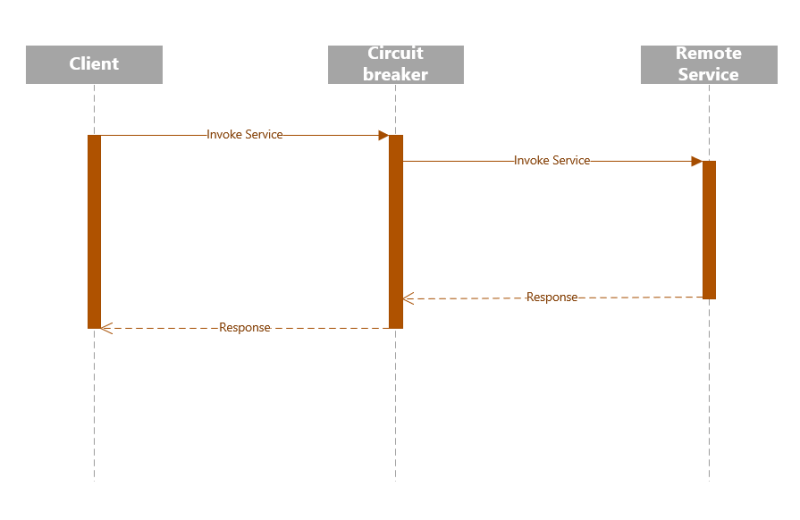
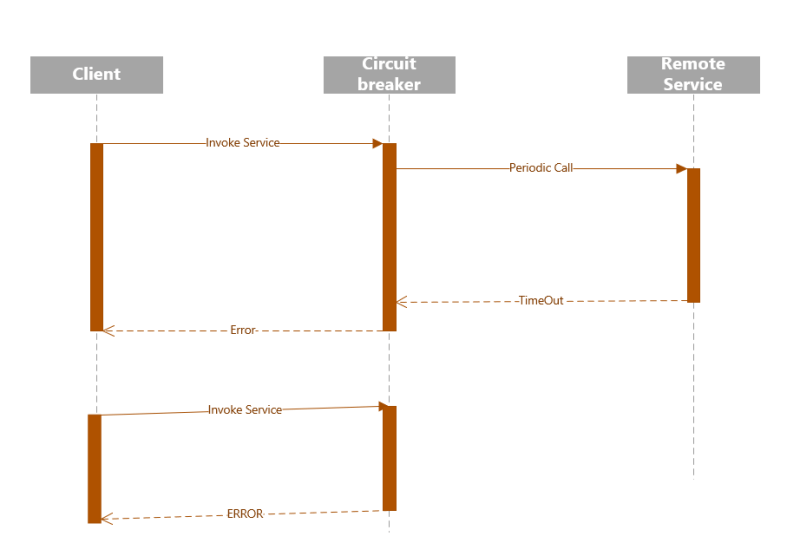
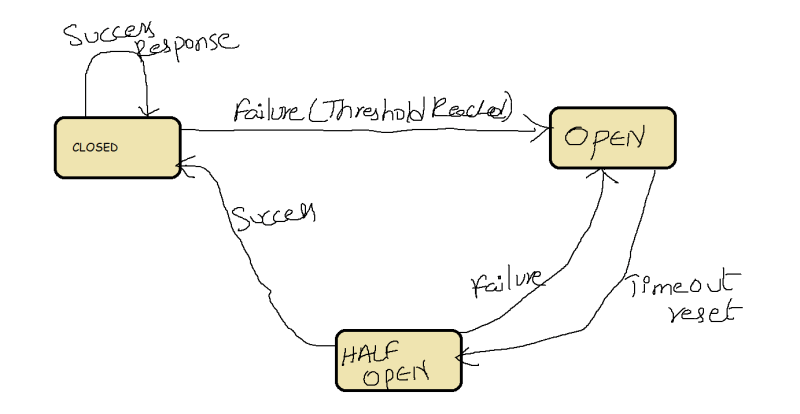
**Circuit Breaker for Microservices**

* As with an electricity circuit breaker (mcb), when the service encounters number of failures, the circuit breaker trips down for a particular duration. In that period, all the attempts made to invoke the failing service will fail immediately
* Once the configured duration is over, the circuit break will allow certain number of requests to pass through and monitor whether the requests succeed. If there are failures then circuit breaker continues to be tripped for a given period
* Circuit breaker has the following states

Closed: The circuit breaker being in a CLOSED state means everything is working fine and all calls pass through remote services



Open: Once the number of timeouts reaches a predetermined threshold in the circuit break, it trips the circuit breaker to the OPEN state. In the OPEN state, the circuit breaker returns an error for all call to the service without making calls to the remote service 

Half Open: After a certain duration, the circuit switches to a HALF-OPEN state to test if the underlying problem still exists. The circuit breaker uses a mechanism to make a trail call to the remote service periodically to check if it has recovered. If the call to the Remote Service failed, then circuit breaker remains in the OPEN state. IF the call returns success then the circuit breaker switches to the CLOSED state. The circuit breaker then returns all external calles to the service with an error during HALF-OPEN state 

**Do I really need a service mesh?**

* A service mesh provides a consistent way to connect, secure and observe microservices
* If you are deploying your first or second microservice, you donot have to use service mesh but rather focus on k8s
* If you have an existing application architecture that provides the observability, security and resilience you need, then you are already in good place
* The exact point at which service mesh clearly outweigh cost varies from organization to organization.
* Generally, teams often realize they need a consistent approach once they have five or six microservices, many users push to a dozen or microservices, before they notice increasing cost of utility code and the increasing complexity of differences across the applications. Best thing observed is try using a service when your application has more or less half a dozen microservices.
* Here are some states that may reduce or eliminate the urgency to use a service
  + All of your microservices are written in one language ("monoglot") by developers in your organization, building from common framework
  + You have partially or totally monolithic architecture where application logic is built into one or two containers instead of several
  + You use application protocols that not served by existing service mesh (so usually NOT HTTP, HTTP2, gRPC)