Monitoring Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes.

White-box monitoring Monitoring based on metrics exposed by the internals of the system, including logs, interfaces like the Java Virtual Machine Profiling Interface, or an HTTP handler that emits internal statistics.

Black-box monitoring Testing externally visible behavior as a user would see it.

The four golden signals of monitoring are latency, traffic, errors, and saturation.

Latency The time it takes to service a request.

Traffic A measure of how much demand is being placed on your system, measured in a high-level system specific metric.

Errors The rate of requests that fail, either explicitly (e.g., HTTP 500s), implicitly (for example, an HTTP 200 success response, but coupled with the wrong content), or by policy (for example, "If you committed to one-second response times, any request over one second is an error").

Saturation How "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained (e.g., in a memory-constrained system, show memory; in an I/O-constrained system, show I/O).

Declarative approach the focus is on what the computer should do rather than how it should do it “this is what i want, now you work out how to do it” (ex. SQL). With an imperative approach, a developer writes code that specifies the steps that the computer must take to accomplish the goal. This is sometimes referred to as *algorithmic* programming (ex. C, C++, Java). In contrast, a functional approach involves composing the problem as a set of functions to be executed. You define carefully the input to each function, and what each function returns. The following table describes some of the general differences between these two approaches.

FUNCTIONAL PROGRAMMING VS. IMPERATIVE PROGRAMMING

Characteristic	Imperative approach	Functional approach
Programmer focus	How to perform tasks (algorithms) and how to track changes in state.	What information is desired and what transformations are required.
State changes	Important.	Non-existent.
Order of execution	Important.	Low importance.
Primary flow control	Loops, conditionals, and function (method) calls.	Function calls, including recursion.
Primary manipulation unit	Instances of structures or classes.	Functions as first-class objects and data collections.

A **data model** organizes data elements and standardizes how the data elements relate to one another.

Types:

Conceptual Data Model are the most simple and abstract. Little annotation or data use occurs in this model, but the overall layout and rules of the data relationships are set. You’ll find elements like basic business rules that need to be applied, the categories or entity classes of data that you plan to include, and any other regulations that may limit layout options. Conceptual data models are frequently used in the discovery stage of a project.

The logical data model expands on the basic framework laid out in the conceptual model, but it considers more relational factors. You’ll see some basic annotation related to overall properties, or data attributes,

but not many annotations that focus on actual units of data. This model is particularly useful in data warehousing plans.

Since the physical data model is the most detailed and usually the final step before database creation, it often accounts for database management system-specific properties and rules. You'll illustrate enough detail about data points and their relationships to create a schema or a final actionable blueprint with all needed instructions for the database build.

Link https://www.datamation.com/big-data/what-is-data-modeling/

differences between SQL vs. NoSQL are:

1. SQL databases are relational, NoSQL databases are non-relational.
2. SQL databases use structured query language and have a predefined schema. NoSQL databases have dynamic schemas for unstructured data.
3. SQL databases are vertically scalable, while NoSQL databases are horizontally scalable.
4. SQL databases are table-based, while NoSQL databases are document, key-value, graph, or wide-column stores.
5. SQL databases are better for multi-row transactions, while NoSQL is better for unstructured data like documents or JSON.

Vertical scaling refers to adding more resources (CPU/RAM/DISK) to your server (database or application server is still remains one) as on demand.

Horizontal scaling refers to adding or removing databases in order to adjust capacity or overall performance, also called "scaling out". Sharding, in which data is partitioned across a collection of identically structured databases, is a common way to implement horizontal scaling. is add more machines to the party

In a more formal, and complete manner, we can describe distribution, scalability, availability, and even fault tolerance (later on these) through the prism of "The Scale Cube" or "AKF Cube".

X-axis: clone/replicate data/processes

Y-axis: functional decomposition, as in microservices

Z-axis: sharding, or data/process splitting by some attribute, could be location, user type or something else

Monolithic architecture is seen as the traditional method of application development. An application in a monolithic architecture is developed as a single package. With all the components working hand in hand with each other and is aligned with OOP principles.

A microservice architecture includes a set of small, independent and self-contained modules that perform various services. Each service should be able to independently implement the corresponding business units. Each module can be written in any language, has its own database and can communicate with other parts of the system through APIs.

Link to seminar answers https://github.com/DivineBee/microservices-cloud/blob/main/docs/Seminar1.pdf

8 fallacies(false assumptions) of distributed computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

1The network is reliable. reliability is a degree to which a product or service conforms to its specifications when in use, even in the case of failures.||| Networks are complex, dynamic, and often unpredictable. Many reasons could lead to a network failure or network-related issues: a switch or a power failure, misconfiguration, an entire datacenter becoming unavailable, DDoS attacks, etc. Due to this complexity and general unpredictability, networks are unreliable.

2. Latency is zero. Latency may be close to zero when you're running apps in your local environment, and it's often negligible on a local area network. However, latency quickly deteriorates in a wide area network. That's because in a WAN data often has to travel further from one node to another, since the network may span large geographical areas.

3. Bandwidth is infinite While latency is the speed at which data travels from point A to point B, bandwidth refers to how much data can be transferred from one place to another in a certain timeframe. When a high volume of data is trying to flow through the network, and there is insufficient bandwidth support, various issues can arise:

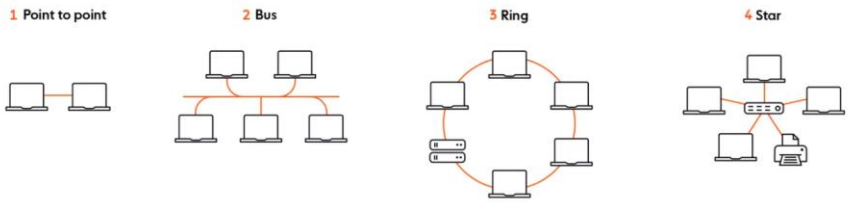
- Queuing delays, bottlenecks, and network congestion.
- Packet loss leading to inferior quality of service guarantees, i.e., messages getting lost or being delivered out of order.
- Abysmal network performance and even overall system instability.

ways to improve your network bandwidth capacity:

1. Comprehensive monitoring. It's essential to track and check usage in your network, so you can quickly identify issues (e.g., who or what is hogging your bandwidth) and take the appropriate remedial measures.
2. Multiplexing. Protocols like HTTP/2, HTTP/3, and WebSockets all support multiplexing, a technique that improves bandwidth utilization by allowing you to combine data from several sources and send it over the same communication channel/medium.
3. Lightweight data formats. You can preserve your network bandwidth by using data exchange formats built for speed and efficiency, like JSON. Another option is MessagePack, a compact binary serialization format that creates even smaller messages than JSON.
4. Network traffic control. You need to think about using mechanisms such as throttling/rate limiting, congestion control, exponential backoff, etc.

4. The network is secure There are many ways a network can be attacked or compromised: bugs, vulnerabilities in operating systems and libraries, unencrypted communication, oversights that lead to data being accessed by unauthorized parties, viruses and malware, cross-site scripting (XSS), and DDoS attacks…

5. Topology doesn't change In a distributed system, network topology changes all the time. Sometimes, it's for accidental reasons or due to issues, such as a server crashing. Other times it's deliberate — we add, upgrade, or remove servers.



6. There is one administrator there is usually more than one administrator of a distributed system in nearly all real-life scenarios. For example, think of modern cloud-native systems that consist of many services developed and managed by different teams. Or consider that customers that use your system also need administrators on their side to manage the integration. Here are a couple of things to consider:

Decouple system components. Ensuring appropriate decoupling allows for greater resiliency in the context of either planned upgrades leading to issues, or unplanned events, such as failures. One of the most popular options facilitating decoupling (or loose coupling) is the pub/sub pattern.

Make troubleshooting easy. It's essential to provide visibility into your system so administrators can diagnose and resolve issues that may occur. Return error messages and throw exceptions so administrators have context and can take the appropriate course of action to fix problems. In addition, logging, metrics, and tracing (often called the three pillars of observability) should be key aspects of your system's design.

7. Transport cost is zero First of all, networking infrastructure has a cost. Servers, network switches, load balancers, proxies, firewalls, operating and maintaining the network, making it secure, not to mention staff to keep it running smoothly — all of these cost money. The bigger the network, the bigger the financial cost. also consider the time, effort, and difficulty involved in architecting a distributed system that works over a highly available, dependable, and fault-tolerant network.

8. The network is homogeneous Most distributed systems need to integrate with multiple types of devices, adapt to various operating systems, work with different browsers, and interact with other systems. Therefore, it's critical to focus on interoperability, thus ensuring that all these components can "talk" to each other, despite being different.

Link to article https://ably.com/blog/8-fallacies-of-distributed-computing

OLTP (Online Transaction Processing), a transactional system - processing transactions in real time. A way of organizing a database, in which the system works with small-sized transactions, but with a large flow, and at the same time the client requires the minimum response time from the system.

Advantages High reliability of data as a result of the transactional approach. The transaction either completes successfully, or fails and the system reverts to its previous state. For any outcome of the transaction, the data integrity is not violated.

Disadvantages OLTP systems are optimized for small discrete transactions. But requests for some complex information (for example, quarterly dynamics of sales volumes for a certain product model in a certain branch), typical for analytical applications (OLAP), will generate complex table joins and whole tables view. One such request will take a lot of time and computer resources, which will slow down the processing of current transactions.

OLAP (Online analytical processing)- performs multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling. Unlike relational databases, OLAP tools do not store individual transaction records in two-dimensional, row-by-column format, like a worksheet, but instead use multidimensional database structures—known as Cubes in OLAP terminology—to store arrays of consolidated information. There are primary five types of analytical OLAP operations in data warehouse: 1) Roll-up 2) Drill-down 3) Slice 4) Dice and 5) Pivot

Advantages Quickly create and analyze "What if" scenarios. It is a powerful visualization online analytical process system which provides faster response times. Allows users to do slice and dice cube data all by various dimensions, measures, and filters.

Disadvantages OLAP requires organizing data into a star or snowflake schema. These schemas are complicated to implement and administer. Transactional data cannot be accessed with OLAP system. Any modification in an OLAP cube needs a full update of the cube. This is a time-consuming process

	OLTP	OLAP
Main Read Pattern	Small nr. of records per query	Aggregate over large nr. of records
Main Write Pattern	Random access, low latency writes	Bulk imports
Usage	End customer	Internal analyst
Data	Latest state of data (current)	History of events that happened in time (logs)
Data size	Gb - Tb	Tb - Pb

OLTP VS OLAP

DB Type	OLTP Database	OLAP Database
Purpose	• Collect and store transactional data. • Maintain data integrity. • Process queries to support business processes run by applications or employees.	• Aggregate transactional data for analysis. • Support business decision making. • Discover trends and insights.
Query type	Simple queries to run commands like: INSERT, UPDATE, DELETE	Complex queries with custom commands
Data source	Transactions	Aggregated transaction data
Data update	Fast updates on separate data points, or small batches	Large, or usually full batch updates
Data view	Flat two-dimensional view	Multidimensional view
Transaction duration	Short transaction (response measured in milliseconds)	Long transactions (response measured in minutes or hours)
User	Operational staff or business applications	Data analysts, business analysts, managers of all levels

A data warehouse centralizes and consolidates large amounts of data from multiple sources. Its analytical capabilities allow organizations to derive valuable business insights from their data to improve decision-making. Over time, it builds a historical record that can be invaluable to data scientists and business analysts.

A typical data warehouse often includes the following elements:

- A relational database to store and manage data
- An extraction, loading, and transformation (ELT) solution for preparing the data for analysis
- Statistical analysis, reporting, and data mining capabilities
- Client analysis tools for visualizing and presenting data to business users
- Other, more sophisticated analytical applications that generate actionable information by applying data science and artificial intelligence (AI) algorithms, or graph and spatial features that enable more kinds of analysis of data at scale

Benefits:

- Subject-oriented. They can analyze data about a particular subject or functional area (such as sales).
- Integrated. Data warehouses create consistency among different data types from disparate sources.
- Nonvolatile. Once data is in a data warehouse, it's stable and doesn't change.
- Time-variant. Data warehouse analysis looks at change over time.

An example **of transparency** - when you keep data in your Google Drive you never notice that it might have changed the server where it is placed. This is both location and migration transparency.

Access transparency – Regardless of how resource access and representation has to be performed on each individual computing entity, the users of a distributed system should always access resources in a single, uniform way.

Location transparency – Users of a distributed system should not have to be aware of where a resource is physically located.

Relocation transparency – Should a resource move while in use, this should not be noticeable to the end user.

Replication transparency – If a resource is replicated among several locations, it should appear to the user as a single resource.

Concurrent transparency – While multiple users may compete for and share a single resource, this should not be apparent to any of them.

Failure transparency – Always try to hide any failure and recovery of computing entities and resources.

Persistence transparency – Whether a resource lies in volatile or permanent memory should make no difference to the user.

Fragmentation transparency – Regardless of what a resource is made of (fragments), the users of a distributed system should always access resources in a single, uniform way.

resilient system is not failure-proof but rather failure-tolerant. Just as with scalability, one of the primary tools we can use to ensure our systems are failure tolerant is having redundancy in our systems, i.e. replication. we just have copies of our data and services, such that in case we lose particular instances, we still have our "backups". For example, we can replicate our databases in case some database instances fail. Or, we could have multiple copies of our services, to ensure that if some server dies, the whole system continues to work.

failover strategies:

Active-active: have both the primary and secondary system running, if primary fails, immediately direct the traffic to the secondary

Active-passive: have only the primary system running, if it fails, launch the secondary system and start receiving all the traffic on it