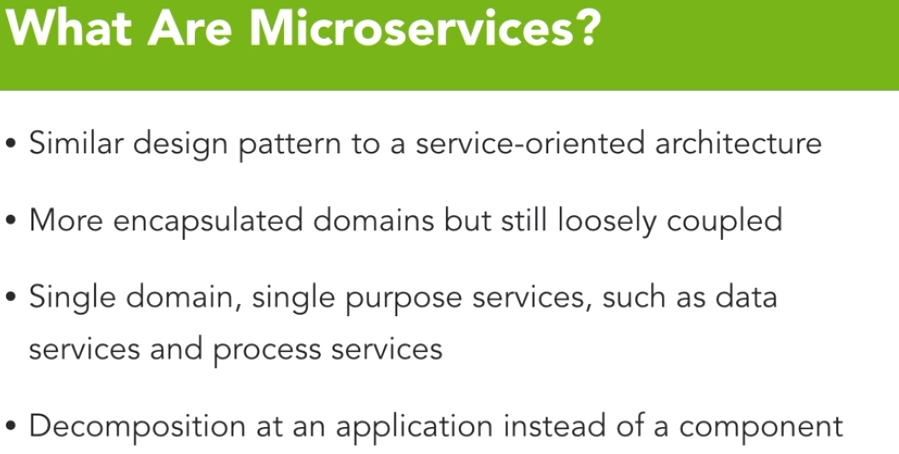
The Spring Cloud set of projects allows a developer to write Cloud-native or Twelve-Factor applications out of the box. The Spring team in Netflix collaborated to build a set of projects that allow for externalizing configuration, service discovery, software load balancing, a self-contained circuit breaker technology, as well as the service consumption.

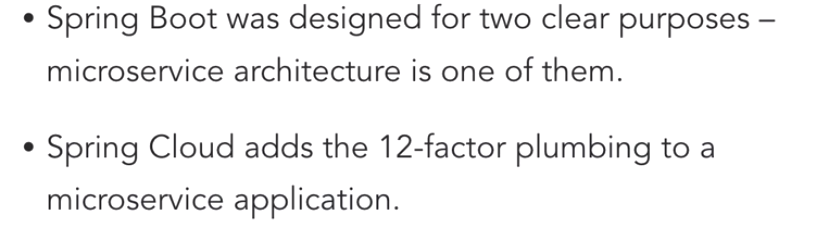
Microservices are based on a very similar design pattern called services-oriented architecture. And in this pattern, essentially all your data access and all your business processes are accessed via services.

Traditionally a services-oriented architecture was based on SOAP.

Whereas microservices are traditionally based on REST.

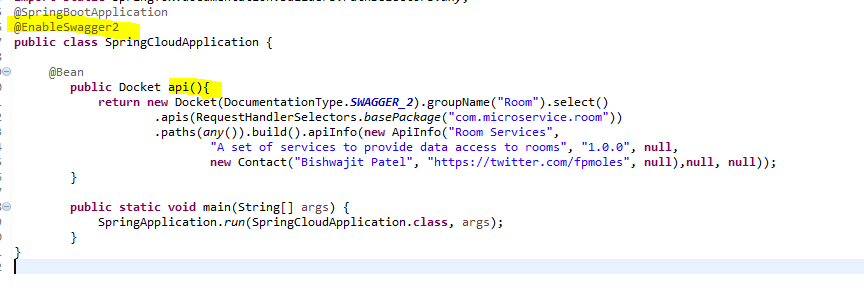
In a microservices architecture, you're dealing with much more encapsulated domains. But there is still a key that everything is very loosely coupled.

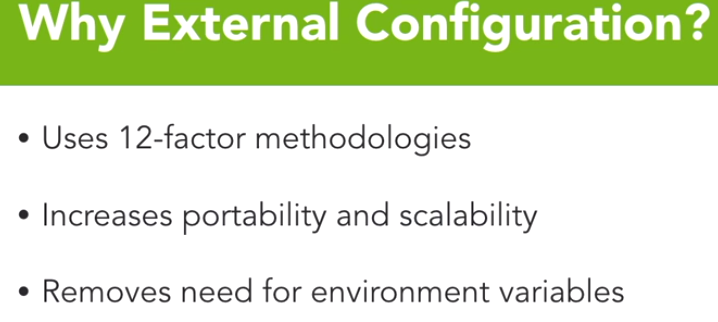




The RepositoryRestController annotation takes a traditional Spring Data repository and adds a proxy around that. And that proxy exposes all of those methods from the Spring Data repository as RESTful web services. In addition to exposing that Spring Data repository, it adds HATEOAS to the result. And specifically, the HAL Implementation. With RepositoryRestController, we're exposing an entire data domain with very little code.

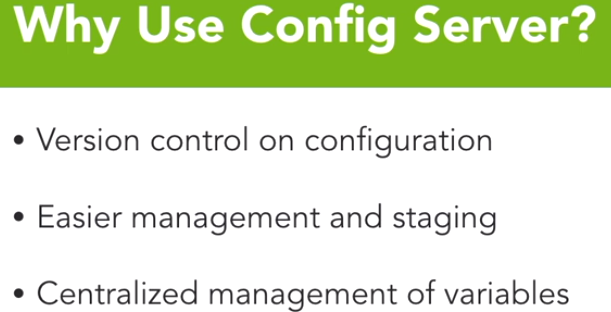
**Spring Documentation using Swagger**:





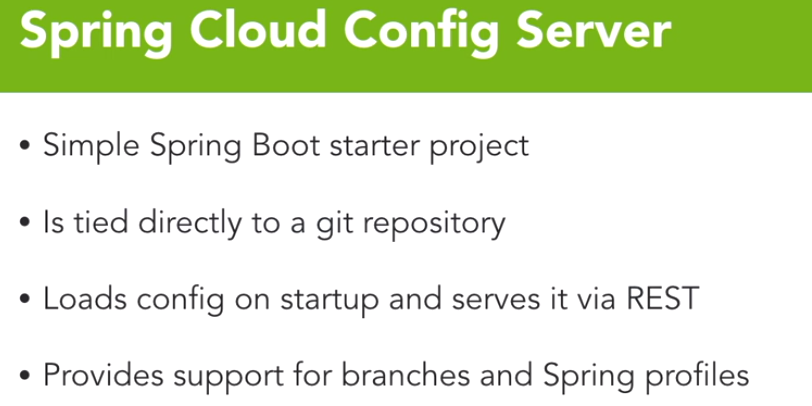
The first question that may come to your mind is, why would I ever externalize our configuration? The first and probably most prevalent reason, is because the 12-factor methodologies specifically state that configuration should be externalized. The reason that they do that, is it increases not only the portability of your application, because if it can be fed all its config at startup, it's not tied to a specific data center or instance.

But it also makes your application more scalable because I can drop into an Amazon server or an Azure server or a local server. And as long as I'm providing that config, it doesn't really matter where that server or VM, or docker container is located. It also removes the need for environment variables when dealing with external config. Now a purist of a 12-factor methodology would say that environment variables are the specific way that you handle external configuration.

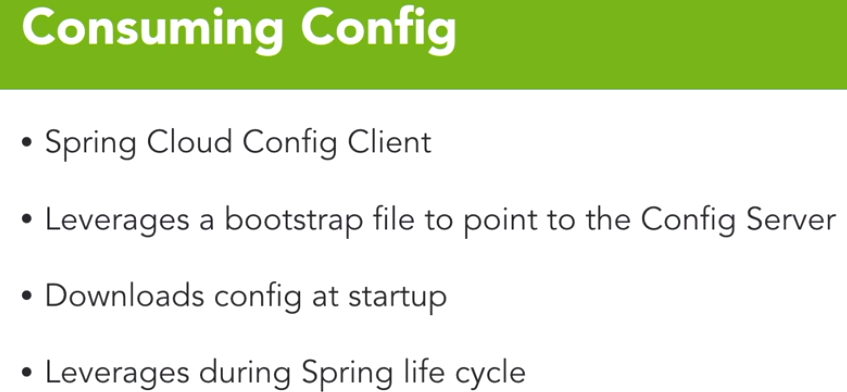


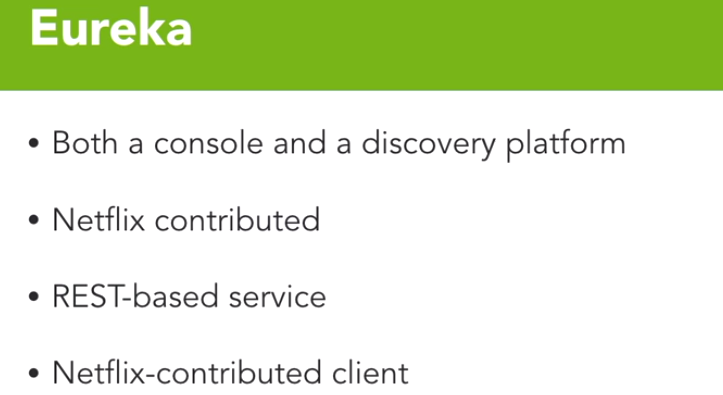
Spring's implementation specifically leverages what they call config server. Now some of the benefits of using the config server model is, that your configuration actually is under version control. In fact, it has to be backed by a Git initialized repository in order for it to even run. And once we have GIT involved, we now have version control of every change that needs to be made.

Another benefit of that version control system, is it allows us a very easy path to manage and stage this configuration. Additionally, config server gives us a centralized management point of all of our configuration variables. As opposed to each container specifically having to have its own environment variables, we have one centralized place that can distribute it to all of the applications running in that data center.

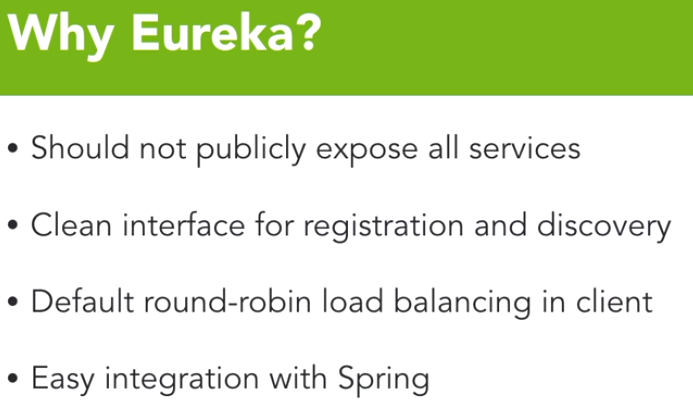


So now I can have a different profile for each data center, and a different instance of config server for each branch or all of them in one config server separated by branches, I now have the ability to deploy through a centralized Git repository to a very large number of config server instances per data center. And that really allows you a lot of flexibility while still having management over your config itself.





Eureka includes both a console and a discovery platform for registering and identifying all of your services within a distributed system. Eureka was created and open sourced by Netflix. It is a rest-based service. And in addition to the console and the platform, Netflix also contributed a Client to consume the discovery platform itself





What feign's going to allow us to do is to simplify that service call across the boundary, it's also going to allow us to integrate later on with histrix.

feign will use ribbon to make its service calls and ribbon uses eureka to do its load balancing.

