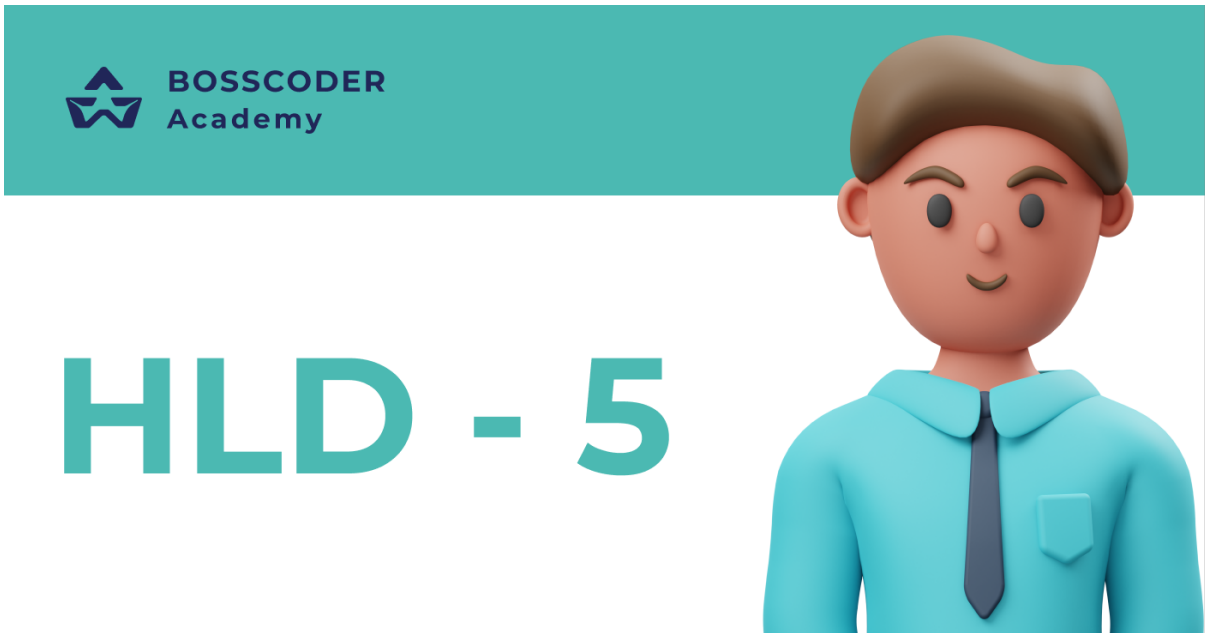
BOSSCODER ACADEMY

Request A Callback



HLD - 5

Abhishek Singh
8th August, 2023

Peer-to-Peer Networks

A Peer-to-Peer (P2P) network is a decentralized architecture where individual nodes, or peers, communicate and share resources directly with each other. Unlike traditional client-server models, where a central server manages interactions, P2P networks distribute tasks and responsibilities across participating peers.

- Peers can act as both clients and servers, enabling them to share and consume resources.
- Resources can include files, data, processing power, and bandwidth.
- There's no single point of failure, as each peer can continue functioning even if other peers go offline.
- P2P networks are highly scalable and can handle a large number of participants.
- Examples of P2P networks include file-sharing platforms like BitTorrent and communication tools like Skype.

Designing a High-Level Code Deployment System

Requirements Gathering:

- **Deployment Flow:** Understand the sequence of steps involved in code deployment. Is there a testing phase before deployment, or is it a direct deployment from the final code?
- **Deployment Scope:** Clarify whether the deployment is local or global. Are deployments restricted to specific regions?
- **Number of Regions:** Determine how many regions are involved in the deployment process.
- **Binary Size:** Gather information about the size of the binary files being deployed.
- **Deployment Frequency:** Understand how often deployments occur in a day.

High-Level Design:

Build Code:

- This component transforms source code into compiled code that is ready for deployment.
- It may involve compiling, optimizing, and generating necessary artifacts.

Deploy Code:

- This component is responsible for transferring compiled code to the target machines for actual deployment.
- It ensures that the new code is correctly propagated to all necessary destinations.

Build Code Component:

- **History Maintenance:** Decide whether a record of deployment history needs to be maintained. This includes tracking which versions of code were deployed, when, and by whom. This history can aid in debugging and auditing.
- **Storage Mechanism:** Choose an appropriate storage mechanism for maintaining deployment history. A combination of a SQL database and a version control system (e.g., Git) might provide the necessary capabilities.
- **Indexing:** Determine whether indexing is required for efficient querying of deployment history records.
- **Concurrency Handling:** Design the system to handle concurrent deployments gracefully. Strategies like row-level locking can prevent conflicts when multiple deployments occur simultaneously.

deployment tasks as they are executed. This tracking helps maintain a detailed history of each deployment, enabling administrators and developers to monitor progress, troubleshoot issues, and maintain an audit trail of code changes.

A 'Jobs' table is used to store information related to deployment tasks. It's a structured way to organize and manage the deployment process. Let's break down the components of the 'Jobs' table and their significance:

1. Job ID: A unique identifier assigned to each deployment job. It provides a reference point to quickly identify and distinguish between different deployment instances.

2. User ID: The ID of the user initiating the deployment. This could be an administrator, developer, or anyone authorized to trigger deployments. It helps track who initiated the deployment for accountability.

3. Binary Name: The name of the compiled binary or artifact being deployed. This could be the executable, library, or package generated from the source code. It aids in identifying the specific version of code being deployed.

4. URL: The URL or file path pointing to the compiled binary. This helps the deployment system locate the binary for transfer to the target machines.

5. Commit Hash: In version control systems like Git, a commit hash uniquely identifies a specific snapshot of code. Including the commit hash in the 'Jobs' table links the deployment job to the exact version of the source code.

Timestamps provide chronological context and help in tracking the sequence of deployments.

7. Status: Indicates the current status of the deployment job. It can take values such as "successful," "failed," or "in progress." This column allows stakeholders to quickly assess the outcome of a deployment and take appropriate actions.

Benefits of 'Jobs' Tracking:

- **Auditing and Accountability:** By maintaining a record of who initiated each deployment, the system can track changes made by different users over time.
- **Troubleshooting:** In case of failed deployments, having detailed deployment records with commit hashes and other metadata helps developers identify the source of issues quickly.
- **Historical Analysis:** Over time, the 'Jobs' table provides a historical record of all deployments, enabling teams to analyze trends, identify patterns, and plan improvements.
- **Rollbacks:** In the event of a critical failure, having access to historical deployment information allows teams to roll back to a previous known working version.
- **Performance Optimization:** Analyzing deployment frequency and success rates can highlight areas for optimization and enhancement in the deployment process.

Server Failure Handling:

- Implement redundancy through load balancing and multiple server

loss risks.

- Set up monitoring tools to detect server failures promptly.
- Plan for automated failover mechanisms or manual interventions in case of server failures to ensure minimal disruption to the deployment process.

Designing a high-level code deployment system requires thoughtful consideration of each component's functionality, data management, concurrency handling, and robustness against failures. By addressing these aspects, the system can efficiently manage the deployment of code across regions and environments, ensuring stability and reliability.

Assessment: https://www.bosscoderacademy.com/blog/hld-5-assessment