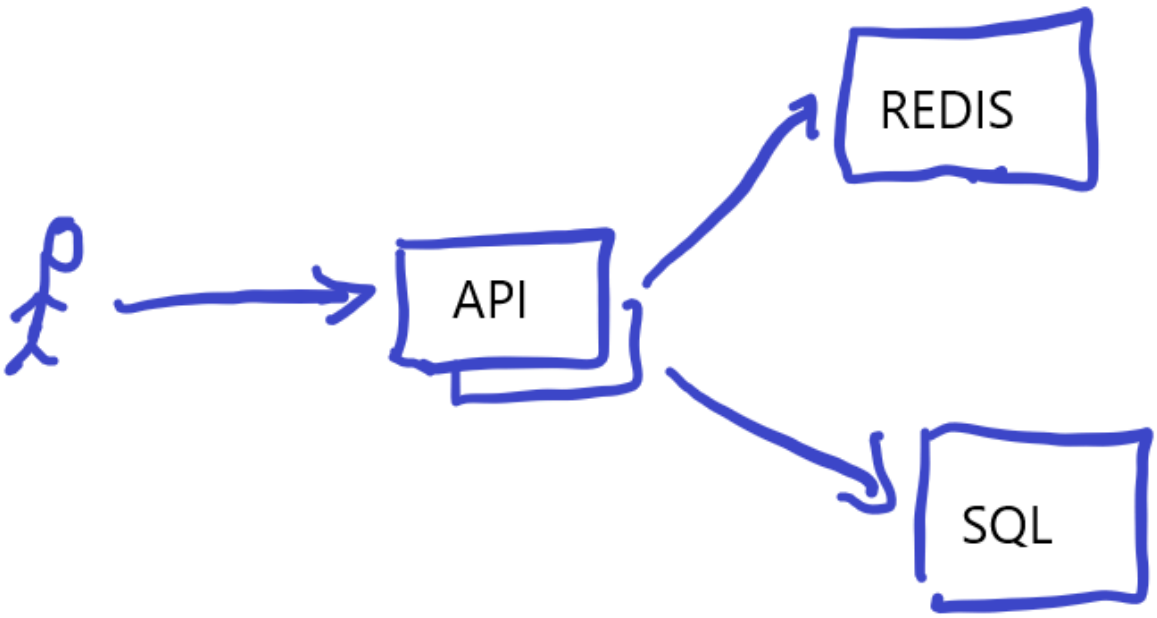
Notes Architecture:



- The current solution runs with Docker and can be easily transitioned to Kubernetes (K8s).
- Currently API is run on a single instance but can easily scale with a LB in front.
- A gateway with a rate limiter can be placed in front of the API to ease on the spam.
- Redis and SQL instances need to be moved to the cloud, hard to scale even with k8s if run on containers.
- Configurations need to be moved to App Configuration and Key Vault.
- API can scale as needed since there is no state.
- SQL can be replicated for fault tolerance.
- SQL is chosen for this implementation because of the write to read ratio 1:10 meaning single write to store the link and multiple reads from the same or different user. SQL uses B-tree which are costly for writes since they propagate to disk while faster for reads. NoSQL dbs use LSM tree and SS tables which are write optimized. In case of throughput issues you can always use the read replicas for fetching.
- Redis is used for easing the load on the SQL. Currently the application uses default configuration for TTL with LRU eviction but it can be easily modified. LRU will make sure that popular urls will be kept for a longer period. Multiple replicas can be added to a Redis cluster / sharding if experiencing performance issues.
- Write Around Cache is used without the invalidation step since there are no updates, only appends.
- SQL index is used on the Generated Suffix for nlogn retrieval of a record preventing linear scan.

- SQL misses are expensive, a Bloom filter may be considered before querying the database.