ai
Run:ai automates resource management and orchestration for distributed machine learning infrastructure. With Run:ai, you can automatically run as many compute intensive experiments as needed.

Here are some of the capabilities you gain when using Run:ai:

•**Advanced visibility**—create an efficient pipeline of resource sharing by pooling GPU compute resources.
•**No more bottlenecks**—you can set up guaranteed quotas of GPU resources, to avoid bottlenecks and optimize billing.
•**A higher level of control**—Run:ai enables you to dynamically change resource allocation, ensuring each job gets the resources it needs at any given time.

Run:ai simplifies machine learning infrastructure pipelines, helping data scientists accelerate their productivity and the quality of their models.

Reasons for Building OS

•**Scalability**:
•They can easily scale horizontally by adding more machines, accommodating increasing loads without significant redesign.
•**Fault Tolerance**:
•Distributed systems can continue functioning even if some components fail, enhancing reliability and availability.
•**Resource Sharing**:
•They allow for sharing of resources (like CPU, storage, etc.) across multiple systems, leading to more efficient utilization.

Cost Efficiency:
•Utilizing commodity hardware can be more cost-effective than investing in a single powerful machine.

•**Geographical Distribution**:
•They can serve users across different locations effectively, reducing latency by processing data closer to the user.
•**Performance**:
•By distributing workloads, these systems can achieve better performance, particularly for tasks that can be parallelized.
•**Flexibility and Maintainability**:
•Components can be updated or replaced independently, allowing for easier maintenance and iterative improvements.

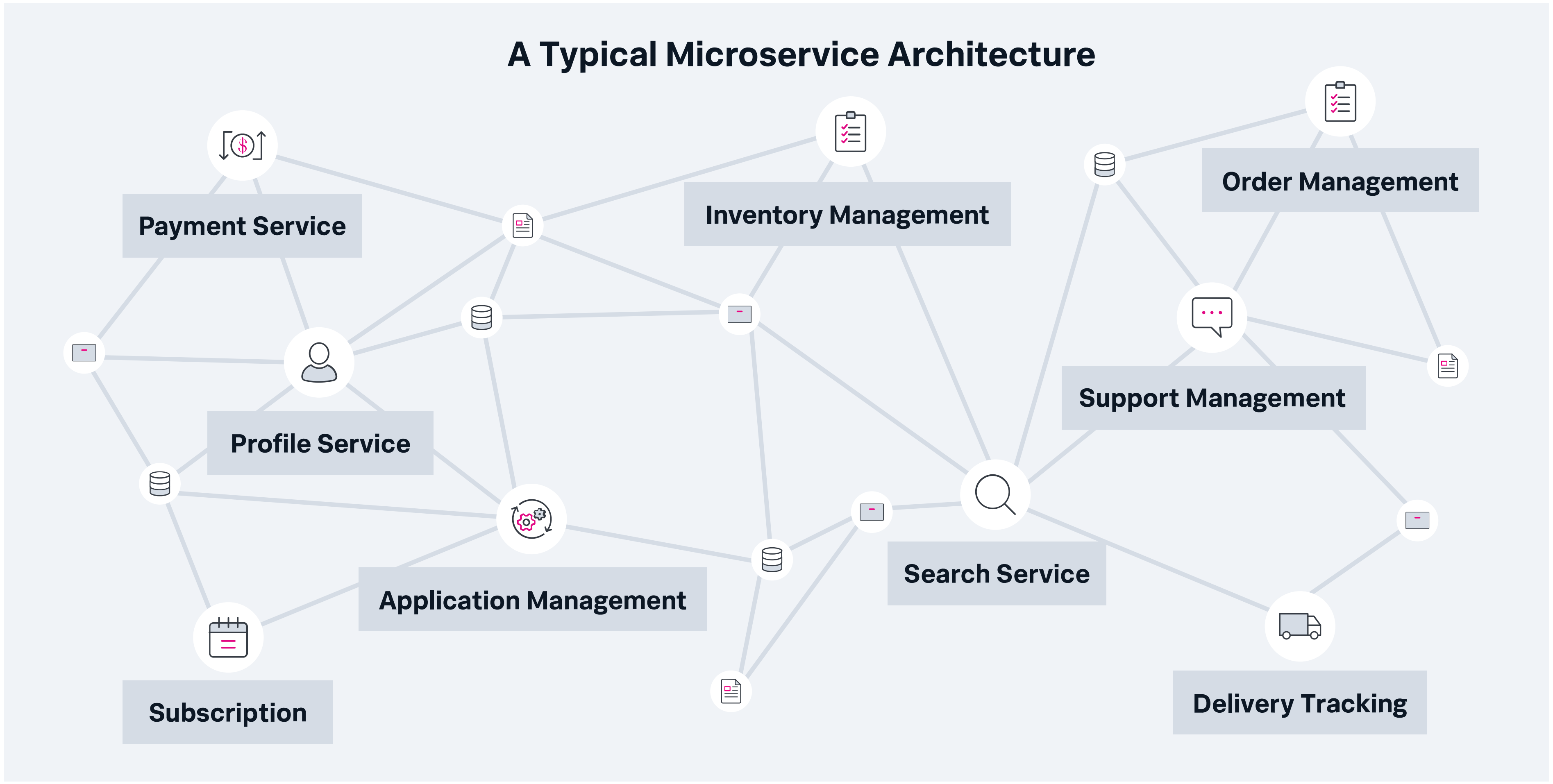
Key Characteristics

- So now that we “get” what distributed systems are, we can start to assign key features to them. Here’s what good distributed systems have in common:
- **Scalability.** The ability to grow as the size of the workload increases is an essential feature of distributed systems, accomplished by adding additional processing units or nodes to the network as needed.
- **Concurrency.** Distributed system components run simultaneously. They’re also characterized by the lack of a “global clock,” when tasks occur out of sequence and at different rates.
- **Availability and fault tolerance.** If one node fails, the remaining nodes can continue to operate without disrupting the overall computation.
- **Heterogeneity.** In most distributed systems, the nodes and components are often asynchronous, with different hardware, middleware, software and operating systems. This allows the distributed systems to be extended with the addition of new components.
- **Replication.** Distributed systems enable shared information and messaging, ensuring consistency between redundant resources, such as software or hardware components, improving fault tolerance, reliability and accessibility.
- **Transparency.** The end user sees a distributed system as a single computational unit rather than as its underlying parts, allowing users to interact with a single logical device rather than being concerned with the system’s architecture.

Benefits, Challenges and Risks of DS

- Distributed systems offer a number of advantages over monolithic, or single, systems:
- **Scalability & flexibility.** It is easier to add computing power as the need for services grows. In most cases today, you can spin up servers to a distributed system on the fly, increasing performance and further reducing time to completion.
- **Fault tolerance.** Distributed systems reduce the risks involved with having a single point of failure, bolstering reliability and fault tolerance.
- **Reliability:** A well-designed distributed system can withstand failures in one or more of its nodes without severely impacting performance. In a monolithic system, the entire application goes down if the server goes down.
- **Speed.** Heavy traffic can bog down single servers when traffic gets heavy, impacting performance for everyone. The scalability of distributed databases and other distributed systems makes them easier to maintain and also sustain high-performance levels.
- **Geo-distribution.** Distributed content delivery is both intuitive for any internet user, and vital for global organizations.

Benefits, Challenges and Risks of DS



Benefits, Challenges and Risks of DS

- Distributed systems are considerably more complex than monolithic computing environments, and raise a number of challenges around design, operations and maintenance. These include:
- **Increased opportunities for failure:** The more systems added to a computing environment, the more opportunity there is for failure. If a system is not carefully designed and a single node crashes, the entire system can go down. While distributed systems are designed to be fault tolerant, that fault tolerance isn't automatic or foolproof.
- **Synchronization process challenges:** Distributed systems work without a global clock, requiring careful programming to ensure that processes are properly synchronized to avoid transmission delays that result in errors and data corruption. In a complex system — such as a multiplayer video game — synchronization can be challenging, especially on a public network that carries data traffic.
- **Imperfect scalability:** Doubling the number of nodes in a distributed system doesn't necessarily double performance. Architecting an effective distributed system that maximizes scalability is a complex undertaking that needs to take into account load balancing, bandwidth management and other issues.
- **More complex security:** Managing a large number of nodes in a heterogeneous or globally distributed environment creates numerous security challenges. A single weak link in a file system or larger distributed system network can expose the entire system to attack.
- **Increased complexity:** Distributed systems are more complex to design, manage and understand than traditional computing environments.

Contd.

- The challenges of distributed systems create a number of correlating risks.
- **Security.** Distributed systems are as vulnerable to attack as any other system, but their distributed nature creates a much larger attack surface that exposes organizations to threats.
- **Risk of network failure.** Distributed systems are beholden to public networks to transmit and receive data. If one segment of the internet becomes unavailable or overloaded, distributed system performance may decline.
- **Governance and control issues.** Distributed systems lack the governability of monolithic, single-server-based systems, creating auditing and adherence issues around data privacy laws. Globally distributed environments are challenging when it comes to providing certain levels of assurance and understanding exactly where data resides.
- **Cost control.** Unlike centralized systems, the scalability of distributed systems allows administrators to easily add additional capacity as needed, which can also increase costs. Pricing for cloud-based distributed computing systems are based on usage (such as the number of memory resources and CPU power consumed over time). If demand suddenly spikes, you might face a massive bill.

How to setup

- Distributed deployments can range from tiny, single department deployments on local area networks to large-scale, global deployments. In addition to their size and overall complexity, organizations can consider deployments based on:
- The size and capacity of their computer network
- The amount of data they’ll consume
- How frequently they run processes and whether they'll be scheduled or ad hoc
- The number of users accessing the system
- Capacity of their data center
- The necessary data fidelity and availability requirements
- Distributed deployments are categorized as departmental, small enterprise, medium enterprise or large enterprise. By no means formal, these categories are a starting point for planning the needed resources to implement a distributed computing system.
- Distributed systems can also evolve over time, transitioning from departmental to small enterprise as the enterprise grows and expands.

Tracking what goes on Distributed System's

- We know clearly that, for all their benefits, distributed systems are complicated. Knowing what goes on within — the observability of that system — is a distinct advantage. Luckily, it's one you can achieve with distributed tracing.
- Without distributed tracing, a globally distributed system environment would be impossible to monitor effectively.
- Distributed tracing, sometimes called distributed request tracing, is a method for monitoring applications — typically those built on a microservices architecture — which are commonly deployed on distributed systems. Distributed tracing is essentially a form of distributed computing in that it's commonly used to monitor the operations of applications running on distributed systems.
- In software development and operations, tracing is used to follow the course of a transaction as it travels through an application. An online credit card transaction as it winds its way from a customer's initial purchase to the verification and approval process to the completion of the transaction, for example. A tracing system monitors this process step by step, helping a developer to uncover bugs, bottlenecks, latency or other problems with the application.
- Distributed tracing is necessary because of the considerable complexity of modern software architectures. A distributed tracing system is designed to operate on a distributed services infrastructure, where it can track multiple applications and processes simultaneously across numerous concurrent nodes and computing environments.

Applying Access control in distributed systems

- Administrators use a variety of approaches to manage access control in distributed computing environments, ranging from traditional access control lists (ACLs) to role-based access control (RBAC).
- One of the most promising access control mechanisms for distributed systems is attribute-based access control (ABAC), which controls access to objects and processes using rules that include information about the user, the action requested and the environment of that request. Administrators can also refine these types of roles to restrict access to certain times of day or certain locations.

Distributed Systems aren't going away

- Distributed systems are well-positioned to dominate computing as we know it for the foreseeable future, and almost any type of application or service will incorporate some form of distributed computing. The need for always-on, available-anywhere computing isn’t disappearing anytime soon.