***[Spring Boot Actuator](http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/" \l "production-ready)?***

[Spring Boot Actuator](http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#production-ready) is a sub-project of Spring Boot. It adds several production grade services to your application with little effort on your part. In this guide, you’ll build an application and then see how to add these services.

***Spring Cloud?***

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

Spring Cloud builds on Spring Boot by providing a bunch of libraries that enhance the behaviour of an application when added to the classpath. You can take advantage of the basic default behaviour to get started really quickly, and then when you need to, you can configure or extend to create a custom solution.

***Spring Cloud Netflix?***

Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms.

With a few simple annotations you can quickly enable and configure the common patterns inside your application and build large distributed systems with battle-tested Netflix components.

The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client Side Load Balancing (Ribbon)..

***Spring HATEOAS?***

Spring HATEOAS provides some APIs to ease creating REST representations that follow the [HATEOAS](http://en.wikipedia.org/wiki/HATEOAS) principle when working with Spring and especially Spring MVC. The core problem it tries to address is link creation and representation assembly.

<dependencies>

<dependency>

<groupId>org.springframework.hateoas</groupId>

<artifactId>spring-hateoas</artifactId>

<version>0.20.0.RELEASE</version>

</dependency>

</dependencies>

**HATEOAS**, an abbreviation for **Hypermedia As The Engine Of Application State**, is a constraint of the [REST application architecture](https://en.wikipedia.org/wiki/Representational_state_transfer) that distinguishes it from most other network application architectures. The principle is that a client interacts with a network application entirely through [hypermedia](https://en.wikipedia.org/wiki/Hypermedia) provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia. By contrast, in some [service-oriented architectures](https://en.wikipedia.org/wiki/Service-oriented_architecture) (SOA), clients and servers interact through a fixed [interface](https://en.wikipedia.org/wiki/Interface_(computing)) shared through documentation or an [interface description language](https://en.wikipedia.org/wiki/Interface_description_language) (IDL).

The HATEOAS constraint decouples client and server in a way that allows the server functionality to evolve independently.

***Microservices with Spring?***

*The term "Microservice Architecture" means a particular way of designing software applications as suites of independently deployable services.*

<https://spring.io/blog/2015/07/14/microservices-with-spring>

***Creating REST-ful, Hypermedia-based Micro-services with Spring Boot?***

<https://www.youtube.com/watch?v=zbeMDM-zDNI>