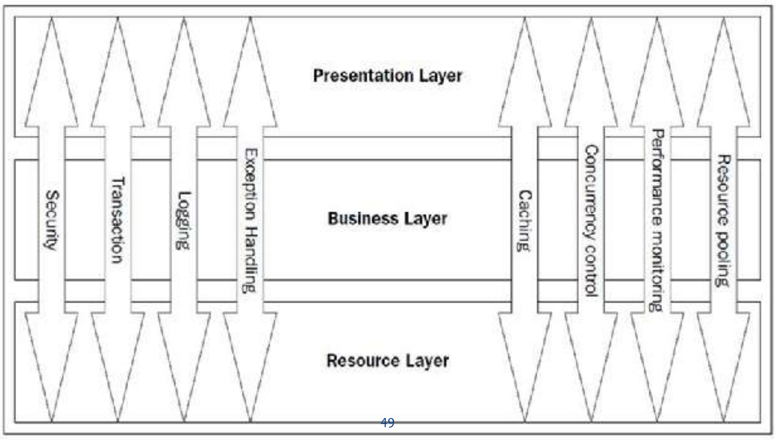
Spring AOP

Object Oriented Programming-Limits

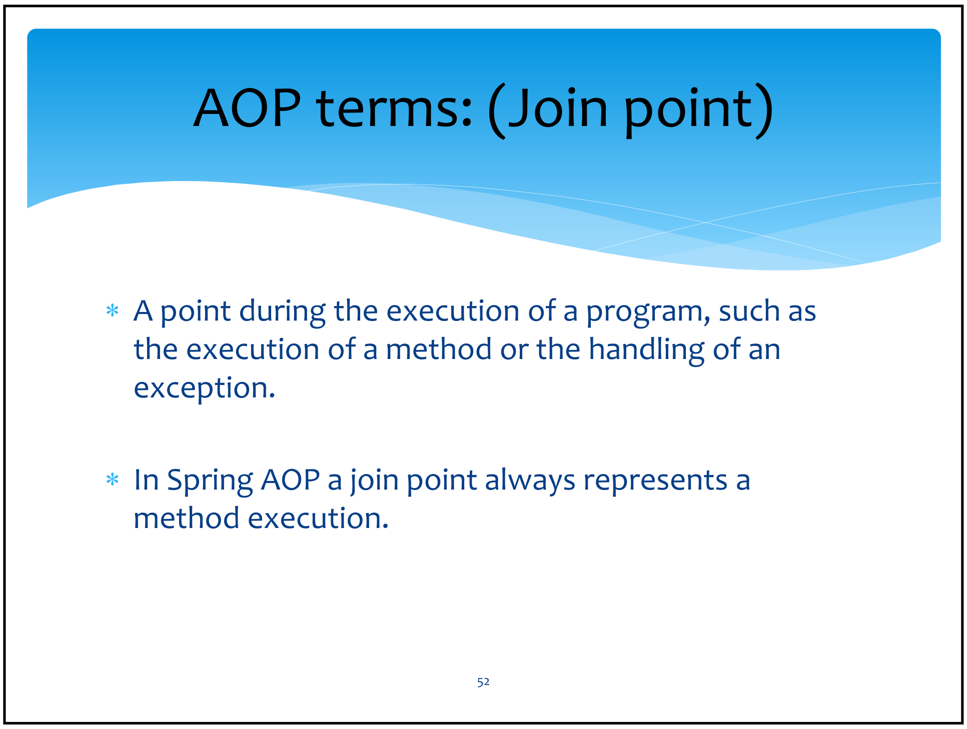


Aspect-oriented programming

- Aspect-oriented programming supports object-oriented programming by de-coupling modules that implement cross-cutting concerns.
- Its purpose is the separation of concerns.
- In object-oriented programming the basic unit is the Class, whereas in aspect-oriented programming it’s the Aspect.

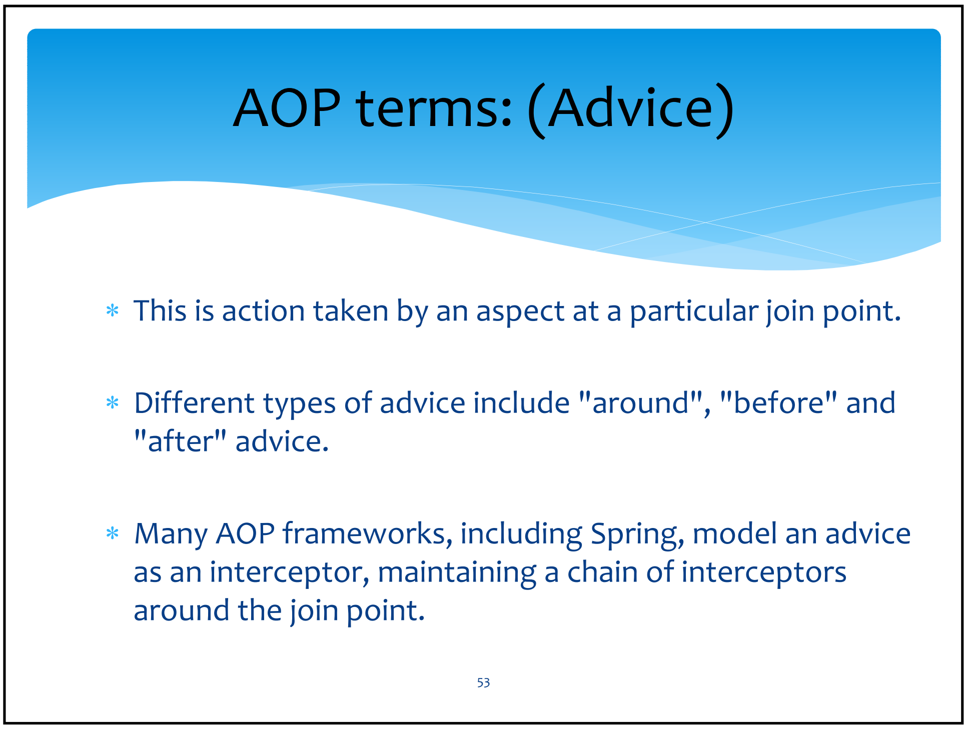
AOP terms: (Aspect)

- A modularization of a concern that cuts across multiple classes.
- Transaction management is a good example of a cross-cutting concern in enterprise Java applications.



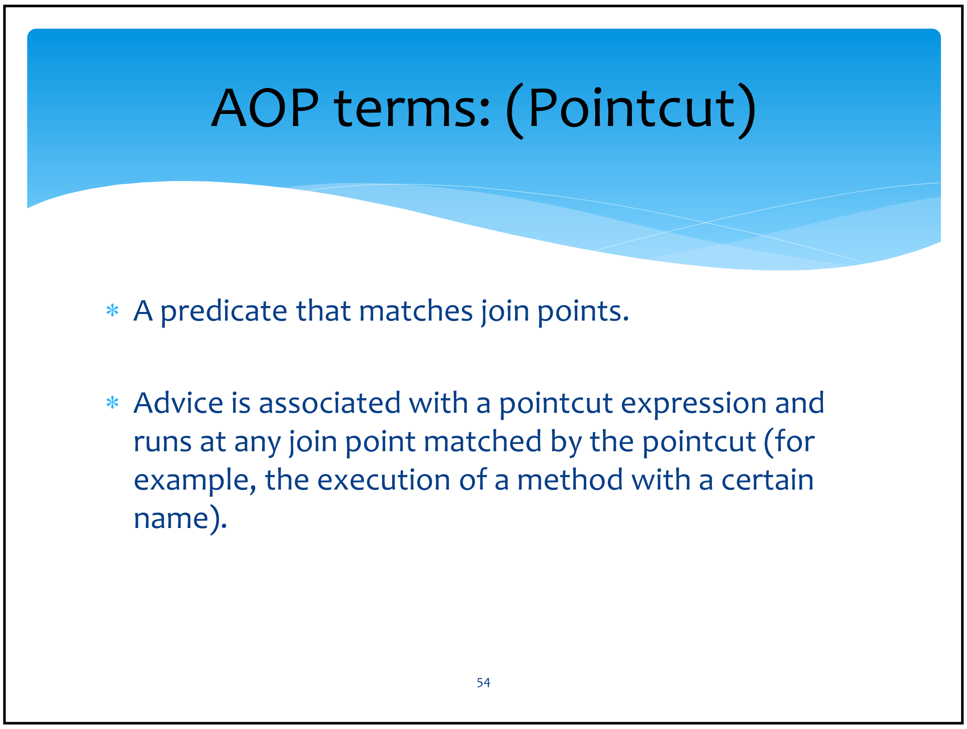
AOP terms: (Join point)

- A point during the execution of a program, such as the execution of a method or the handling of an exception.
- In Spring AOP a join point always represents a method execution.



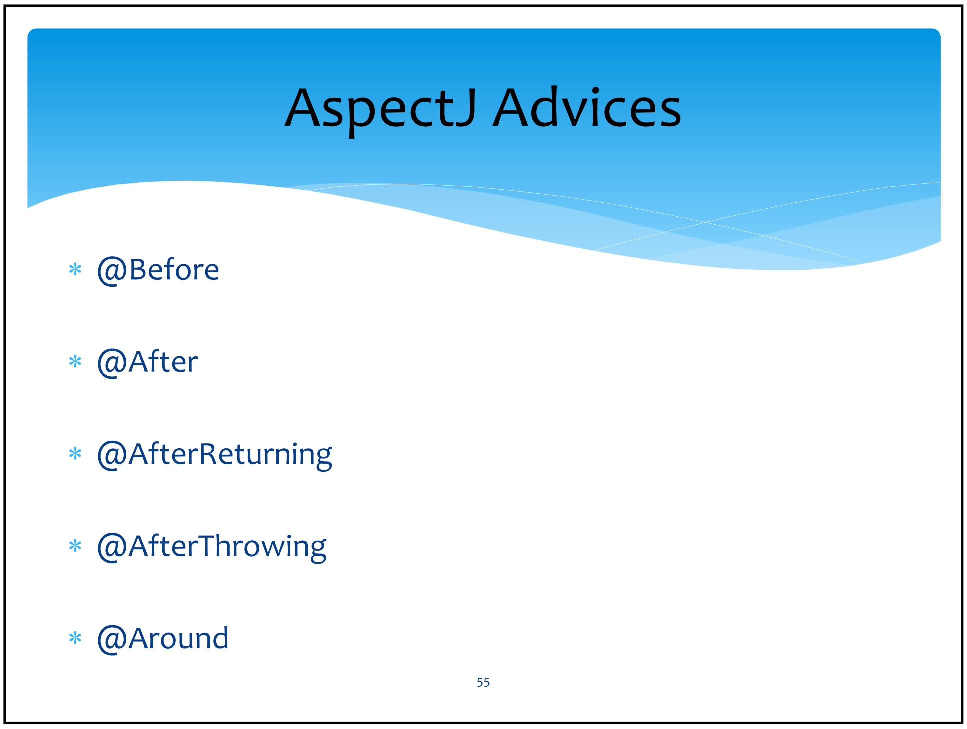
AOP terms: (Advice)

- This is action taken by an aspect at a particular join point.
- Different types of advice include "around", "before" and "after" advice.
- Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors around the join point.



AOP terms: (Pointcut)

- A predicate that matches join points.
- Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name).



AspectJ Advices

- @Before
- @After
- @AfterReturning
- @AfterThrowing
- @Around

MICROSERVICES

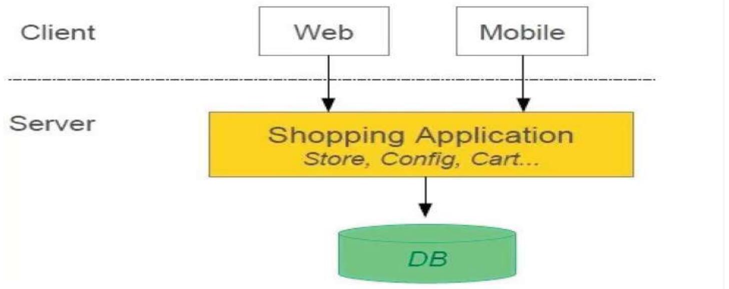
What is Microservices?

- Microservices is not a new term. It coined in 2005 by Dr Peter Rodgers
- It was then called micro web services based on SOAP. It became more popular since 2010.
- Micoservices allows us to break our large system into number of independent collaborating processes.

What is Microservices?

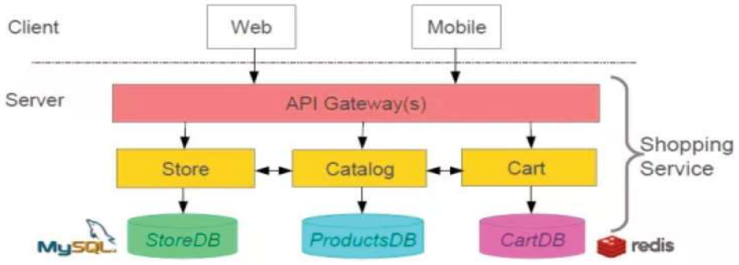
- Microservices architecture allows to avoid monolith application for large system.
- It provides loose coupling between collaborating processes running independently in different environments with tight cohesion.
- It is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack.

Example : Monolith Architecture



- This is Monolith architecture i.e. all collaborating components combine all in one application.

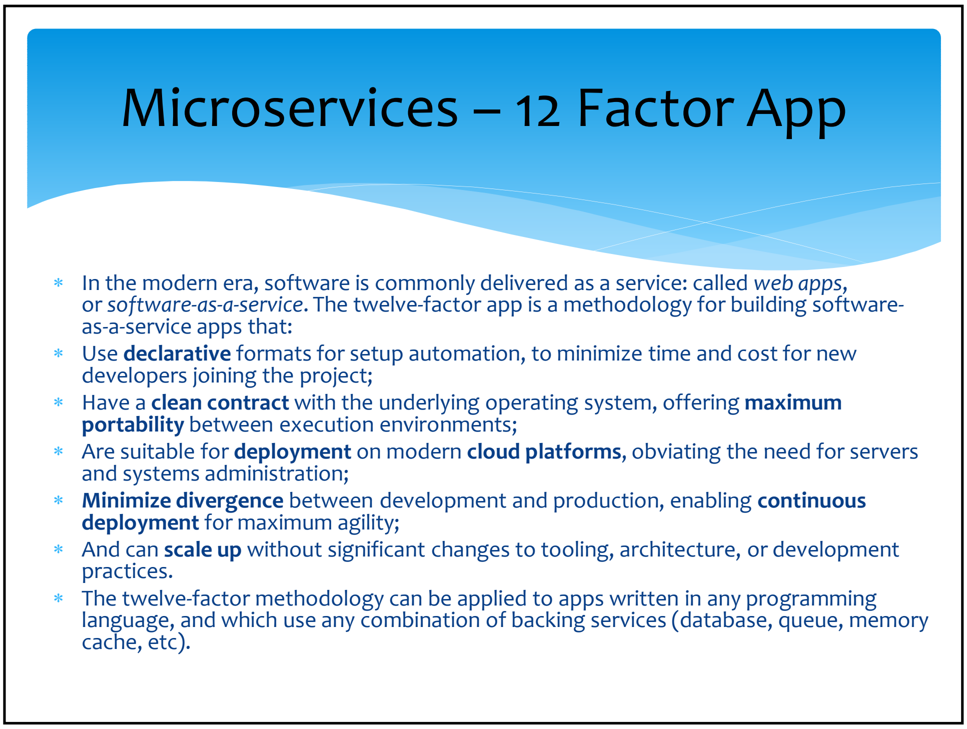
Example : Microservices Architecture



- This is Microservices architecture , One large Application divided into multiple collaborating processes

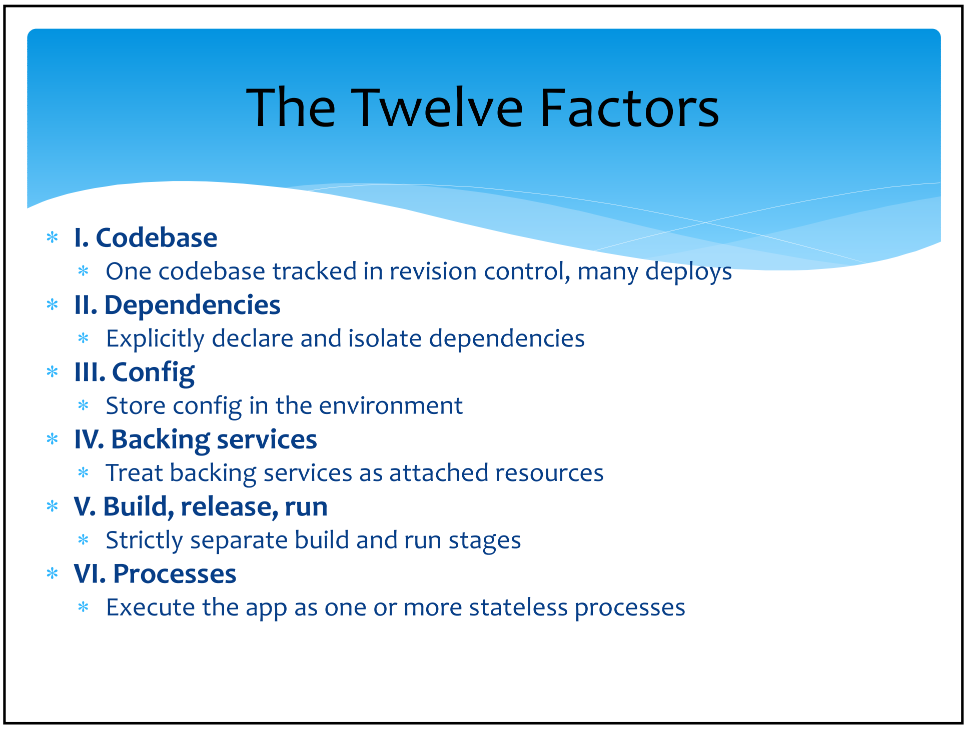
Microservices Benefits

- Smaller code base is easy to maintain.
- Easy to scale as individual component.
- Technology diversity i.e. we can mix libraries, databases, frameworks etc.
- Fault isolation i.e. a process failure should not bring whole system down.
- Better support for smaller and parallel team.
- Independent deployment of various components()
- Reduced Deployment time



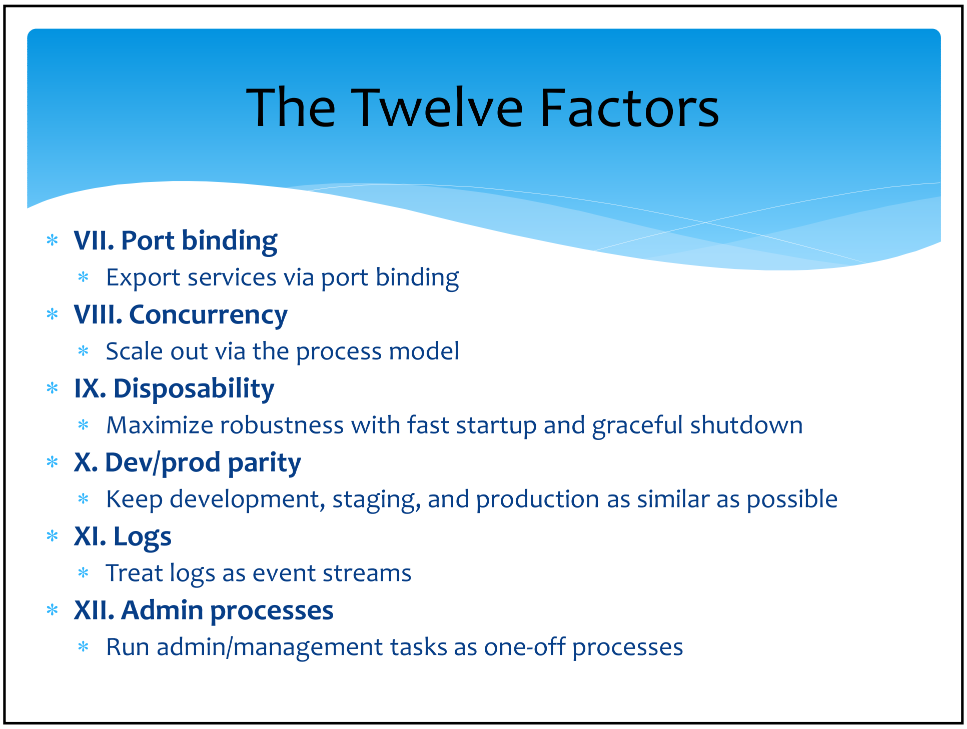
Microservices – 12 Factor App

- In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:
- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.
- The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).



The Twelve Factors

- **I. Codebase**
  - One codebase tracked in revision control, many deploys
- **II. Dependencies**
  - Explicitly declare and isolate dependencies
- **III. Config**
  - Store config in the environment
- **IV. Backing services**
  - Treat backing services as attached resources
- **V. Build, release, run**
  - Strictly separate build and run stages
- **VI. Processes**
  - Execute the app as one or more stateless processes



The Twelve Factors

- **VII. Port binding**
  - Export services via port binding
- **VIII. Concurrency**
  - Scale out via the process model
- **IX. Disposability**
  - Maximize robustness with fast startup and graceful shutdown
- **X. Dev/prod parity**
  - Keep development, staging, and production as similar as possible
- **XI. Logs**
  - Treat logs as event streams
- **XII. Admin processes**
  - Run admin/management tasks as one-off processes