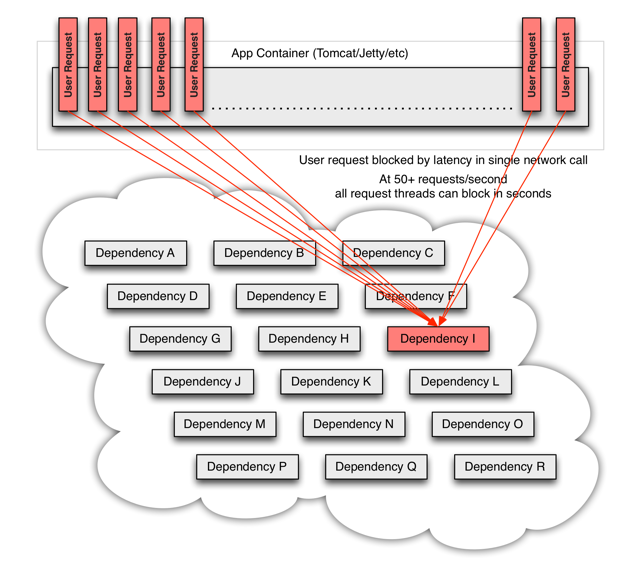
**Hystrix:**

Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.

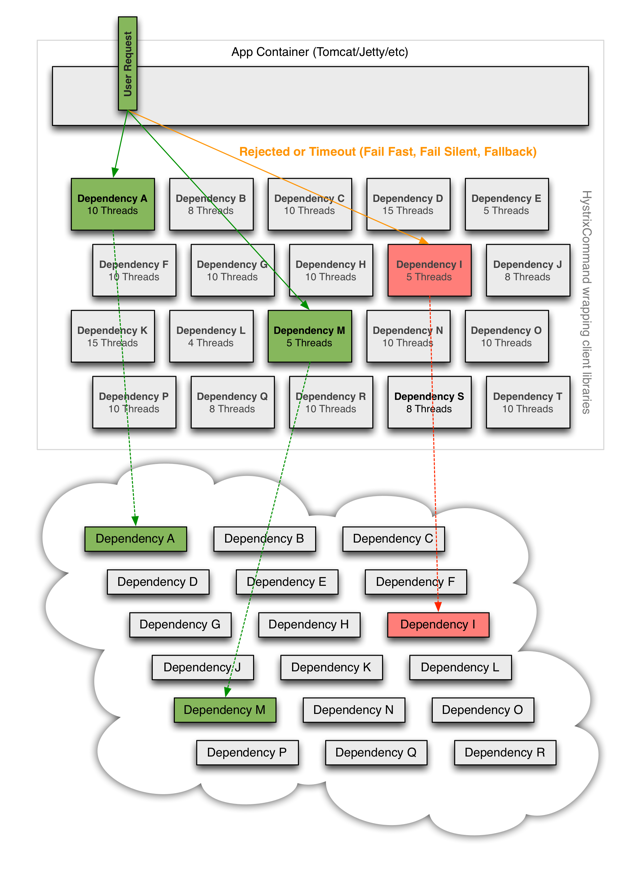
In a distributed architecture like microservices, one service may require to use other services as dependencies to accomplish his work. Every point in an application that reaches out over the network or into a client library that can potentially result in network requests is a source of failure. Worse than failures, these applications can also result in increased latencies between services. And this leaves us to another big problem, suppose you are developing a service on a Tomcat which will open two connections to two services, if one of this service takes more time than expected to send back a response, you will be spending one thread of Tomcat pool (the one of current request) doing nothing rather than waiting an answer. If you don't have a high traffic site this may be acceptable, but if you have a considerable amount of traffic all resources may become saturated and block the whole server.

A schema from this scenario is provided on Hystrix wiki:

[](https://github.com/Netflix/Hystrix/wiki/images/soa-3-640.png)

The way to avoid previous problem is to add a thread layer which isolates each dependency from each other. So each dependency (service) may contain a thread pool to execute that service. In Hystrix this layer is implemented by HystricxCommand object, so each call to an external service is wrapped to be executed within a different thread.

A schema of this scenario is provided on Hystrix wiki:

[](https://github.com/Netflix/Hystrix/wiki/images/soa-4-isolation-640.png)

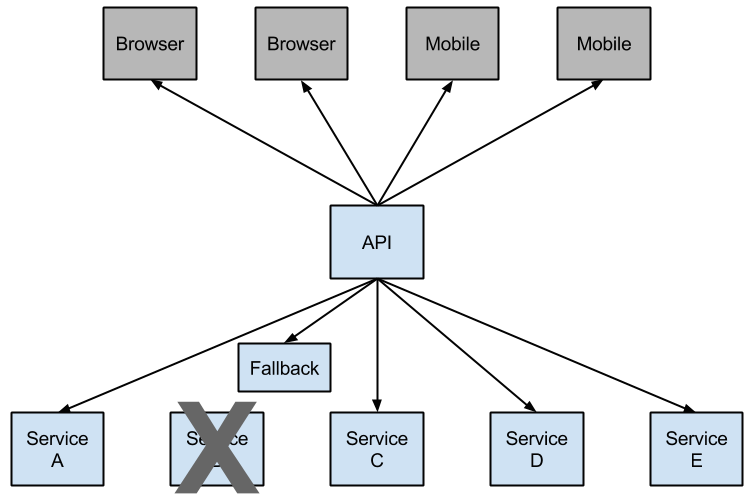
But also Hystrix provides other features:

Each thread has a timeout so a call may not be infinity waiting for a response.

Perform fallbacks wherever feasible to protect users from failure.

Measure success, failures (exceptions thrown by client), timeouts, and thread rejections and allows monitorizations.

Implements a circuit-breaker pattern which automatically or manually to stop all requests to an external service for a period of time if error percentage passes a threshold.



**Example of HystricxCommand** for circuit breaker:

**Step 1:** Dependencies required

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix</artifactId>

</dependency>

**Step 2:** Annotations required in application.java

@EnableCircuitBreaker

Step 3: Add fallback code to endpoints

@HystrixCommand(groupKey = "notification-service", fallbackMethod = "fallbackforgetEmployeeJSONById")

@RequestMapping(value = "/hr/employee/{id}", produces = MediaType.APPLICATION\_JSON\_VALUE, method = RequestMethod.GET)

public ResponseEntity<Employee> getEmployeeJSONById(@PathVariable Long id) {

//some code

}

public ResponseEntity<Employee> fallbackforgetEmployeeJSONById(@PathVariable Long id) {

return new ResponseEntity("from fall back of getEmployeeJSONById", HttpStatus.OK);

}

**Example to use Dashboard:**

Step 1: dependencies required

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>

</dependency>

**Step 2:** Annotation required

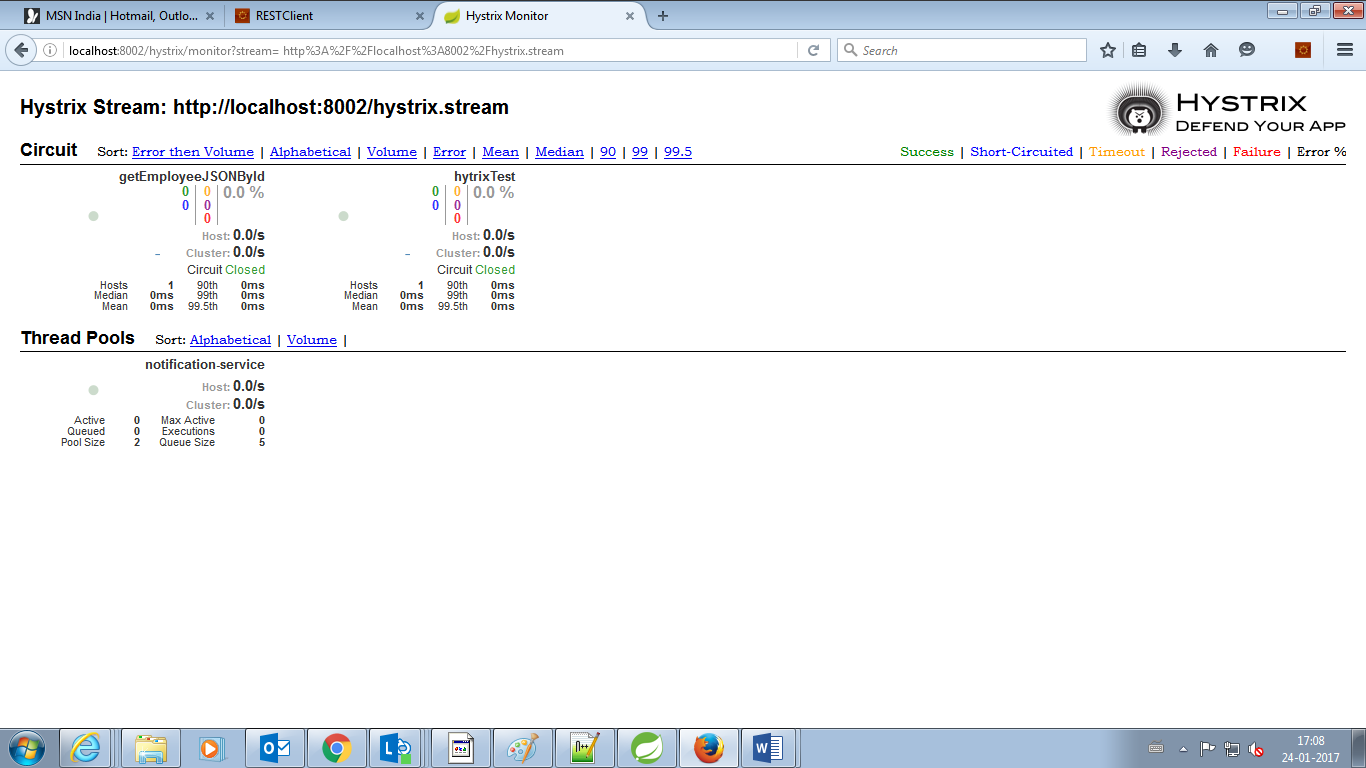
@EnableHystrixDashboard

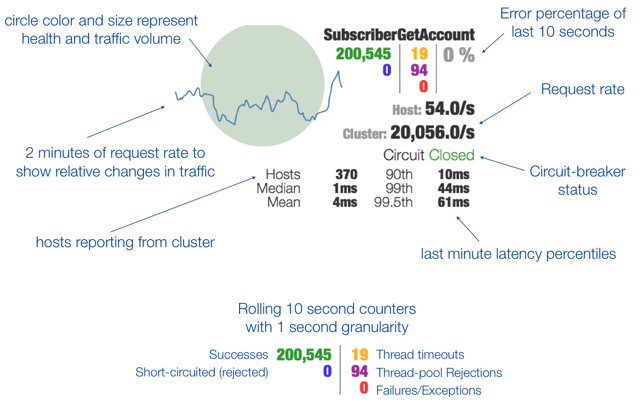
Step 3: Hit all the endpoints which are protected by fallback code at least

Step 4: Now we can try the below endpoints

<http://localhost:port/health>

http:// localhost:port/hystrix ---- here enter http:// localhost:port/hystrix.stream to check the dashboard where we can see the monitoring of various circuits





**Turbine:**

**Step 1:** dependencies required

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-turbine</artifactId>

<exclusions>

<exclusion>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

</exclusion>

</exclusions>

</dependency>

**Step 2:** Annotation required

@EnableTurbine

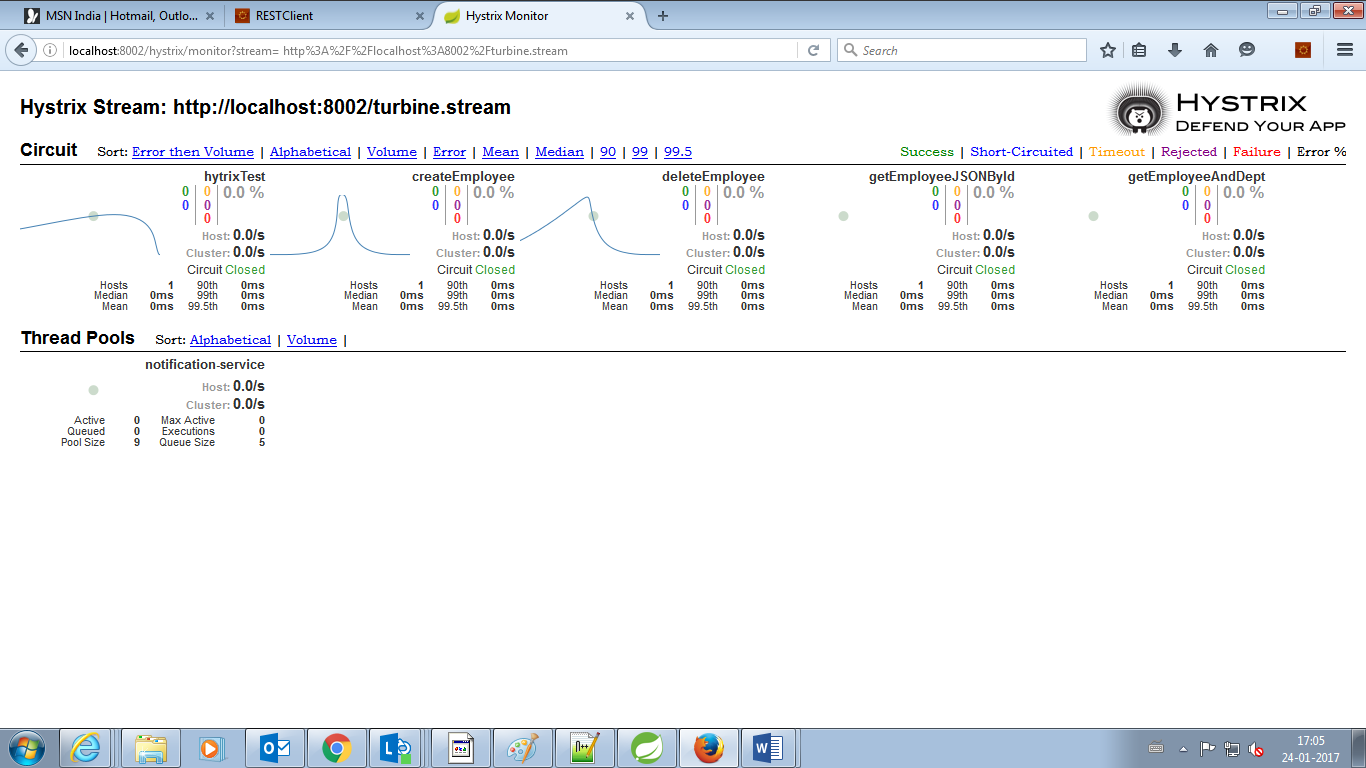
**Step 3:** Add the below configuration in application.yml file in the hystrix turbine server.

turbine:

clusterNameExpression: new String("default")

appConfig: employeeConfigEurekaClient,hrConfigEurekaClient

appConfig --- should contain all the services name which need to be monitored in single dashboard using turbine. Turbine will check with Eureka to find the services instance.



**Note:**

* In order to view hystrix dashboard/turbine every time after the server starts each endpoint must be hit at least once.
* And evert endpoint whose circuit breaker we are monitoring through dashboard/turbine should have a fallback method
* Fallback method signature should match with respective endpoint method