Head pattern is to have resources of your service isolated within pools so that if one fails , the other will continue working.

Service handles 50 concurrent inbound HTTP request simultaneously.

Service provides 2 set of REST API end points.v1 and v2.

50 request to v1 and 5 request to v2.

Use Resilince4J library for implementing bulkhead pattern

Io.github.resilience4J

Resiliaence4J-spring-boot2

Version : 1.7.1

Pool of concurrent permits for v1

Resilience4J:

Bulkhead:

Instances:

V1:

Max-concurrent-calls:40

Controller Class :

@RestController  
@RequestMapping(**"/v1"**)  
**public class** V1Controller {  
**private static final** Logger **LOGGER** = Logger.getLogger(V1Controller.**class**.getSimpleName());  
  
@GetMapping  
**public void** performAction() **throws** InterruptedException {  
**LOGGER**.log(Level.**INFO**, **"V1 endpoint operation is in progress"**);  
Thread.sleep(1000);  
}  
}

Create aspect class to control calls to our v1 end points:

@Aspect  
@Component  
**public record** BulkheadRestControllerAspect(BulkheadRegistry bulkheadRegistry) {  
  
@Around(**"within(com.ponomarev.example.bulkhead.rest.v1..\*)"**)  
**public** Object proceedInternal(ProceedingJoinPoint proceedingJoinPoint) **throws** Throwable {  
**return** process(proceedingJoinPoint, "V1");  
}

**private** Object process(ProceedingJoinPoint proceedingJoinPoint, String bulkheadName) **throws** Throwable {  
Bulkhead bulkhead = **bulkheadRegistry**.bulkhead(bulkheadName);  
**return** bulkhead.executeCheckedSupplier(proceedingJoinPoint::proceed);  
}  
}

By default , application will return HTTP response status code -500

Lets make it 429 by creating rest controller advice class

@RestControllerAdvice  
**public class** RestErrorHandler {  
@ExceptionHandler(BulkheadFullException.**class**)  
**public** ResponseEntity<String> bulkheadException(BulkheadFullException exception) {  
**return** ResponseEntity.status(HttpStatus.**TOO\_MANY\_REQUESTS**)  
.body(exception.getMessage());  
}  
}

**max-concurrent-calls**: 5

We will call our endpoint 30 times and expect to have 5 HTTP responses with status codes 200 and 5 with status code 429.

|  |
| --- |
| @EnableAutoConfiguration |
|  | @SpringBootTest( |
|  | webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT |
|  | ) |
|  | @AutoConfigureMockMvc |
|  | @ActiveProfiles("test") |
|  | public class BulkheadV1Test { |
|  | @Autowired |
|  | private MockMvc mockMvc; |
|  |  |
|  | @Test |
|  | void bulkheadTest() throws Exception { |
|  | int numberOfRequests = 30; |
|  | ExecutorService executor = Executors.newFixedThreadPool(numberOfRequests); |
|  | List<CompletableFuture<Integer>> cfList = new ArrayList<>(); |
|  |  |
|  | for (int i = 0; i < numberOfRequests; i++) { |
|  | cfList.add(CompletableFuture.supplyAsync(() -> performRequest(), executor)); |
|  | } |
|  |  |
|  | int countOK = 0; |
|  | for (CompletableFuture<Integer> cf : cfList) { |
|  | if (cf.get(1, TimeUnit.SECONDS) == HttpStatus.OK.value()) { |
|  | countOK++; |
|  | } |
|  | } |
|  |  |
|  | // 5 - is max-concurrent-calls in application-test.yml |
|  | Assertions.assertEquals(5, countOK); |
|  | } |
|  |  |
|  | private int performRequest() { |
|  | try { |
|  | MvcResult mvcResult = mockMvc.perform(get("/v1")) |
|  | .andDo(print()) |
|  | .andReturn(); |
|  | return mvcResult.getResponse().getStatus(); |
|  | } catch (Exception e) { |
|  | throw new RuntimeException(e); |
|  | } |

Resilience4J provides 2 implementation of bulkhead pattern

* Semaphorebulkhead are better for synchronous calls
* FixedThreadPoolBulkhead for aync calls