Redis Java Client Jedis Connection Pool Configurations

Jedis Connection Pool configuration can be specified inside your `application.properties` file of the spring application. For example, to set maximum active threads to 12, add "`spring.redis.jedis.pool.max-active=12`" . Refer to this redis-developer project properties. https://github.com/redis-developer/redis-microservices-demo/blob/master/caching-service/src/main/resources/application.properties#L12

The specific properties names are documented on the spring docs. https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html#spring.redis.jedis.pool.max-active

Incase of non spring applications, the same configurations can be configured via JedisPool object Configuration.
https://javadoc.io/doc/redis.clients/jedis/2.9.0/redis/clients/jedis/JedisPool.html

The default set by Jedis client can be seen here
https://github.com/redis/jedis/blob/master/src/main/java/redis/clients/jedis/JedisPoolConfig.java

Table below covers the default and recommended Jedis connection pool configurations inline with the recommendations notes.

Configuration	Description	Default	Recommended	Recommendations
`DEFAULT_MAX_TOTAL = 8;`	Maximum active connections to Redis instance. Sets the cap on the number of	8	use case dependent,	For very low latency applications, a low ratio of maxTotal to concurrent users should work fine. However as the latency

	objects that can be allocated by the pool at a given time. Use a negative value for no limit.		refer to recommendation notes	increases, so too should the value for maxTotal. When choosing a value, consider how many concurrent calls into Redis under load. Each connection does have some memory and CPU overhead, so setting this to a very high value may have negative side effects. Recommended number of connections to start with for a mid scale application is 100.
`DEFAULT_MAX_IDLE = 8;` `jedisPoolConfig.setMaxIdle(-1);`	Maximum number of "idle" connections in the pool. In other words, the number of connections to Redis that just sit there and do nothing.	8	depends on application traffic, configure it based on recommendation notes	maxTotal should not be less than maxIdle since maxIdle caps maxTotal. This can be configured the same as maxTotal to avoid connection ramp-up costs when your application has many bursts of load in a short period of time. If a connection is idle for a long time, it will be evicted until the idle connection count hits minIdle.
`DEFAULT_MIN_IDLE = 0;` `jedisPoolConfig.setMinIdle(-1);`	Minimum number of idle connections to Redis - these can be seen as always open and ready to serve. In other words, target for the minimum number of idle connections to maintain in the pool.	0	Use case dependent, refer to recommendation notes	This is the number ready for immediate use that remains in the pool even when load has reduced. When choosing a value, consider your *steady-state* concurrent requests to Redis. For instance, if your application is calling into Redis from 10 threads simultaneously, then you should set this to above 10. If the configured value of minIdle is greater than the configured value for maxIdle then

				the value of maxIdle will be used instead.

`DEFAULT_TEST_ON_BORROW = false`
 	
`jedisPoolConfig.setTestOnBorrow(false);`
	Send Redis PING on borrow. Tests whether connection is dead when connection retrieval method is called. If the connection fails to validate, it will be removed from the pool and destroyed, and a new attempt will be made to borrow an object from the pool.	false	false	Recommendation is false, reason - additional RTT on the conn exactly when the app needs it, reduces performance.

`DEFAULT_TEST_ON_CREATE = false`
`jedisPoolConfig.setTestOnCreate(false);`
	testOnCreate – true if newly created objects should be validated before being returned from the borrowObject() method	false	false	Same as above

`DEFAULT_TEST_ON_RETURN = false`	Tests whether connection is dead when returning a connection to the pool. true if objects are validated before being returned from the borrowObject() method	false	false	Same as above

`DEFAULT_TEST_WHILE_IDLE = false;`
 	
`jedisPoolConfig.setTestWhileIdle(true);`
 	
`jedisPoolConfig.setTimeBetweenEvictionRunsMillis(1000);`
	Tests whether connections are dead during idle periods.
	
	testWhileIdle – true so objects will be validated by the evictor
	
	If the connection fails to validate, it will be removed from the pool and destroyed. Note that setting this property has no effect unless the idle object evictor is enabled by setting timeBetweenEvictionRunsMillis to a positive value.
		false	True & eviction set to 1000 ms	Send periodic Redis PING for idle pool connections. Recommendation (True), reason - test and heal connections while they are idle in the pool.

`DEFAULT_BLOCK_WHEN_EXHAUSTED = true;`
 	
`jedisPoolConfig.setBlockWhenExhausted(true);`
	blockWhenExhausted – true if borrowObject() should block when the pool is exhausted(the maximum number of "active" objects has been reached)	true	true	

`DEFAULT_MAX_WAIT_MILLIS = -1L;`
 	
`jedisPoolConfig.setMaxWaitMillis(-1);`
	Maximum time, in milliseconds, to wait for a resource when exhausted action is set
	
	. Sets the maximum amount of time (in milliseconds) the
		-1	-1	Generally, if your application is going to retry the connection to the database indefinitely anyway, you are ok with this setting. However if your app doesn't, then set it to a more reasonable number.

	borrowObject() method should block before throwing an exception when the pool is exhausted and getBlockWhenExhausted is true. When less than 0, the borrowObject() method may block indefinitely.			
`DEFAULT_TIME_BETWEEN_EVICTION_RUNS_MILLIS = -1L;` `jedisPoolConfig.setTimeBetweenEvictionRunsMillis(1000);`	Idle connection checking period. Sets the number of milliseconds to sleep between runs of the idle object evictor thread. When positive, the idle object evictor thread starts. When non-positive, no idle object evictor thread runs. Params: timeBetweenEvictionRunsMillis – number of milliseconds to sleep between eviction runs.	-1	1000	
`DEFAULT_NUM_TESTS_PER_EVICTION_RUN = 3;` `setNumTestsPerEvictionRun(-5);`	Maximum number of connections to test in each idle check. numTestsPerEvictionRun – max number of objects to examine during each evictor run. When positive, the number of tests performed for a run	3	-5	

	will be the minimum of the configured value and the number of idle instances in the pool. When negative roughly one nth of the idle objects will be tested per run(getNumIdle/abs(getNumTestsPerEvictionRun))			

`jedisPoolConfig.setLifo(true)`
 	
`DEFAULT_LIFO = true;`
	whether the pool has LIFO (last in, first out) behaviour with respect to idle objects - always returning the most recently used object from the pool, or as a FIFO (first in, first out) queue, where the pool always returns the oldest object in the idle object pool	true	true	

`jedisPoolConfig.setFairness(false);`	True means that waiting threads are served as if waiting in a FIFO queue.	false	false	

Fine-tuning Recommendations

1. Use try-with-resources so the connection is automatically closed. Sample code in the block below. If connections are not properly closed, expect to see an increase in the

number of connections on the server side until no more connections can be created. This is a connection leak.

```
public Site findById(long id) {
   try(Jedis jedis = jedisPool.getResource()) {
       String key = RedisSchema.getSiteHashKey(id);
       Map<String, String> fields = jedis.hgetAll(key);
       if (fields == null || fields.isEmpty()) {
           return null;
       } else {
           return new Site(fields);
       }
   }
}
```

2. Get Pool Usage statistics. Debugging performance problems due to JedisPool contention issues will be easier if the pool usage is logged. If you see exceptions when trying to get a connection from the pool, you should log usage stats.

```
public static String getPoolUsage(JedisPool jedisPool) {
   int active = jedisPool.getNumActive();
   int idle = jedisPool.getNumIdle();
   int total = active + idle;
   return String.format("Pool Usage: Total=%d, Active=%d, Idle=%d,
Waiters=%d", total, active, idle, jedisPool.getNumWaiters());
}
```

References

1. https://support.redislabs.com/hc/en-us/articles/360008095580-JedisPool-recommended-configuration

2. https://support.redislabs.com/hc/en-us/articles/360018833813-Connection-pool-management-with-Lettuce-and-Jedis-

3. https://gist.github.com/JonCole/925630df72be1351b21440625ff2671f

4. https://github.com/apache/commons-pool/blob/master/src/main/java/org/apache/commons/pool2/impl/GenericObjectPoolConfig.java

5. https://github.com/spring-projects/spring-data-redis/blob/master/src/main/java/org/springframework/data/redis/connection/PoolConfig.java

6. https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisProperties.java