

1. Stateless vs Stateful Servers

Stateless and stateful servers differ in how they manage information about client interactions. A **stateless server** does not retain any record of previous requests once a response has been sent. Each request from a client is processed independently and must contain all the information required for execution. This design greatly simplifies server logic and improves scalability, since requests can be distributed across multiple servers using load balancers without worrying about session data. Stateless servers are also more reliable in distributed systems because failure of one server does not disrupt ongoing interactions. Maintenance and recovery are easier because there is no session data to restore. Common examples include RESTful web services, HTTP servers serving web pages, and DNS servers.

A **stateful server**, in contrast, preserves information about a client's session across multiple requests. This state may include login credentials, transaction history, or application progress. Stateful servers allow richer, context-aware interactions but are more complex to manage in distributed environments. Scaling is harder because client requests often need to return to the same server instance, and failures may result in data loss or interrupted sessions. Additional mechanisms such as session replication are required to improve reliability. Examples of stateful servers include database management systems, online banking transaction systems, and multiplayer online game servers.

2. Advantages of Containers over Virtual Machines in Distributed Deployment

Containers offer significant advantages over virtual machines in distributed system deployments due to their efficiency and flexibility. One major advantage is that containers are lightweight, as they share the host operating system kernel rather than running a full guest OS like VMs. This reduces memory and storage overhead and allows many more containers to run on the same physical machine.

Another important advantage is faster startup time. Containers can be launched in seconds, compared to minutes for Virtual machines, enabling rapid scaling and quick recovery from failures. This is particularly beneficial in microservices architectures where services are frequently created and destroyed. Containers also improve portability and consistency, as applications are packaged together with their dependencies, ensuring predictable behavior across development, testing, and production environments.

In addition, container orchestration tools such as Kubernetes simplify distributed deployment by providing automated scaling, service discovery, load balancing, and fault tolerance. While Virtual Machines still provide stronger isolation and security in some cases, containers offer superior performance, resource utilization, and scalability, making them the preferred choice for modern distributed systems.

