On the whiteboard, we'll outline the following components:

1. Load Balancer (HAproxy): The load balancer acts as a traffic distribution point and distributes incoming requests across multiple servers. We'll add HAproxy as our load balancer, which provides high availability and load balancing capabilities.
2. Web Server (Nginx): We'll include one web server running Nginx, which will handle incoming HTTP requests from users. Nginx will receive requests from the load balancer and communicate with the application servers to process and generate responses.
3. Application Servers: We'll have two application servers to handle the application's logic and dynamic content generation. The application servers will execute the codebase and communicate with the database.
4. Application Files (Codebase): The codebase contains the website's application logic and functionality. It will be deployed on both application servers, allowing for redundancy and scalability.
5. Database (MySQL): We'll have a MySQL database to store and manage the website's data. The database will be accessible by the application servers for read and write operations.

Now, let's address the specifics of this infrastructure:

- Load Balancer Distribution Algorithm: The load balancer will be configured with a distribution algorithm, such as round-robin, to evenly distribute incoming requests across the two application servers. Round-robin works by sequentially assigning each new request to the next available server in a circular order.
- Active-Active vs. Active-Passive Setup: The load balancer enables an Active-Active setup, where both application servers actively handle incoming requests simultaneously. This setup provides higher availability and load distribution. In contrast, an Active-Passive setup would involve one server actively serving requests while the other server remains in standby until the active server fails.
- Database Primary-Replica (Master-Slave) Cluster: The MySQL database will be configured as a Primary-Replica cluster. In this setup, the primary (master) node handles write operations and replicates the changes to the replica (slave) node(s). The replica nodes can handle read operations, offloading the read traffic from the primary node and improving performance.
- Difference between Primary and Replica nodes: The primary node in the database cluster handles write operations, ensuring data consistency across the cluster. It receives write requests from the application servers and replicates the changes to the replica nodes. The replica nodes primarily handle read operations, providing scalability and distributing the read traffic among multiple nodes.

Now, let's discuss the issues with this infrastructure:

1. Single Points of Failure (SPOF): The load balancer, web server, and database all represent potential single points of failure. If any of these components fail, the website's availability will be affected. To address this, you could introduce redundancy by adding

additional load balancers, web servers, and implementing database clustering techniques like multi-master setups or automated failover mechanisms.

2. Security Issues: The infrastructure lacks a firewall and HTTPS support, which can expose the system to security risks. It's essential to implement proper security measures, including firewalls to restrict unauthorized access and HTTPS to encrypt communication between the users and the web server.
3. Lack of Monitoring: Monitoring tools are necessary to ensure the infrastructure's health, performance, and detect any issues proactively. Monitoring solutions can provide insights into server health, traffic patterns, resource utilization, and alert administrators in case of any abnormalities or performance degradation.