Q2 a) Node.Js clusters are used to run several instances of the same Node application on a single thread. Using the cluster module, we can split workloads among application threads. The cluster module enables us to build child processes that share the same server ports. To handle severe loads, we need to launch a cluster of Node.js processes, which requires several cores. Worker threads in Node.Js allows us to run JavaScript code in parallel with the main thread. Before worker threads NodeJS could execute one operation at a time. Worker threads provide the capability to perform parallel processing by creating separate threads. Worker threads are ideal for CPU-intensive activities, where parallel processing can considerably enhance performance, while Clusters can improve the scalability of networked applications by dividing incoming requests over numerous processes.

Q2b) We can use the cluster module to create multiple worker processes. Each worker runs on a separate CPU core, allowing the application to handle more requests concurrently. We can use load balancing to distribute incoming requests evenly across multiple servers or processes . Also we can use asynchronous programming concepts like Promises, async/await, and callbacks to keep the event loop non-blocking. We can implement cashing strategies to reduce load on the server. Redis for e.g can be used to frequently access data.

Q2c) Synchronous functions stops the execution of code until the current operation is complete. This means that the event loop waits for the operation to complete before proceeding to the next task. Synchronous functions can significantly degrade performance, especially in applications with high I/O demands. Asynchronous functions in Node.js allow operations to be executed without blocking the event loop. With asynchronous, multiple operations can be handled concurrently, improving the overall efficiency of the application.