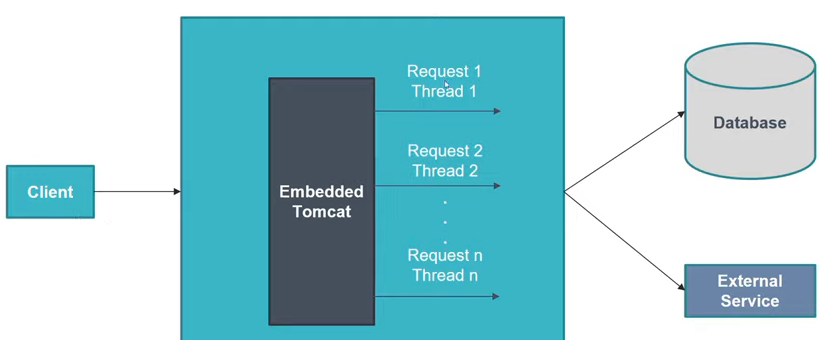
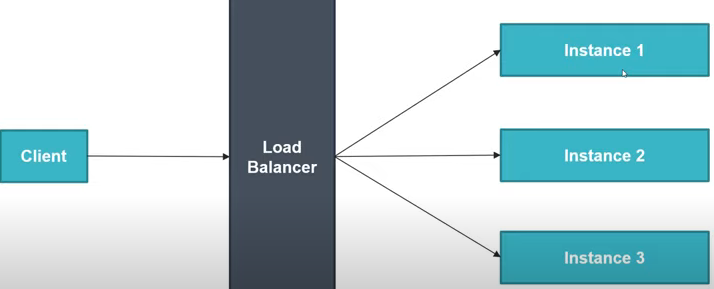
* Traditional REST APIs works in a ***Thread Per Request Model***.

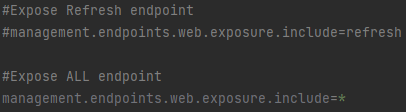


* Every request is assigned with one thread and that thread is dedicated to that request only. These threads are called worker threads. If there is any DB call or any external service call, the worker thread stays idle, until the it receives response. Once the request is complete, the thread is free and available to serve another request.
* Suppose we are having a lot of requests to our Spring Boot REST application and our dedicated worker threads are idle waiting on the response from external systems. Worker threads will not be able to serve the requests, even if they are idle. This can be a problem with Thread Per Request Model. Possible solutions can be:
  + Increase the worker thread count. In Spring Boot, by default, the embedded Tomcat can have ***maximum 200 worker threads***. This means, the Spring Boot REST application can handle maximum 200 concurrent requests at the same time. We can configure/increase this worker thread number using property: ***server.tomcat.threads.max***. But there is limit to it and we can’t just increase the worker thread count as many as we want. It may lead to OutOfMemory issues.
  + We can also have solutions as below, with multiple application instances, a load balancer distributing the requests across instances. But increasing instances may cost more on the cloud.



* + Another better approach is to create Reactive REST APIs using Spring Webflux.
* ***How to expose all the Actuator endpoints?***

By default, Actuator exposes few endpoints, such as health, health-path. To expose all the endpoints, we need below configuration in application properties file.



* Ways to run a Spring-Boot application:
  + From project directory we can run:



* + We can also use the below command.

