

**Vertical Scaling**

* **Concept:**
  + Involves **increasing the capacity of a single server**.
  + Think of it like upgrading a car engine for more power instead of buying another car.
* **How it's done:**
  + **Adding more RAM:** Provides more memory for the server to handle increased workloads.
  + **Upgrading CPU:** Increases processing power to handle more requests concurrently.
  + **Increasing storage:** Provides more space for data, logs, and other files.
  + **Improving network bandwidth:** Allows for faster data transfer and communication.
* **Example:** Upgrading a server with a faster processor and more RAM to handle increased traffic to a website.

**Horizontal Scaling**

* **Concept:**
  + Involves **adding more servers** to your infrastructure to distribute the workload.
  + Think of it like adding more servers to a fleet of delivery trucks to handle increased orders.
* **How it's done:**
  + **Adding more servers:** Deploying additional servers to the network.
  + **Load balancing:** Distributing incoming traffic across multiple servers using a load balancer.
    - **Load balancers** can use various algorithms (e.g., round robin, least connections, least response time) to distribute traffic efficiently.
* **Example:** Adding more web servers to a cluster to handle a surge in website traffic during a sale.

**Load Balancing**

* **Purpose:**
  + Distributes incoming traffic across multiple servers in a horizontal scaling environment.
  + Prevents any single server from becoming overloaded.
  + Improves system performance, reliability, and availability.
* **Methods:**
  + **Round Robin:** Distributes requests evenly across all servers in a circular fashion.
  + **Least Connections:** Directs requests to the server with the fewest active connections.
  + **Least Response Time:** Directs requests to the server that has responded to previous requests the fastest.
  + **IP Hashing:** Directs requests from the same IP address to the same server.

**Key Considerations:**

* **Cost:** Vertical scaling can be more cost-effective initially, while horizontal scaling can become expensive as you add more servers.
* **Complexity:** Horizontal scaling introduces complexities such as server coordination, data synchronization, and maintaining consistency across multiple servers.
* **Scalability:** Horizontal scaling generally offers greater scalability potential as you can add an unlimited number of servers to handle increasing demand.
* **Fault Tolerance:** Horizontal scaling improves fault tolerance as the failure of one server does not necessarily bring down the entire system.

**In Summary:**

* **Vertical scaling** increases the capacity of individual servers.
* **Horizontal scaling** adds more servers to the system.
* **Load balancing** is crucial for effectively distributing traffic across multiple servers in a horizontally scaled environment.

The best scaling strategy depends on various factors, including the specific application, budget constraints, performance requirements, and scalability needs.

\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*

**Context Summary:**

* **Execution of Terminal Commands in JavaScript**:
  + The Node.js child\_process module provides an exec function to execute terminal commands.
  + Using the util module of Node.js, exec can be converted into a promise-based function using promisify.
* **UUID4**:
  + A unique identifier generator (uuid4) is used to create random IDs for projects.
* **Code Snippet Explanation**:
  + A new projectId is generated using uuid4.
  + A directory is created for the project using fs.mkdir.
  + The npm create vite@latest sandbox -- --template react command is executed in the newly created directory using the promisified exec function.
  + This setup happens in a backend API endpoint (api/v1/projects) triggered by a POST request.
  + The API sets up a React project template (vite) dynamically.
* **Frontend Integration with useMutation**:
  + The useMutation hook from @tanstack/react-query simplifies API calls for mutations (e.g., POST, PUT, DELETE).
  + The hook provides functionalities like:
    - **mutateAsync**: A function to execute the API mutation.
    - **onSuccess**: Callback for successful mutations.
    - **onError**: Callback for errors during the mutation.
    - Other useful states: isPending, isSuccess, error.

**Code Explanation with Context:**

**1. Backend Code with uuid4:**

javascript

CopyEdit

const projectId = uuid4(); // Generate a random, unique project ID.

console.log("New project id is: ", projectId);

await fs.mkdir(`./projects/${projectId}`); // Create a directory for the project.

const response = await execPromisified(

'npm create vite@latest sandbox -- --template react',

{ cwd: `./projects/${projectId}` } // Execute the command in the new project's directory.

);

* This snippet generates a project directory with a unique ID (projectId) and initializes a Vite-based React project in the backend.

**2. Frontend Code with useMutation:**

javascript

CopyEdit

export const useCreateProject = () => {

const { mutateAsync, isPending, isSuccess, error } = useMutation({

mutationFn: createProjectApi, // API function to create the project.

onSuccess: (data) => {

console.log("Project created successfully", data); // Handle success case.

},

onError: (error) => {

console.log("Error creating the project", error); // Handle error case.

}

});

return {

createProjectMutation: mutateAsync, // Expose mutation function.

isPending, // Expose pending state.

isSuccess, // Expose success state.

error // Expose error state.

};

};

* **Purpose**: This hook manages the API interaction for creating a project.
* **Flow**:
  1. createProjectApi is the function that triggers the POST request to the backend (api/v1/projects).
  2. The hook handles states like:
     + isPending: Shows if the request is ongoing.
     + isSuccess: Indicates successful project creation.
     + error: Captures any issues during the API call.
  3. mutateAsync is the primary function for triggering the mutation.

**Improved Context:**

This explanation provides clarity on the backend project initialization process, the dynamic project creation endpoint, and the frontend integration for triggering API calls using useMutation. It highlights both the technical implementation and the real-world use case of these tools and methods.