CLONES

A person can interact with your service and be served at any request of any server – application servers involve a lot of servers.

– Every server contains exactly the same codebase and does not store any user-related data, like sessions or profile pictures.

– Sessions need to be stored in a centralized data store – accessible to all application servers:
+ Either external database
+ Or external persistent cache (Redis) – better performance than external database

External = data store does not reside on the application servers – somewhere in or near the data center of your application servers

– **Why This Matters?**

- **Session Consistency**: Users can be routed to any application server, and their session state will remain consistent.
- **Load Balancing**: Session data is not tied to a specific server, allowing load balancers to distribute requests efficiently.
- **Fault Tolerance**: If an application server goes down, session data is not lost, as it is stored centrally.

– **Use Case Scenarios**

- **High-Traffic Applications**: Websites with many users, where user sessions need to be maintained across various servers.
- **Microservices Architectures**: Different services need to share session data to provide a seamless user experience.
- **Cloud Environments**: Applications deployed in cloud environments often use centralized data stores for session management due to their distributed nature.

– **How do we make sure that a code change is sent to all of the servers without one server still serving old code?** – Capistrano: a deployment automation tool primarily designed for Ruby on Rails applications but can be adapted for other technologies.

Steps After Outsourcing Sessions and Standardizing the Codebase:

Outsource Sessions:

- Session data is stored in a centralized external data store (e.g., Redis or a database).
- This decouples session management from the application servers, allowing them to be stateless.

Serve the Same Codebase:

- Ensure all servers run the same code version by automating deployment with Capistrano.

Create an Amazon Machine Image (AMI):

– An Amazon Machine Image is a snapshot of an EC2 instance that includes the operating system, application code, runtime, libraries, and configurations.

– Steps to Create an AMI:

+ Deploy your application on a server.
+ Ensure it is fully configured and running the latest code.
+ Create an AMI from this server, capturing its state.

– Use the AMI as a Base for New Instances:

+ New EC2 instances can be launched using the AMI, ensuring they start with the same base setup.
+ This process guarantees consistency in the environment and application state across all instances.

– Initial Deployment of Latest Code:

+ After launching a new instance from the AMI, **perform an initial deployment using Capistrano to ensure the instance has the latest code changes.**
+ This step is crucial as the AMI might not always include the very latest code if updates were made after the image was created.