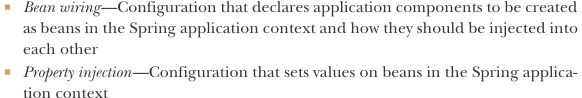
6. Working with configuration properties

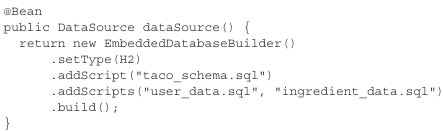
# 6.1 Fine-tunning autoconfiguration

-Kinds of configuration in Spring:



In Spring’s XML and Java configuration, these 2 types of configurations are often declared explicitly in the same place.

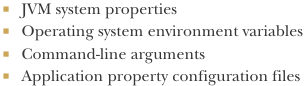
-In Java configuration, a @Bean method instantiate a bean and then set values to its properties:

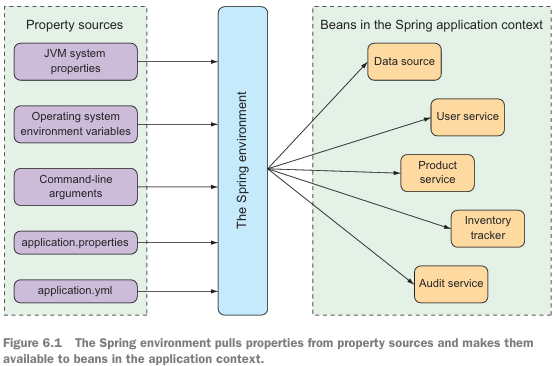


+Autoconfiguration makes this method unnecessary. If H2 dependency is available in runtime classpath, SB automatically creates in Spring application context an DataSource bean.

## 6.1.1 Understanding Spring’s environment abstraction

-Spring environment abstraction is one-stop shop for any configurable property. It abstracts the origins of properties so that beans needing those properties can consume them from Spring. Spring environment pulls from several property sources + aggregates them into a single source from which Spring beans can be injected.





-The beans that are automatically configured by SB are all configurable by properties drawn from Spring environment.

+Example: change default port

+Use command-line argument:

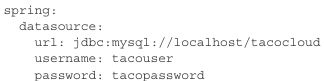


+Set it one time as an OS environment variable:

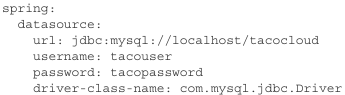


## 6.1.2 Configuring a data source

-Configure URL and credentials for database:



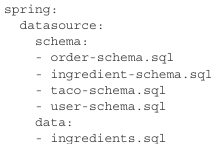
+If there’s a problem, set driver class:



+SB uses this connection data when autoconfiguring DataSource bean. This bean will be pooled using HikariCP connection pool if it’s available on classpath. If not, SB looks for and uses one of 2 other implementations:



-Specify the database initialization scripts to run:



+Instead, configure data source in Java Naming and Directory Interface JNDI (<http://mng.bz/MvEo>).



## 6.1.3 Configuring the embedded server

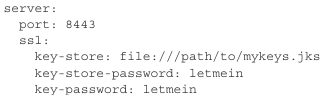
-Set the servlet container’s port by server.port. Start randomly chosen available port: 

-One of the most common things to do with underlying container is to set it up to handle HTTPS requests:  
+Create a keystore using JDK’s keytool command-line:



You’ll be asked several questions (name, orga, pass)

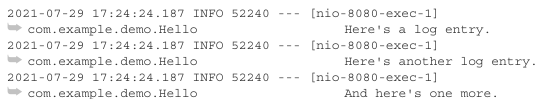
+Set properties to enable HTTPS on command-line or application.yml:



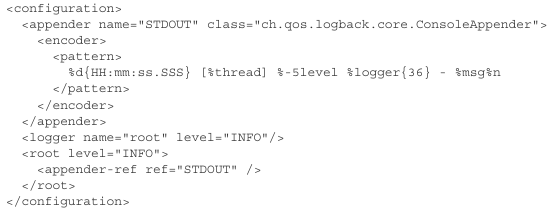
## 6.1.4 Configuring logging

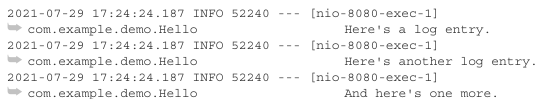
-Most apps provide some form of logging. If app doesn’t log anything, the libraries will log their activity.

-By default, SB configures logging via **Logback** (<http://logback.qos.ch/>).



-For full control over the logging configuration, you can create a logback.xml file at the root of classspath (resources):

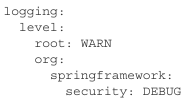




-The most common changes to logging configuration are to change the logging levels and specify a file where the logs should be written. You can make those change without creating logback.xml

-Set logging levels, create properties logging.level

+Example: Set root logging level to WARN, but log Spring Security logs at DEBUG level.



-You can collapse Spring Security package name to a single line for easier reading:



-You want to write log entries to TacoCloud.log at /var/logs:



+By default, log files rotate once they reach 10 MB

## 6.1.5 Using special property values

-Set a property named gretting.welcome to echo the value of spring.application.name:



+You can embed that placeholder amid other text:

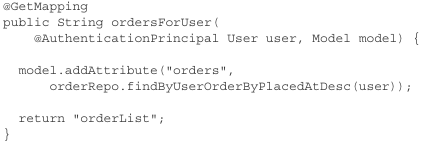


# 6.2 Creating your own configuration properties

-Configuration properties are properties of beans that have been designated to accept configuration from Spring’s environment abstraction. So how those beans are designated to consume those configuration.

-To support property injection of configuration properties, SB provides @ConfigurationProperties: It specifies that the properties of that bean can be injected from properties in Spring environment.

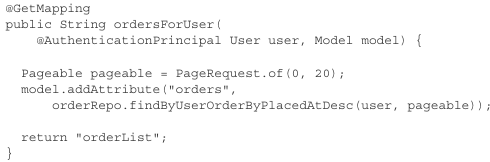
+Example: add method to OrderController to list authenticated user’s past orders:



+Add this method to OrderRepository:



-To limit the number of orders displayed to the most recent 20 orders:

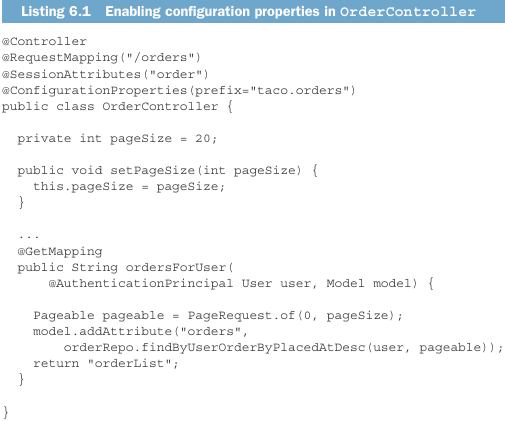


Also change OrderRepository:



+PageRequest object that implemented Pageable to request the 1st page (page zero) with a page size 20 to get up to 20 of the most recently placed orders for user.

-You can set the hardcode with a custom configuration property: Add a new property pageSize to OderController, then annotate OrderController with **@ConfigurationProperties**:



+prefix=taco.orders: when setting pageSize property, you need to use a configuration property named taco.Orders.pageSize.



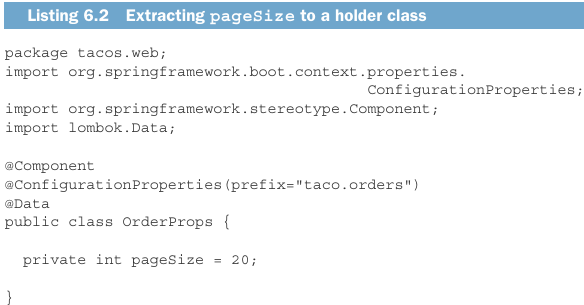
Or set it as an environment variable:



## 6.2.1 Defining configuration property holders

**-@ConfigurationProperties** are often placed on **beans** whose purpose in the app is to be **holders** of configuration data. This keeps configuration-specific details out of the controllers and other app classes. It also makes it easy to share common configuration properties among beans that may make use of that info.

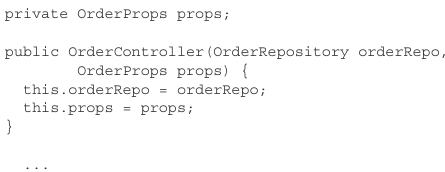
-You could extract pageSize to a separate class:

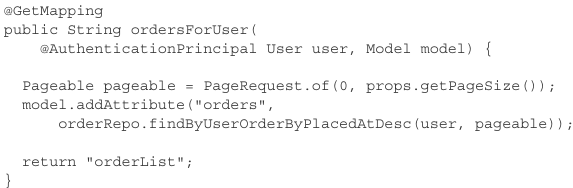


+@Component: Spring component scanning will automatically discover it and create is as a bean in Spring application context.

-Configuration property holders: beans that have their properties injected from Spring environment. They can be injected into any other bean that needs those properties.

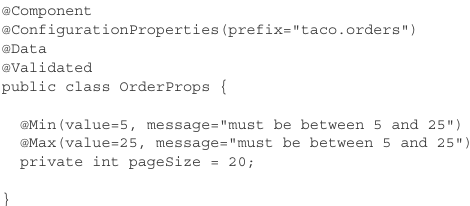
-**Inject** OrderProps bean to OrderController:





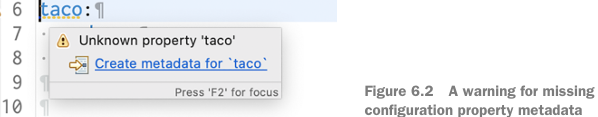
+Now OrderController is no longer responsible for handling its own configuration properties.->OrderController neater and reuse properties in OrderProps in any other bean.

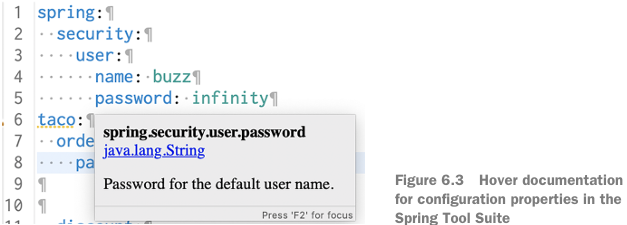
+You can collect configuration properties that pertain to orders in one place: OrderProps. If you want to apply validation, just change it in OrderProps:



## 6.2.2 Declaring configuration property metadata

-Depending on IDE, notice that taco.orders.pageSize entry in application.yml has a warning: Unknown Property ‘taco’. Because there’s missing metadata concerning the configuration property you just created.

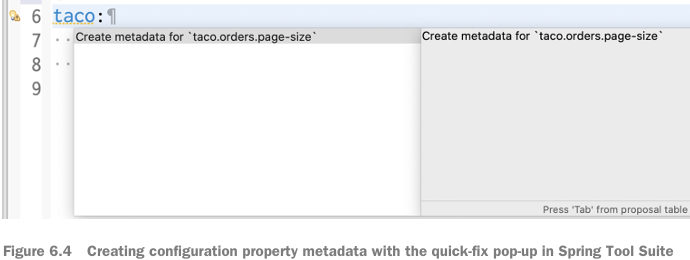


-Configuration property metadata is **optional**. But it can be useful for providing some minimal documentation around configuration properties in IDE. 

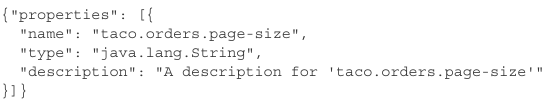
-Create metadata for your custom configuration properties: create a file under META\_INF (resources/…) named **additional-spring-configuration-metadata.json**

**-Quix-fixing missing metadata**:

+If you using Spring Tool Suite, place cursor on the line with missing metadata warning and open quick-fix pop-up (CMD-1 on Mac, Crt1 on Window).



+Select “Create metadata for…” to add some metadata for property:

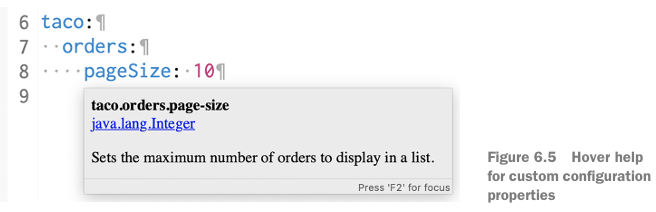


+You need to edit a little:

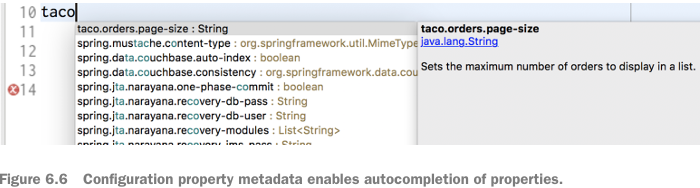




+The warning should be gone:



+You get autocompletion help from IDE:



# 6.3 Configuring with profiles

-When apps are deployed to different runtime environment, some configuration details differ (Example: database connection).

-One way to configure properties uniquely in one environment over another is to use **environment variables** to specify configuration properties instead of defining them in application.yml:



-> cumbersome + no way to track changes

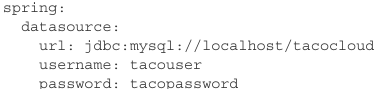
-**Spring profiles**: type of conditional configuration where different beans, configuration classes and configuration properties are applied or ignored **based on** what profiles are active at **run time**.

## 6.3.1 Defining profile-specific properties

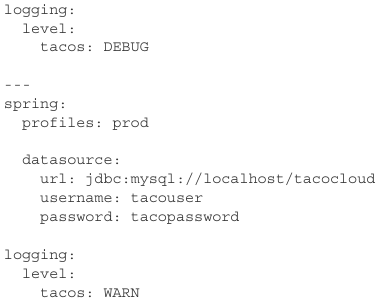
-One way to define profile-specific properties to create another yaml file containing only the properties for production.

+Name convention: application—{profile name}.yml

+Specify the configuration properties to that profile:



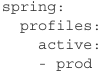


-Another way to specify profile-specific properties only with YAML: place profile-specific alongside nonprofiled properties in application.yml, separated by 3 hyphens and spring.profiles property to name the profile. 

+The 1st section are common to all profiles or are default. If the profile prod is active, the property logging will be overridden with WARN.

## 6.3.2 Activating profiles

-Active profile: Include it in the list of profile names given to **spring.profiles.active**:



+This is the worst way: If you set the active profile in application.yml, that profile becomes the default profile. –You need to set the active profile with environment variables:



+If you run app as JAR file, set the active profile with command-line argument:



+You can specify more than one active profile:

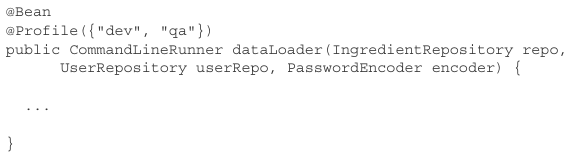
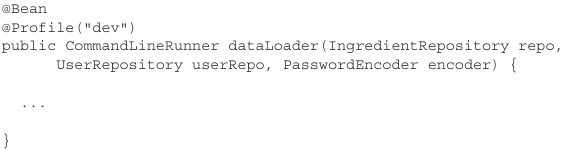


In YAML:

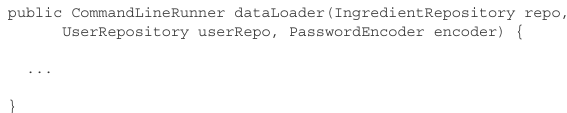


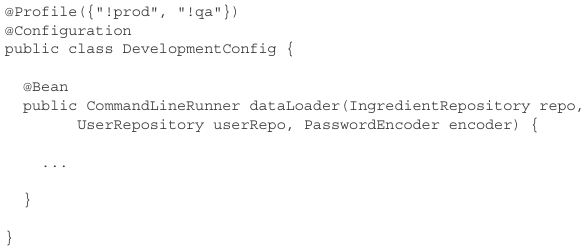
## 6.3.3 Conditionally creating beans with profiles

-Load the embedded database:









# -Summary

