Week 7

Basics of Apache Maven

- Apache Maven is a software project management and comprehension tool.
- Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.
- Using maven we can build and manage any Java based project.

Maven is a popular open-source build tool developed by the Apache Group to build, publish, and deploy several projects at once for better project management. The tool provides allows developers to build and document the lifecycle framework

What maven does?

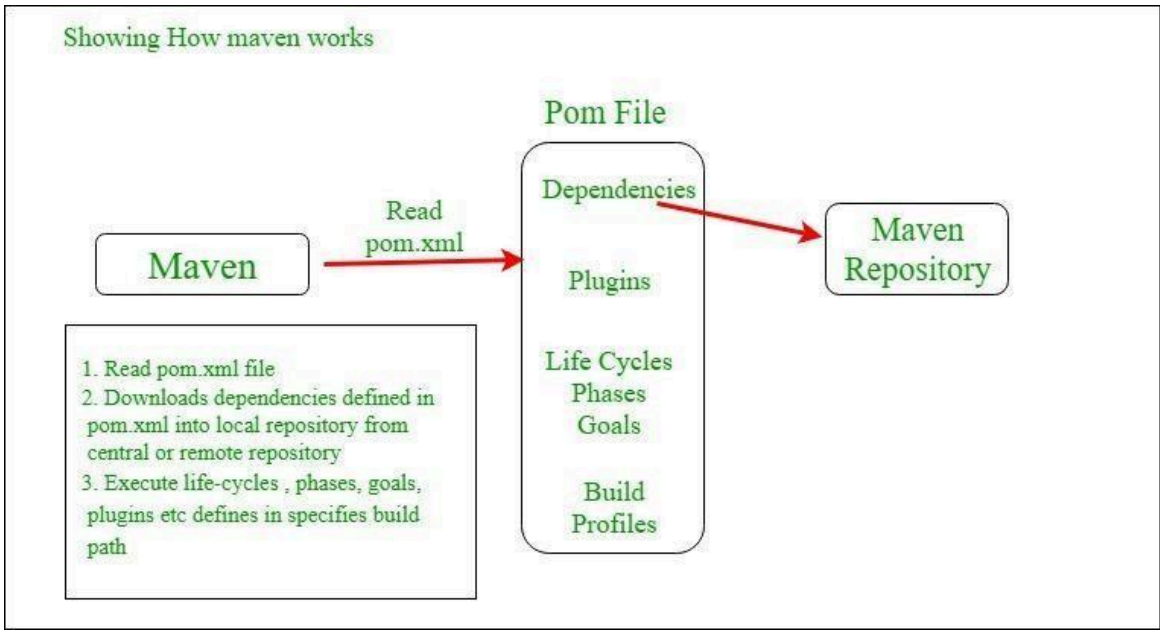
Maven does a lot of helpful task like

1. We can easily build a project using maven.
2. We can add jars and other dependencies of the project easily using the help of maven.
3. Maven provides project information (log document, dependency list, unit test reports etc.)
4. Maven is very helpful for a project while updating central repository of JARs and other dependencies.
5. With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.
6. Using maven we can easily integrate our project with source control system (such as Subversion or Git).

Core Concepts of Maven:

1. **POM Files:** Project Object Model(POM) Files are XML file that contains information related to the project and configuration information such as dependencies, source directory, plugin, goals etc.
2. **Dependencies and Repositories:** Dependencies are external Java libraries required for Project and repositories are directories of packaged JAR files.
3. **Build Life Cycles, Phases and Goals:** A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals.
4. **Build Profiles:** Build profiles a set of configuration values which allows you to build your project using different configurations.
5. **Build Plugins:** Build plugins are used to perform specific goal. you can add a plugin to the POM file.

How maven works?



Project management tool

- Apache Maven is a build tool.
- This is a software project management tool that gives a new concept of the project object model (POM).
- Maven allows the developer to automate the handling of the creation of the original folder format, performing the assortment and testing and the packaging and deployment of the final output.
- It cuts down the considerable number of steps in the base process, and it makes it just one-step process to do a build.

Understanding pom.xml

- POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.
- The POM contains information about the project and various configuration detail used by Maven to build the project(s).

- POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.
- Some of the configuration that can be specified in the POM are following
- project dependencies
- plugins
- goals
- build profiles
- project version
- developers
- mailing list

Before creating a POM, we should first decide the project group (groupId), its name (artifactId) and its version as these attributes help in uniquely identifying the project in repository.

POM Example

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.project-group</groupId>
<artifactId>project</artifactId>
<version>1.0</version>
</project>
```

It should be noted that there should be a single POM file for each project.

- All POM files require the project element and three mandatory fields: groupId, artifactId, version.

- Projects notation in repository is groupId:artifactId:version.
- Minimal requirements for a POM −

Sr.No.	Node & Description
1	Project root This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification.
2	Model version Model version should be 4.0.0.
3	groupId This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.
4	artifactId This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository.
5	version This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example − com.company.bank:consumer-banking:1.0 com.company.bank:consumer-banking:1.1.

What is Spring?

- The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications.
- One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using plain old Java objects (POJOs).

- Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.

Why Spring?

1. Easy, Simple, and Lightweight

- Spring is easy to learn and implement which is composed of modules where we can write spring applications normally with interfaces and abstract classes just like java applications.
- Spring helps in coupling and wiring the components which make working with Spring easier and simpler.
- It is lightweight as we can inject dependencies as per our need

2. Builds Secure Web Applications

- Spring provides security if Spring Security is on the classpath and we can customize the security settings further for basic authentication and prevent vulnerabilities in our project.

3. MVC Pattern

- Model View Controller which helps in separating implementation and business logic
- The view is where requests are received first which are taken by the controller to the corresponding model for results which are then taken to view to show at the front side of the application.

4. Easy Communication with Databases

- With the support of DAO(Data Access Object) in Spring, data access related technologies such as Hibernate, JDBC, and JPA make it easier to communicate with databases.

5. Modular Design

- It is divided into components such as core container, data access / integration, web, and test which work together but independently.
- Implementation can also be divided in Spring according to the MVC pattern

6. Can be Integrated with Other Frameworks

- It can be used with other frameworks such as Struts, and Hibernate.

7. Dependency Injection

- Dependency Injection helps in decreasing coupling and dependency between classes in Spring projects so that the program becomes maintainable and reusable.
- Modules of one project can be used effectively in another project like a login page, and registration page which not only saves them time but also promotes code reusability

8. Follows Aspect-Oriented Programming

- Aspect-Oriented Programming allows us to think differently about the structure of the program by enabling the modularization of concerns.
- It helps in breaking down the logic into parts known as concerns and the concerns help in dividing the business logic of an application and in increasing the modularity.

9. Testing becomes easy

- classes can be tested independently without having to depend on one another

10. Handle external resources easily

- Resource and ResourceLoader are the interfaces present in Spring to handle external resources

How Spring works

A web application (layered architecture) commonly includes three layers:

1. Presentation/view layer (UI) - This is the outermost layer which handles the presentation of content and interaction with the user.

2. Business logic layer - The central layer that deals with the logic of a program.
3. Data access layer - The deep layer that deals with data retrieval from sources.

Important terms

Autowiring - The process by which Spring identifies dependencies and matches and populates them.

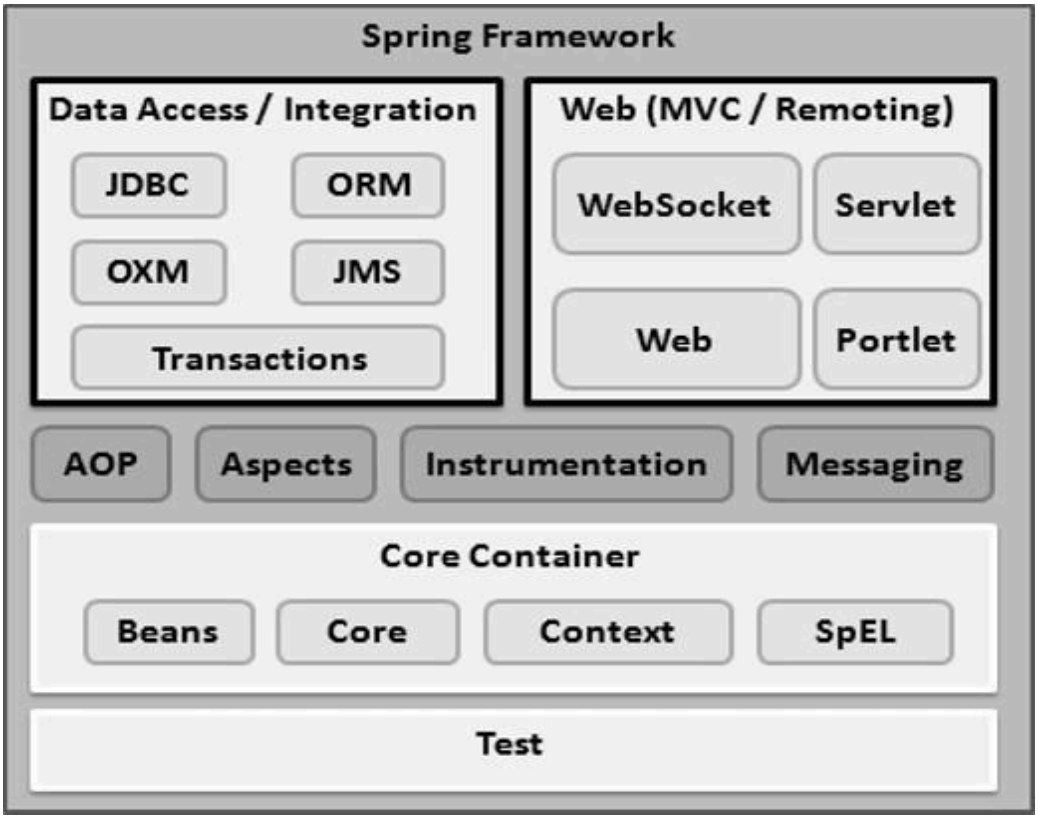
Bean - A Spring bean is an object that is instantiated, created, and managed by the IoC container. Beans are the backbone of an application.

Dependency injection - A programming design pattern that makes code loosely coupled, meaning that any change in the application of one, will not affect the other.

Inversion of control (IoC) - Taking control away from the class and giving it to the Spring Framework.

Inversion of control container - This is the core of the Spring Framework where objects are created, wired together, configured, and managed throughout their life cycle.

Spring Framework Architecture



Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules

- The **Core module** provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Bean module** provides BeanFactory, which is a sophisticated implementation of the factory pattern.
- The **Context module** builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured.
- The **SpEL module** provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules

- The **JDBC module** provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
- The **ORM module** provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
- The **OXM module** provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service **JMS module** contains features for producing and consuming messages.
- The **Transaction module** supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules

- The **Web module** provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.

- The **Web-MVC module** contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The **Web-Socket module** provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The **Web-Portlet module** provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Miscellaneous

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules

- The **AOP module** provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
- The **Aspects module** provides integration with AspectJ, which is again a powerful and mature AOP framework.
- The **Instrumentation module** provides class instrumentation support and class loader implementations to be used in certain application servers.
- The **Messaging module** provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
- The **Test module** supports the testing of Spring components with JUnit or TestNG frameworks.

Key components of Spring Framework

- Spring Core
- Spring AOP
- Spring Web MVC
- Spring DAO
- Spring ORM
- Spring context
- Spring Web flow

SpringBoot

- Spring Boot is an open source Java-based framework used to create a micro Service.
- Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**. You can get started with minimum configurations without the need for an entire Spring configuration setup

Advantages

Spring Boot offers the following advantages to its developers −

- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time

Why SpringBoot?

- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container

Compare Spring and SpringBoot

Spring	Spring Boot
Spring Framework is a widely used Java EE framework for building applications.	**Spring Boot Framework** is widely used to develop **REST APIs**.
It aims to simplify Java EE development that makes developers more productive.	It aims to shorten the code length and provide the easiest way to develop **Web Applications**.

The primary feature of the Spring Framework is **dependency injection**.	The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement.
It helps to make things simpler by allowing us to develop **loosely coupled** applications.	It helps to create a **stand-alone** application with less configuration.
The developer writes a lot of code (**boilerplate code**) to do the minimal task.	It **reduces** boilerplate code.
To test the Spring project, we need to set up the sever explicitly.	Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc.
It does not provide support for an in-memory database.	It offers several plugins for working with an embedded and **in-memory** database such as **H2**.
Developers manually define dependencies for the Spring project in **pom.xml**.	Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement.

understanding the spring initializer interface

Spring Initializer is a web-based tool. With the help of Spring Initializer, we can easily generate the structure of the Spring Boot Project. It offers extensible API for creating JVM-based projects.

Annotations

Annotations are used to provide supplemental information about a program.

- Annotations start with '**@**'.
- Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.

- Annotations basically are used to provide additional information, so could be an alternative to XML and Java marker interfaces.

Spring Annotations List

@Service

Annotate all your service classes with @Service. All your business logic should be in Service classes.

```
@Service
public class CompanyServiceImpl implements CompanyService {
...
}
```

@Repository

Annotate all your DAO classes with @Repository. All your database access logic should be in DAO classes.

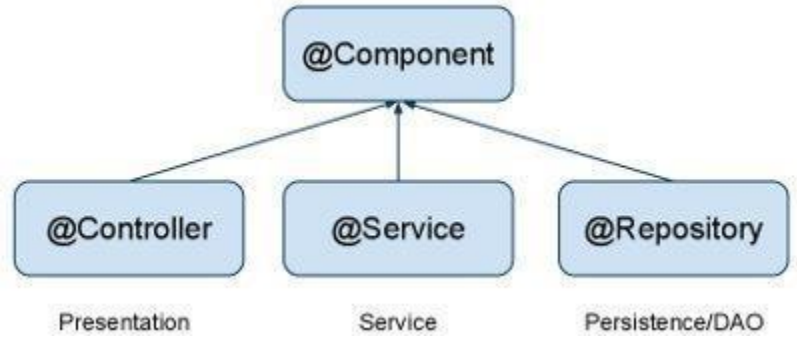
```
@Repository
public class CompanyDAOImpl implements CompanyDAO {
...
}
```

@Component

Annotate your other components (for example REST resource classes) with @Component

```
@Component
public class ContactResource {
...
}
```

@Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.



@Autowired

Let Spring auto-wire other beans into your classes using @Autowired annotation.

```
@Service
public class CompanyServiceImpl implements CompanyService {

  @Autowired
  private CompanyDAO companyDAO;

  ...
}
```

@Transactional

Configure your transactions with @Transactional spring annotation.

```
@Service
public class CompanyServiceImpl implements CompanyService {

  @Autowired
  private CompanyDAO companyDAO;

  @Transactional
  public Company findByName(String name) {

    Company company = companyDAO.findByName(name);
    return company;
  }
  ...
}
```

@Scope

To change default behavior, use @Scope spring annotation.

```
@Component
@Scope("request")
public class ContactResource {
...
}
```

What is inversion of control?

- Spring IoC (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle.
- Inversion of Control (IoC) is a design principle that allows classes to be loosely coupled and, therefore, easier to test and maintain.
- IoC refers to transferring the control of objects and their dependencies from the main program to a container or framework.

What is dependency injection?

- Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control).
- The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods.

Types of Spring Dependency Injection.

1. **Constructor Injection:** In the constructor injection, the injector supplies the service (dependency) through the client class constructor.
2. **Property Injection:** In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.
3. **Method Injection:** In this type of injection, the client class implements an interface which declares the method(s) to supply the dependency and the injector uses this interface to supply the dependency to the client class.

Practice:

Spring IoC container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the

objects There are two types of IoC containers. They are:

1. BeanFactory
2. ApplicationContext

Using BeanFactory

The XmlBeanFactory is the implementation class for the BeanFactory interface.

```
Resource resource=new ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
```

Using ApplicationContext

The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface.

```
ApplicationContext context =
  new ClassPathXmlApplicationContext("applicationContext.xml");
```

ComponentScanning

- we use the @ComponentScan annotation along with the @Configuration annotation to specify the packages that we want to be scanned.
- @ComponentScan without arguments tells Spring to scan the current package and all of its sub-packages.

```
@Confi
guration
@Comp
onentSc
an
public
class
SpringC                                                    new
ompone
ntScanA
pp {
private static
ApplicationContext
applicationContext;
@Bean
public
Example
Bean
example
Bean() {
return
new
Example
Bean();
}
 p ıblic static void main(String[] args) {
    applicationContext     =
AnnotationConfigApplicationContext(S
pringComponentScanApp.class);

for     (String     beanName     :     applicationContext.getBeanDefinitionNames())     {
```

```
        System.out.println(beanName);
        }
 }
}
```

```
package com.baeldung.componentscan.springapp.animals;
// ...
@Componen
t
public class Cat {}
```

```
package com.baeldung.componentscan.springapp.animals;
 // ...
@Componen
t
public class Dog {}
```

DI in spring Boot

Dependency Injection is the ability of an object to supply dependencies of another object. It mainly has three classes involved:

- Client Class: This is the dependent class and is dependent on the Service class.
- Service Class: This class provides a service to the client class.
- Injector Class: This class is responsible for injecting the service class object into the client class

There are mainly three types of Dependency Injection:

- **Constructor Injection:** In this type of injection, the injector supplies dependency through the client class constructor.
- **Setter Injection / Property Injection:** In this type of injection, the injector method injects the dependency to the setter method exposed by the client.
- **Interface Injection:** In this type of injection, the injector uses Interface to provide dependency to the client class. The clients must implement an interface that will expose a setter method which accepts the dependency.

Autowiring

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Qualifier

@Qualifier is an annotation used to specify which Spring managed bean should be injected via @Autowired.

Bean

- Any object in the Spring framework that we initialize through Spring container is called Spring Bean.
- Any normal Java POJO class can be a Spring Bean if it's configured to be initialized via container by providing configuration metadata information.
- In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.

- A bean is an object that is instantiated, assembled, and managed by a Spring IoC container.

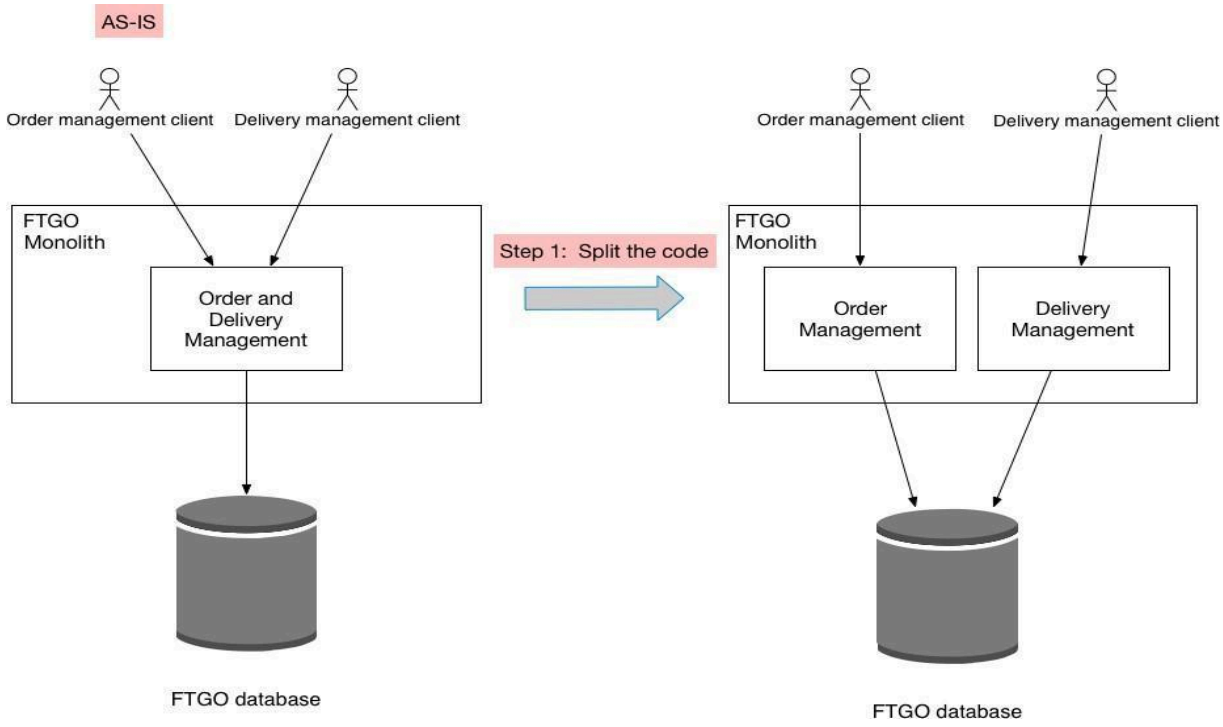
Spring Bean Scopes

Sr.No.	Scope & Description
1	singleton This scopes the bean definition to a single instance per Spring IoC container (default).
2	prototype This scopes a single bean definition to have any number of object instances.
3	request This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
4	session This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
5	global-session This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Converting monolithic application to microservices architecture

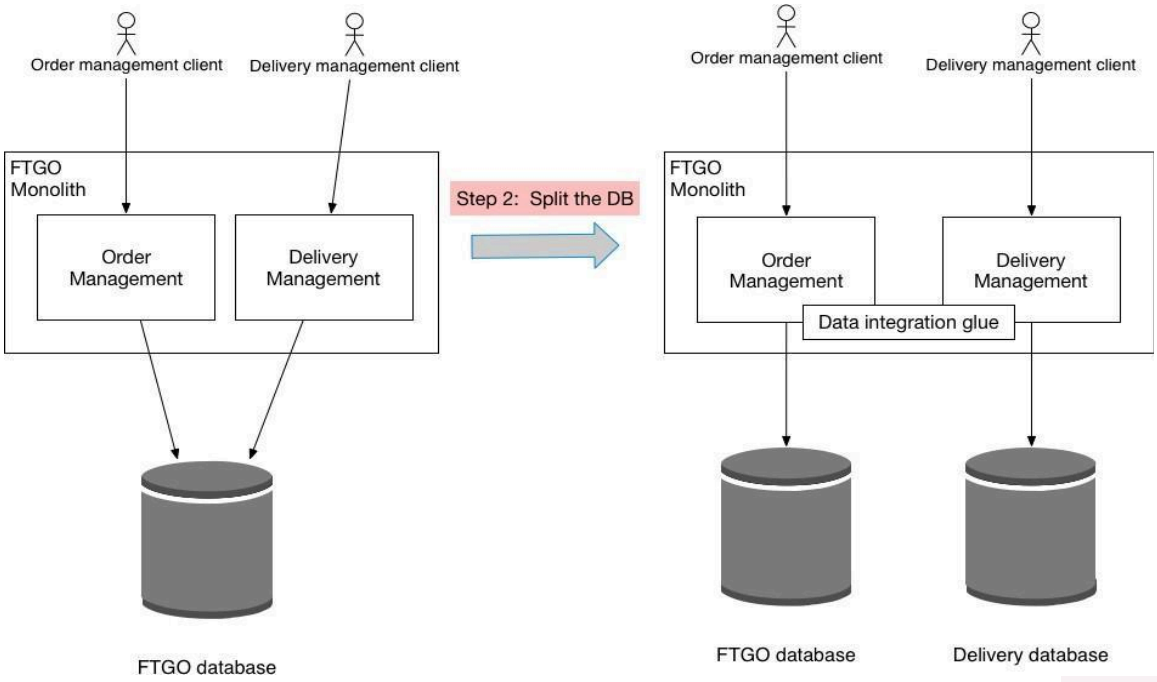
Step 1: Split the code.
Step 2: Split the database.
Step 3: Define a standalone Delivery Service.
Step 4: Use the standalone Delivery Service.
Step 5: Remove the delivery management functionality from the FTGO monolith.

Step 1: Split the code



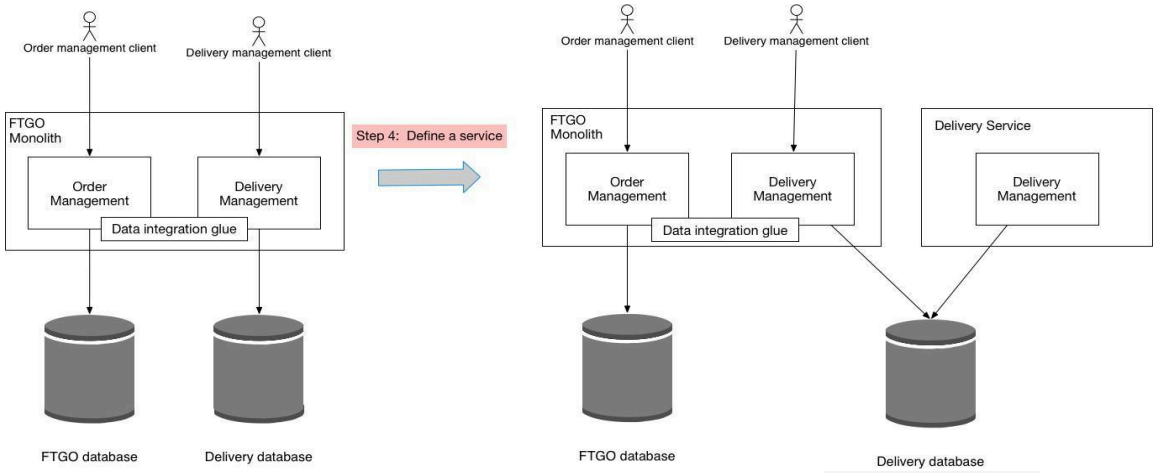
The first step is to split the code and convert delivery management into a separate, loosely coupled module within the monolith.

Step 2: Split the database



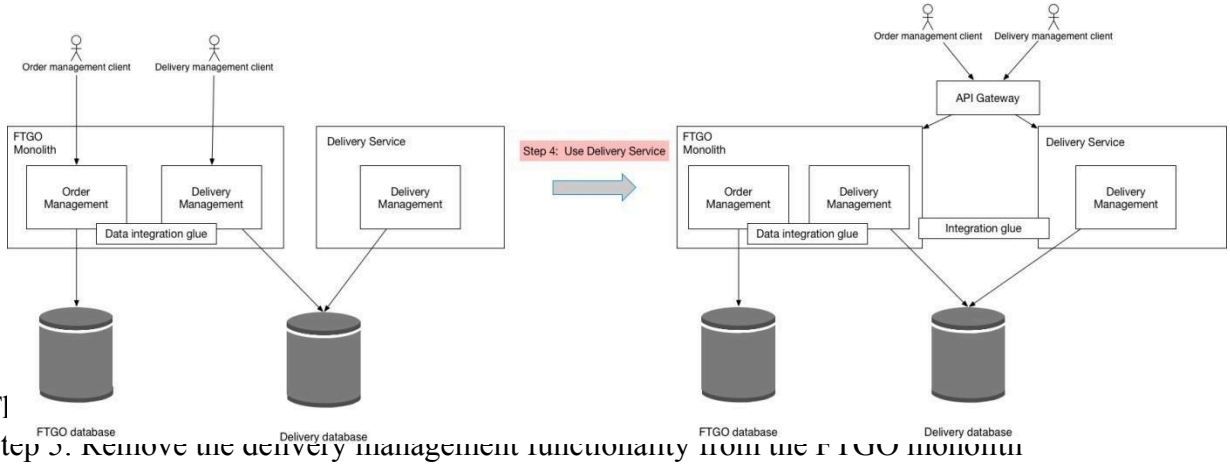
The second step is split the database and define a separate database schema for the ftgo-delivery-service module.

Step 3: Define a standalone Delivery Service



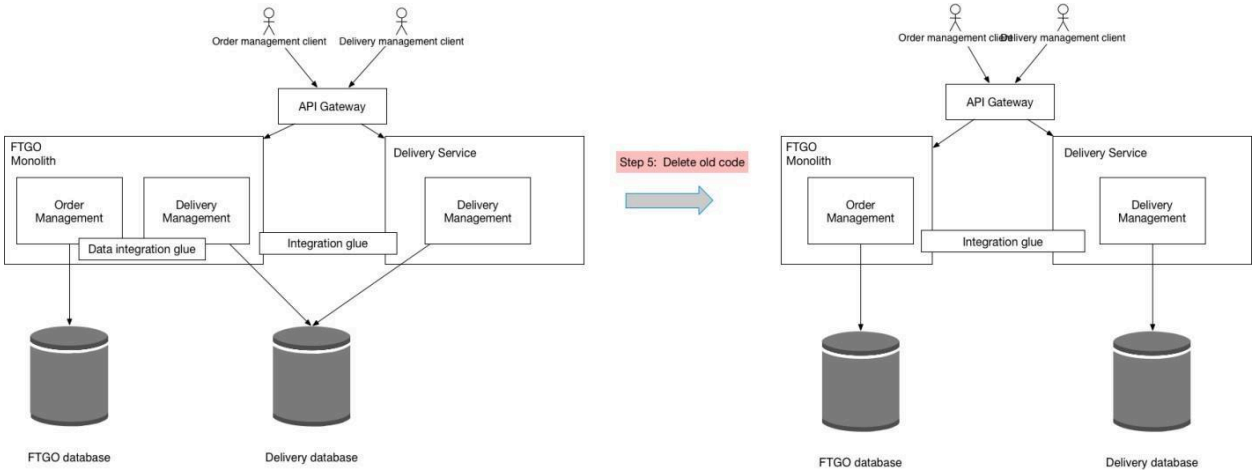
The third step of the refactoring process is to define a standalone Delivery Service and deploy it. The service does not, however, handle production traffic. Instead, it can, for example, be tested in production.

Step 4: Use the standalone Delivery Service



Th

Step 5: Remove the delivery management functionality from the FTGO monolith



The fifth step of the refactoring process is to remove the now obsolete delivery management logic from the monolith.