Spring Cloud Config

Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system. With the Config Server you have a central place to manage external properties for applications across all environments. The concepts on both client and server map identically to the Spring `Environment` and `PropertySource` abstractions, so they fit very well with Spring applications, but can be used with any application running in any language. As an application moves through the deployment pipeline from dev to test and into production you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate. The default implementation of the server storage backend uses git so it easily supports labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content. It is easy to add alternative implementations and plug them in with Spring configuration.

Features

Spring Cloud Config Server features:

- HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content)
- Encrypt and decrypt property values (symmetric or asymmetric)
- Embeddable easily in a Spring Boot application using `@EnableConfigServer`

Config Client features (for Spring applications):

- Bind to the Config Server and initialize Spring `Environment` with remote property sources
- Encrypt and decrypt property values (symmetric or asymmetric)

Getting Started

As long as Spring Boot Actuator and Spring Config Client are on the classpath any Spring Boot application will try to contact a config server on `[http://localhost:8888](http://localhost:8888)`, the default value of `spring.cloud.config.uri`. If you would like to change this default, you can set `spring.cloud.config.uri` in `bootstrap.[yml | properties]` or via system properties or environment variables.

@Configuration
@EnableAutoConfiguration
@RestController
public class Application {

```
  @Value("${config.name}")
  String name = "World";

  @RequestMapping("/")
  public String home() {
    return "Hello " + name;
  }

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

}
```

The value of `config.name` in the sample (or any other values you bind to in the normal Spring Boot way) can come from local configuration or from the remote Config Server. The Config Server will take precedence by default. To see this look at the `/env` endpoint in the application and see the `configServer` property sources.

To run your own server use the `spring-cloud-config-server` dependency and `@EnableConfigServer`. If you set `spring.config.name=configserver` the app will run on port 8888 and serve data from a sample repository. You need a `spring.cloud.config.server.git.uri` to locate the configuration data for your own needs (by default it is the location of a git repository, and can be a local `file:..` URL).