How Nodes Work Together in a Cluster? - Part 1

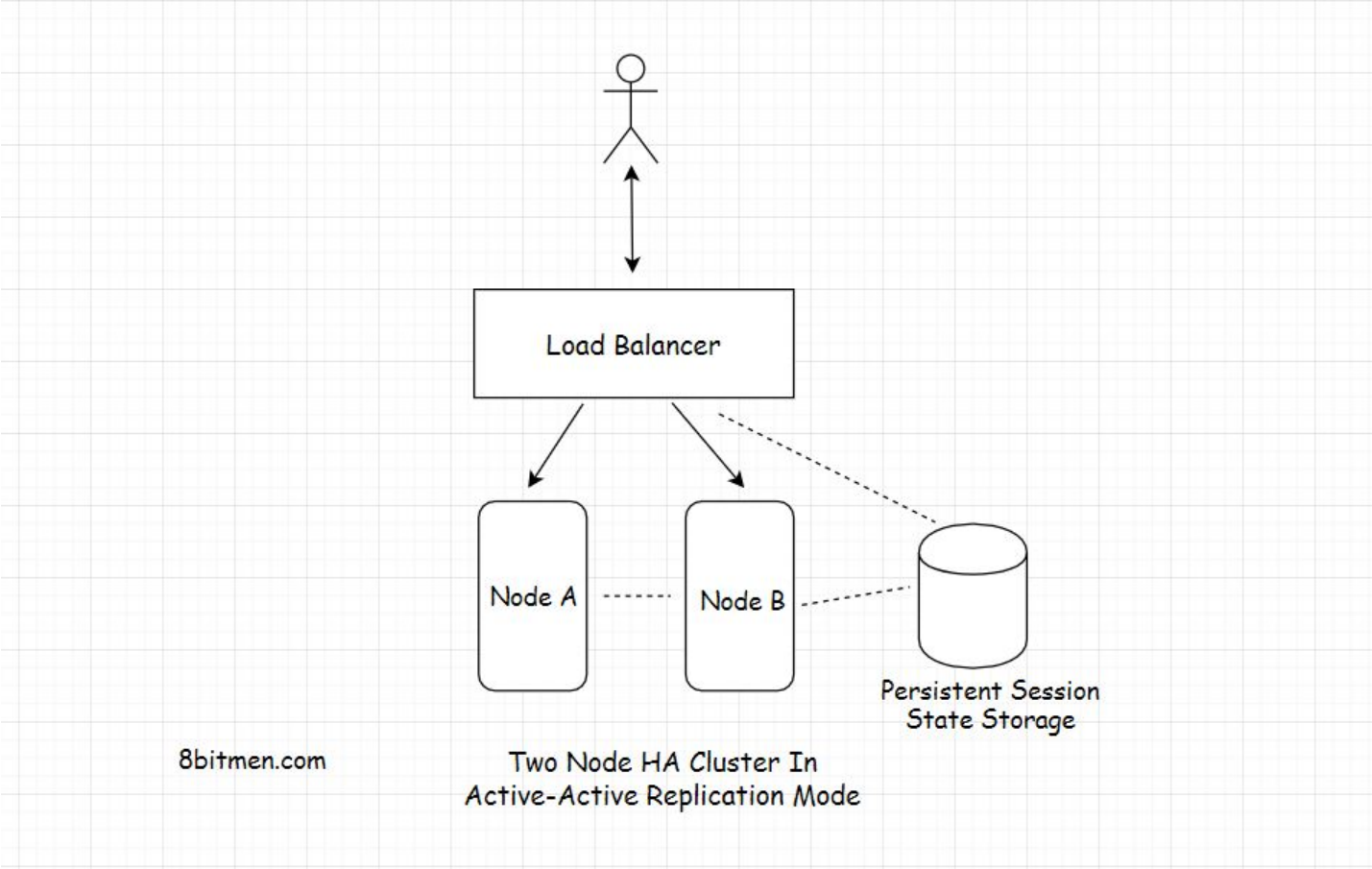
This lesson covers communication between nodes in a cluster.

We'll cover the following

Overview

In this lesson, I'll explain how nodes communicate in a cluster, maintaining a consistent state across, with a very simple example of a *two-node cluster*.

Let's assume we have a service running on a cluster consisting of just two nodes.

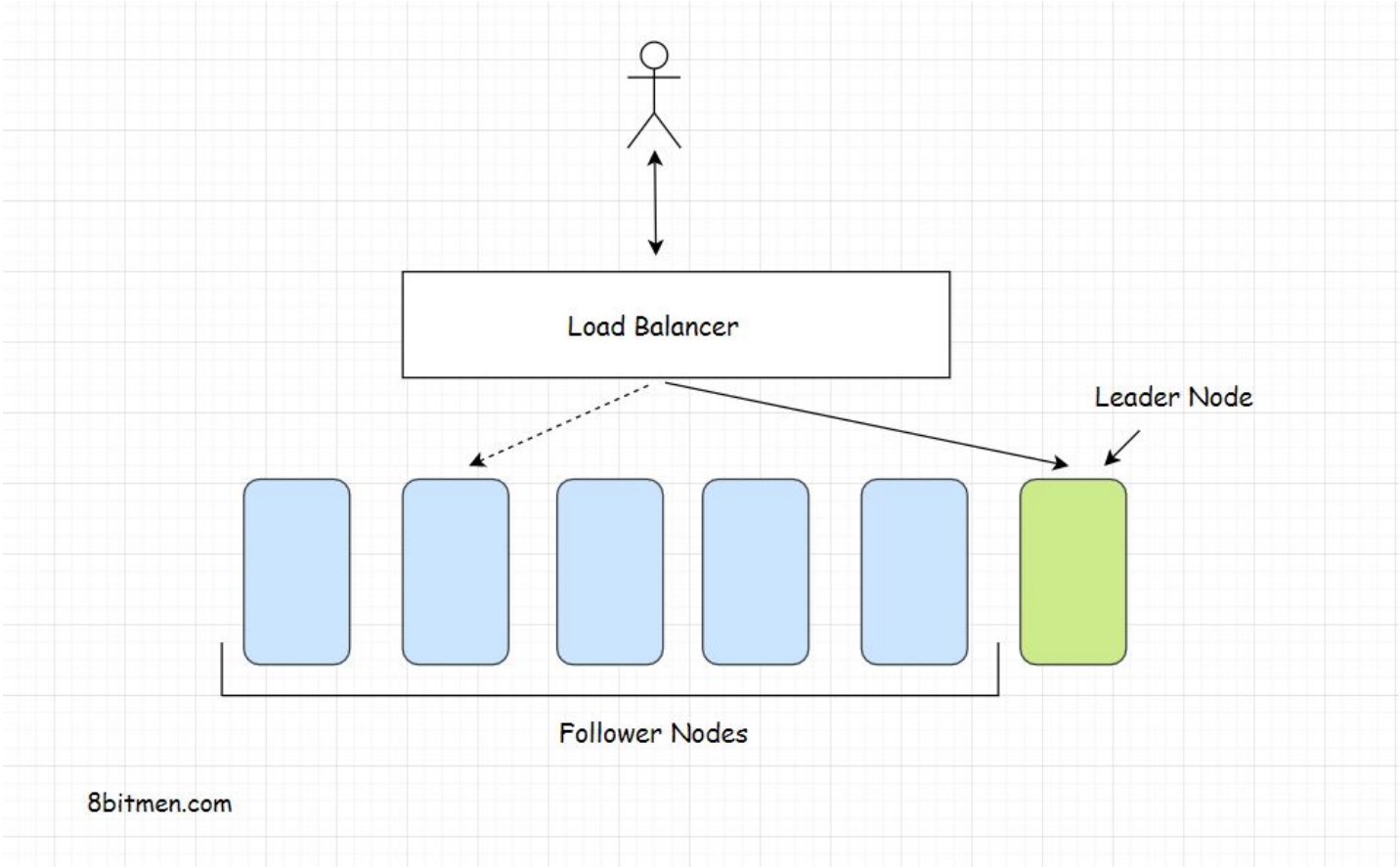


As you can see in the diagram, the load balancer routes the traffic across the two active nodes in the cluster. The nodes are aware of each other with the help of regular heartbeat signals sent to each other, primarily using *IP Multicast*.

regular heartbeat signals sent to each other, primarily using *IP Multicast communication*.

Every heartbeat has a timeout duration; if a certain node doesn't respond to the heartbeat signal within a stipulated time, it is assumed offline. The cluster always maintains a list of active servers.

Generally, every cluster has a node that acts as a leader taking care of the communication, and the onus of reaching a consensus among all the nodes in the cluster is on the cluster leader.



The leader approach comes in handy when the cluster has more than two nodes running. Also, the functionality of the cluster largely depends on how the distributed system that runs on it is designed and deployed and the underlying algorithms that are implemented.

For now, let's keep things simple. In this lesson, we'll stick with the two-node cluster. I'll discuss more on node coordination in the upcoming lessons.

Node coordination scenario

We'll now walk through a scenario where we have the two nodes in the cluster *Node A* and *Node B* up and running, ready to receive user requests.

When the user sends a request to the backend, the load balancer routes the request

to *Node A*. The job of the load balancer is to distribute the traffic intelligently

across the nodes in the cluster. The web server that is *Node A* creates a user session, say *US_1*, and begins processing the user request.

During the request processing, assume a technical glitch brings down *Node A*, and the server goes offline. The cluster detects that *Node A* has gone offline; it then re-routes the request to *Node B*. *Node B* is a replica of *Node A*.

Two nodes are used in the cluster in *active-active replication* mode to make the service highly available. So, in case one of the two nodes goes down, the other can take over.

And that's what exactly happens. *Node A* goes down due to a failure, but the service is still up, and *Node B* takes over, handling the traffic load of the service.

Persistent session storage

Both nodes in the cluster persist the session information in in-memory persistent session storage. Persistent session storage is useful because if any of the nodes go down, another node can take over by accessing the session information of the offline node from the memory. It then can return the response to the end-user without them noticing anything.

Let's continue this discussion in the next lesson.