* The spring team created **SpringBoot** to address the complexity of configuration.

Along with spring framework there are many other spring sister projects that help to build applications addressing modern business needs:

* Spring Data: Simplifies data access from relational and NoSQL data stores.
* Spring Batch: Provides powerful batch processing framework.
* Spring Security: Robust security framework to secure applications.
* Spring Social: Supports integration with social networking sites like Facebook, Twitter, LinkedIn, GitHub, etc.
* Spring Integration: An implementation of Enterprise Integration Patterns to facilitate integration with other enterprise applications using lightweight messaging and declarative adapters.

In the initial days, spring framework provides an XML-based approach for configuring beans. Later spring introduced XML-based DSLs, Annotations, and JavaConfig-based approaches for configuring beans.

**Example:**

**Java-Config based configuration**

@Configuration

@EnableTransactionManagement

@EnableJpaRepositories(basePackages="com.sivalabs.demo")

@PropertySource(value = { "classpath:application.properties" })

public class AppConfig

{

@Autowired

private Environment env;

@Bean

public static PropertySourcesPlaceholderConfigurer placeHolderConfigurer()

{

return new PropertySourcesPlaceholderConfigurer();

}

@Value("${init-db:false}")

private String initDatabase;

@Bean

public PlatformTransactionManager transactionManager()

{

EntityManagerFactory factory = entityManagerFactory().getObject();

return new JpaTransactionManager(factory);

}

@Bean

public LocalContainerEntityManagerFactoryBean entityManagerFactory()

{

LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();

HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();

vendorAdapter.setGenerateDdl(Boolean.TRUE);

vendorAdapter.setShowSql(Boolean.TRUE);

factory.setDataSource(dataSource());

factory.setJpaVendorAdapter(vendorAdapter);

factory.setPackagesToScan("com.sivalabs.demo");

Properties jpaProperties = new Properties();

jpaProperties.put("hibernate.hbm2ddl.auto", env.getProperty("hibernate.hbm2ddl.auto"));

factory.setJpaProperties(jpaProperties);

factory.afterPropertiesSet();

factory.setLoadTimeWeaver(new InstrumentationLoadTimeWeaver());

return factory;

}

@Bean

public HibernateExceptionTranslator hibernateExceptionTranslator()

{

return new HibernateExceptionTranslator();

}

@Bean

public DataSource dataSource()

{

BasicDataSource dataSource = new BasicDataSource();

dataSource.setDriverClassName(env.getProperty("jdbc.driverClassName"));

dataSource.setUrl(env.getProperty("jdbc.url"));

dataSource.setUsername(env.getProperty("jdbc.username"));

dataSource.setPassword(env.getProperty("jdbc.password"));

return dataSource;

}

@Bean

public DataSourceInitializer dataSourceInitializer(DataSource dataSource)

{

DataSourceInitializer dataSourceInitializer = new DataSourceInitializer();

dataSourceInitializer.setDataSource(dataSource);

ResourceDatabasePopulator databasePopulator = new ResourceDatabasePopulator();

databasePopulator.addScript(new ClassPathResource("data.sql"));

dataSourceInitializer.setDatabasePopulator(databasePopulator);

dataSourceInitializer.setEnabled(Boolean.parseBoolean(initDatabase));

return dataSourceInitializer;

}

}

In our AppConfig.java configuration class we have done the following:

* Marked it as a Spring Configuration class using **@Configuration** annotation.
* Enabled Annotation based transaction management using **@EnableTransactionManagement**
* Configured **@EnableJpaRepositories** to indicate where to look for Spring Data JPA repositories
* Configured PropertyPlaceHolder bean using **@PropertySource** annotation and **PropertySourcesPlaceholderConfigurer** bean definition, which loads properties from **application.properties** file.
* Defined beans for **DataSource**, JPA **EntityManagerFactory**, **JpaTransactionManager**.
* Configured **DataSourceInitializer** bean to initialize the database by executing **data.sql** script on application start up.

Let me explain what just happened.

**1. Easy dependency Management**

* First thing to observe is we are using some dependencies named like **spring-boot-starter-\***. Remember I said “95% of the times I use the same configuration. So when you add **springboot-starter-web** dependency by default it will pull all the commonly used libraries while developing Spring MVC applications such as **spring-webmvc, jackson-json, validation-api** and **tomcat**.
* We have added **spring-boot-starter-data-jpa** dependency. This pulls all the **spring-data-jpa** dependencies and also adds **Hibernate** libraries because the majority of the applications use Hibernate as JPA implementation.

**2. Auto Configuration**

* Not only the **spring-boot-starter-web** adds all these libraries but also configures the commonly registered beans like **DispatcherServlet, ResourceHandlers, MessageSource** etc beans with sensible defaults.
* We also added **spring-boot-starter-Thymeleaf** which not only adds the Thymeleaf library dependencies but also configures **ThymeleafViewResolver** beans as well automatically.
* We haven’t defined any of the **DataSource, EntityManagerFactory, TransactionManager**etc beans but they are automatically gets created. How? If we have any in-memory database drivers like **H2** or **HSQL** in our classpath then SpringBoot will automatically create an in-memory **DataSource** and then registers **EntityManagerFactory, TransactionManager** beans automatically with sensible defaults. But we are using MySQL, so we need to explicitly provide MySQL connection details. We have configured those MySQL connection details in **application.properties** file and SpringBoot creates a **DataSource** using these properties.

**3. Embedded Servlet Container Support**

The most important and surprising thing is we have created a simple Java class annotated with some magical annotation **@SpringApplication** having a main method and by running that main we are able to run the application and access it at **http://localhost:8080/**.

**Where is the servlet container comes from?**   
We have added **spring-boot-starter-web** which pulls the **spring-boot-starter-tomcat** automatically and when we run the main() method it started tomcat as an **embedded container** so that we don’t have to deploy our application on any externally installed tomcat server.

By the way have you observe that our packaging type in **pom.xml** is **‘jar’ not ‘war’**. Wonderful!

**Spring Data**

Spring Data’s mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.   
  
It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services. This is an umbrella project which contains many subprojects that are specific to a given database. The projects are developed by working together with many of the companies and developers that are behind these exciting technologies.

Features

* Powerful repository and custom object-mapping abstractions
* Dynamic query derivation from repository method names
* Implementation domain base classes providing basic properties
* Support for transparent auditing (created, last changed)
* Possibility to integrate custom repository code
* Easy Spring integration via JavaConfig and custom XML namespaces
* Advanced integration with Spring MVC controllers
* Experimental support for cross-store persistence

Main modules

* [Spring Data Commons](http://docs.spring.io/spring-data/commons/docs/current/reference/html/) - Core Spring concepts underpinning every Spring Data project.
* [Spring Data Gemfire](http://projects.spring.io/spring-data-gemfire) - Provides easy configuration and access to GemFire from Spring applications.
* [Spring Data JPA](http://projects.spring.io/spring-data-jpa) - Makes it easy to implement JPA-based repositories.
* [Spring Data KeyValue](https://github.com/spring-projects/spring-data-keyvalue) - Map-based repositories and SPIs to easily build a Spring Data module for key-value stores.
* [Spring Data LDAP](http://projects.spring.io/spring-data-ldap) - Provides Spring Data repository support for [Spring LDAP](https://github.com/spring-projects/spring-ldap).
* [Spring Data MongoDB](http://projects.spring.io/spring-data-mongodb) - Spring based, object-document support and repositories for MongoDB.
* [Spring Data REST](http://projects.spring.io/spring-data-rest) - Exports Spring Data repositories as hypermedia-driven RESTful resources.
* [Spring Data Redis](http://projects.spring.io/spring-data-redis) - Provides easy configuration and access to Redis from Spring applications.
* [Spring Data for Apache Cassandra](http://projects.spring.io/spring-data-cassandra) - Spring Data module for Apache Cassandra.
* [Spring Data for Apache Solr](http://projects.spring.io/spring-data-solr) - Spring Data module for Apache Solr.

Community modules

* [Spring Data Aerospike](https://github.com/aerospike/spring-data-aerospike) - Spring Data module for Aerospike.
* [Spring Data Couchbase](http://projects.spring.io/spring-data-couchbase) - Spring Data module for Couchbase.
* [Spring Data DynamoDB](https://github.com/michaellavelle/spring-data-dynamodb) - Spring Data module for DynamoDB.
* [Spring Data Elasticsearch](http://projects.spring.io/spring-data-elasticsearch) - Spring Data module for Elasticsearch.
* [Spring Data Hazelcast](https://github.com/hazelcast/spring-data-hazelcast) - Provides Spring Data repository support for Hazelcast.
* [Spring Data Neo4j](http://projects.spring.io/spring-data-neo4j) - Spring based, object-graph support and repositories for Neo4j.

Related modules

* [Spring Data JDBC Extensions](http://projects.spring.io/spring-data-jdbc-ext) - Provides extensions to the JDBC support provided in the Spring Framework.
* [Spring for Apache Hadoop](http://projects.spring.io/spring-hadoop) - Simplifies Apache Hadoop by providing a unified configuration model and easy to use APIs for using HDFS, MapReduce, Pig, and Hive.
* [Spring Content](https://paulcwarren.github.io/spring-content/) - Associate content with your Spring Data Entities and store it in a number of different stores including the File-system, S3, Database or Mongo’s GridFS.