Availability: System Design Concept

Many of us may have experienced moments where we could not access certain applications due to an outage or unavailability. Recently, YouTube faced a global outage that stopped users from streaming videos for about an hour. You may wonder about the reason and How one can prevent it from happening? Let's Find out.

What is Availability?

Availability is the percentage of time in a given period that a system is available to perform its task and function under normal conditions. One way to look at is how resistant a system is to failures. The percentage of availability that a system requires depends on the usage of the system. Let us take some examples.

Air Traffic Control systems are among the best examples of systems that require high availability. In today's world, where air travel is so complex and busy, a single error in directing airplanes can lead to catastrophic results. In contrast, a system with few visitors and not prone to catastrophic failures require slightly lesser available systems. High Availability comes with a cost, so we have to optimize according to our needs.

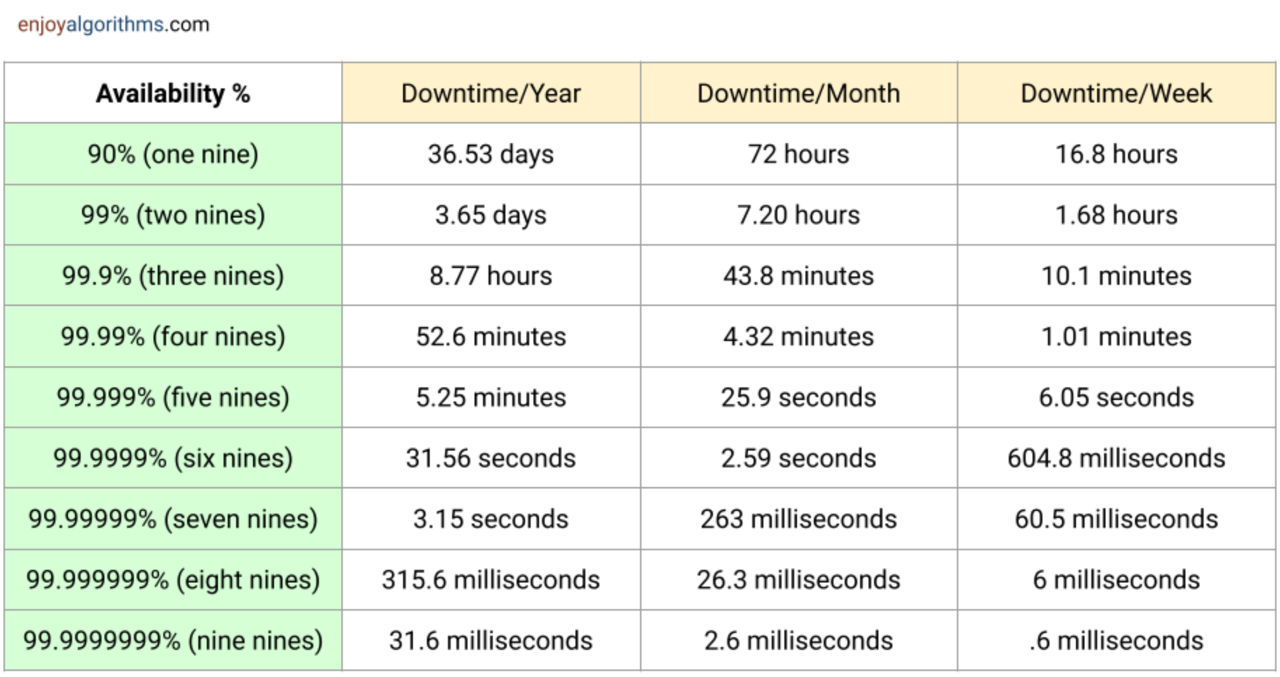
How is Availability Measured?

A system's availability is measured as the percentage of a system's uptime in a given time period or by dividing the total uptime by the total uptime and downtime in a given period of time.

Availability = Uptime ÷ (Uptime + Downtime)

The Nine's of Availability

Availability can also be expressed in terms of Nines. In high-demand applications, we usually measure availability in terms of Nines rather than percentages. If availability is 99.00 percent, it is said to have “2 nines” of availability, and if it is 99.9 percent, it is called “3 nines,” and so on. A system with 5 nines (i.e., 99.999%) of availability is said to have a Gold Standard of Availability. Let's take a look at different Nines of Availability.



Availability %	Downtime/Year	Downtime/Month	Downtime/Week
90% (one nine)	36.53 days	72 hours	16.8 hours
99% (two nines)	3.65 days	7.20 hours	1.68 hours
99.9% (three nines)	8.77 hours	43.8 minutes	10.1 minutes
99.99% (four nines)	52.6 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.25 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.56 seconds	2.59 seconds	604.8 milliseconds
99.99999% (seven nines)	3.15 seconds	263 milliseconds	60.5 milliseconds
99.999999% (eight nines)	315.6 milliseconds	26.3 milliseconds	6 milliseconds
99.9999999% (nine nines)	31.6 milliseconds	2.6 milliseconds	.6 milliseconds

How do we achieve High Availability?

High availability is the ability of a system to maintain operation despite the failure of components. To increase availability, we can use **redundancy** by duplicating or adding additional hardware (servers or storage) components. For example, a system with two identical web servers behind a load balancer can continue operating even if one of the servers goes down, as the load balancer will redirect traffic to the remaining server. So by adding redundancy, we can make the system more resilient to failure.

- **Passive Redundancy:** Here only some of the components (server or storage device) are active at any given time and backup components are available in case of a failure. If some component will fail, the backup component will takes over and becomes active. This will allow system to continue to operate and maintain availability.
- **Active Redundancy:** Here multiple active components (servers or storage devices) work simultaneously to perform the task. In the

event of a failure of one of the active components, the other active components can take over and maintain the availability of the system.

It is important to note that redundancy alone is not enough to guarantee high availability. **Failure detection mechanisms** must also be in place to identify failures. This requires regular high-availability testing and the ability to take corrective action whenever one of the components in the system becomes unavailable.

There are both **hardware** and **software** based approaches to achieving high availability. Redundancy is a hardware-based approach, while other techniques such as top-to-bottom or distributed high-availability approaches can involve both hardware and software. Software-based downtime reduction techniques can also be effective.

There is a trade-off between the availability of a system and its performance. To achieve high availability, we often take measures to implement redundancy or disaster recovery strategies, which can hurt other aspects of system performance (higher latency or lower throughput). For example, implementing redundancy may involve replicating data or tasks across multiple resources, which can increase the time it takes to complete a task, resulting in higher latency.

Difference between High Availability and Fault Tolerance

Both high availability and fault tolerance are strategies used to achieve high uptime in systems, but they approach the problem differently. High availability is about system or component's ability to remain operational and accessible with minimal downtime. On other side, Fault tolerance is about system or component's ability to continue functioning normally even in the event of a failure.

- Fault tolerance involves utilization of multiple systems that run in parallel. In the event of a failure in the main system, another system can take over without any loss of uptime. This requires advanced hardware that can detect component faults and enable the systems to operate in coordination. However, it may take longer for complex networks and devices to respond to malfunctions, and technical issues that result in a system crash may also cause the failure of redundant systems running in parallel, leading to a system-wide failure.

- High availability, on the other hand, also uses software-based approach to minimize server downtime rather than relying on hardware redundancy. A high-availability cluster uses a collection of servers together to achieve maximum redundancy. This can be more flexible and easier to implement than a fault-tolerant system, but it may not provide the same level of protection against system failures.

Thanks to Chiranjeev and Navtosh for their contribution in creating the first version of this content. If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!