Tiny URL System design:

The user will provide input as a string of URL to get it shortened as a part of request body. The request will go to the network load balancer which will route the request to individual running API instances. The API has single endpoint /api with POST method to create new URL from the algorithm and with GET method to fetch the original URL. After successful creation of tiny URL, the tiny URL being the primary key and original URL being the value is stored in the Database. The GET endpoint will fetch the original URL for the key being passed and redirect to original URL. The GET endpoint is protected by distributed cache to reduce the database calls.

How does your system ensure that 2 URLs never map to the same shortened URL?
-> The original URL sent by user is being converted to a tiny URL consisting of 62 chars (a-z,A-Z,0-9) with steps as Original URL -> md5(OURL) -> base62(md5 hash) . This bas62 string is the tiny URL to be stored in database as primary key to OURL. It will be always unique for a string.

How will you ensure the system is very low latency?
-> GET endpoint is protected by distributed cache mechanism which will reduce the backend calls and subsequently the latency. The cache has no expiry.

What will happen if the machine storing the URL mapping dies?
-> There will be a database replica maintained and updated frequently. If the current machine dies it will be switched over to its replica. Frequent database snapshots are also created for recovery.

How do you make sure your system is consistent? This is to say, if I have shortened a URL, given the shortened URL, my system should always return the original URL no matter when I call your system.
-> The subsequent GET response will always be served from the cache. If the cache is down circuit will break and hits will go to actual database until cache comes up.

How do you make sure that your service never goes down?
-> If one of the API goes down NLB will not send any request to unhealthy instances. Autoscaler will maintain at least 3 replicas running in healthy state. Autoscaler will scale up the instances if CPU utilization goes beyond 60% or memory utilization goes beyond 90%. If NLB goes down there will be a backup NLB replacing it.