Optional failure modes:

1. Data loss and backup – There is no real database used. This means that if for some reason one of the endpoints fails before we extract the information from it, the information will be lost. In addition, there is no backup process of the endpoints or the workers. In order to deal with this, we need to add a caching system and a database.
2. Workers failure handling – we don't handle the failure of workers well enough. If they fail in the middle of a job, we lose this job information. In addition, the endpoint that is responsible for this worker might not know about the worker failure. We can solve this by making sure that no task is lost because of a worker failure by using a database or by the endpoint keeping the jobs until it receives the job result from the worker. In addition, to make sure the endpoint knows about a worker failure we can create a function on the endpoint that receives "alive" messages from the workers and will know a worker is dead if no message is received for some determined time frame.
3. Logging – There is no logging system that can help us detect and track error and other important things happening in the system. To solve this, a logging system can be added that will collect data from the endpoints and workers and save it in some DB, like ELK.
4. Cache – There is no use of cache in the system. One of the things that it affects is that we don't save results of the sha512 computing. This means that we will potentially recompute the same data and number of iterations multiple times instead of computing it once and caching it for future use. To solve this problem, you need to create a cache system as part of the service.
5. Monitoring – we didn't define any monitoring procedure like cloudwatch that can help us monitor errors and system flows. To solve this, we can use cloudwatch or any other monitoring service.
6. Network security – The endpoint server is open to the world on port 5000. This could lead to unwanted requests and to a potential system breach. To solve this, some network monitoring can be applied to check the incoming and outgoing requests. In addition, we should create strict FW rules that allow only the needed IPs and ports.
7. DDOS – there is no protection against DOS or DDOS attacks. A flood of requests can potentially bring the system down. This can be solved by restricting the number of requests that will be processed from one IP in some time frame and also restricting the amount of requests that will be processed in some time frame in general.
8. High system load – We don't handle the case of high CPU usage or high disk usage. This can be solved by monitoring disk and CPU usage and in the case of high usage to give more CPU/disk space or to make sure to not get to a system overload. In addition, we don't use a node that deals with splitting the work between the endpoints. Instead, we send the requests directly to one of the nodes. This can cause the endpoints to overload and fail. A possible solution for this would be to create some "orchestrator" nodes that will be in charge of splitting the work between the endpoints and will handle endpoint failures.
9. Network failure – If communication to one of the nodes fails, we don't handle it well. We will keep on working and receiving jobs, but we won't try to restart the node or try to reconnect it. This can be solved by using some network failure handling system or by monitoring the errors using CloudWatch and then restarting the connection in case of a network failure.
10. Testing – There is no testing framework as part of our system. This can cause errors when uploading new versions without properly testing them. A solution would be to create a testing framework that will be used every time a new version is uploaded.