**Topics:**

* History of Microservices
* What is Micro services?
* Monolithic Architecture
* Challenges of Monolithic Architecture
* Reasons for using Microservice
* Benefits of Microservices
* Advantages of Microservices:
* Restrictions of Microservices
* Microservices with Spring Boot

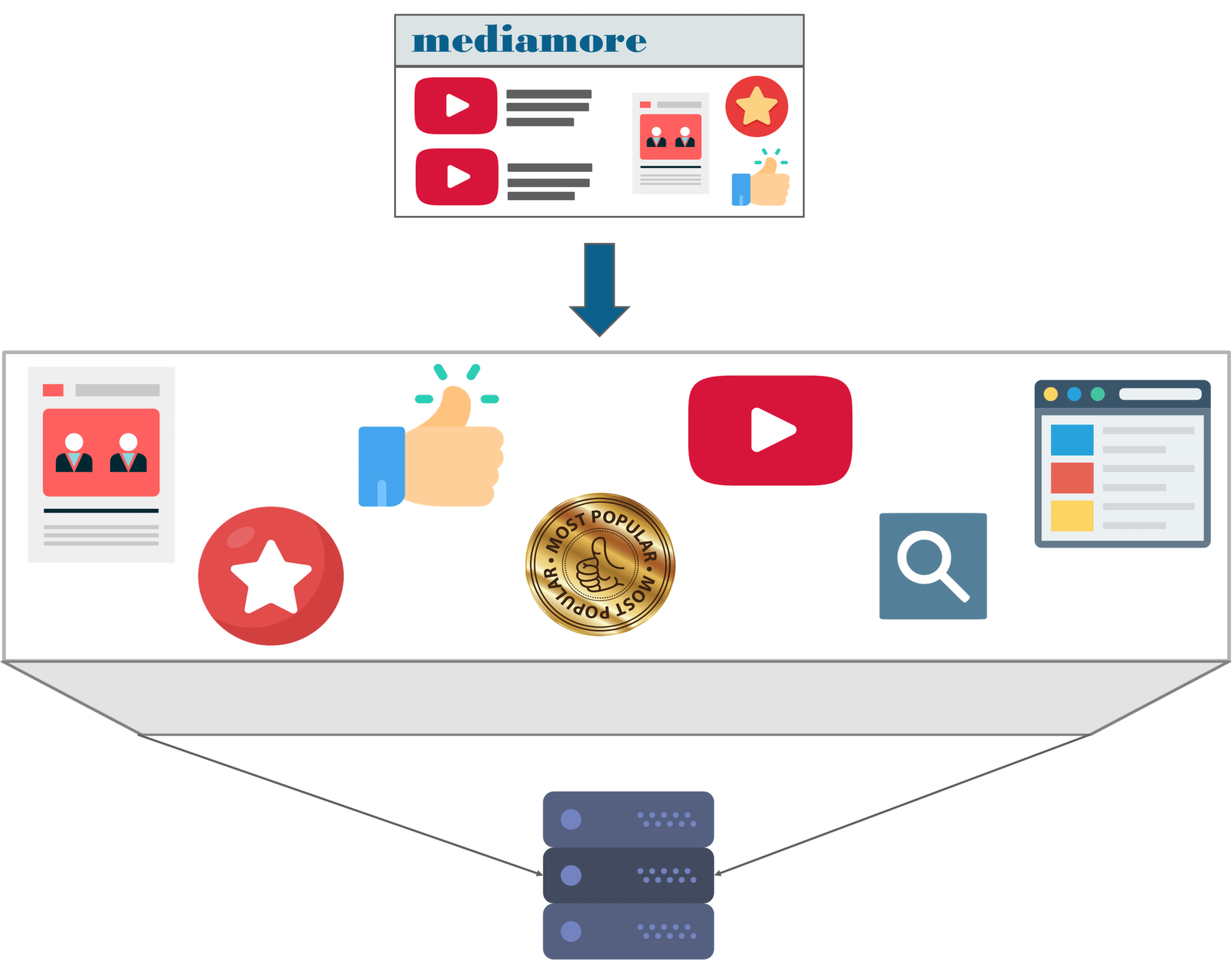
**History of Microservices:**

* Micro services were termed by **Dr Peter Rodgers in 2005** and was initially known as micro web services.

**What is Micro services?**

* Microservices is an architecture wherein all the components of the system are put into individual components, which can be **built, deployed, and scaled individually**.
* A microservice is a **small, loosely coupled distributed service**.
* Microservice Architectures evolved as a solution to the scalability and innovation challenges with Monolith architectures (Monolith applications are typically huge – more 100, 000 line of code). It allows you to take a large application and decompose or break into easily manageable small components with narrowly defined responsibilities.
* The main principle behind micro services is to break up single large ‘**monolithic’** system into multiple independent components/processes.
* Microservices architecture allows decoupled components to be built and deployed independently to integrate into a single larger system. Saying that, it means that these individual components are **loosely coupled** with each other, each having a **tight cohesion**.
* **Tight cohesion** means that the components are in itself very specific to what they need to do.
* These components interact with each other through a standard **XML/JSON** interface (**lightweight** language agnostic communication mechanisms), irrespective of the technologies used to create the component.
* The opposite of a Microservices architecture would be a “**monolithic**” application where all functionality is part of a single program running in a single environment.
* Also to let you know, some of the well-known large companies who have successfully applied microservices architectures are: **Amazon, eBay, Gilt, Netflix, PayPal, and Twitter**.
* Microservices are increasingly used in the development world as developers work to create larger, more complex applications that are better developed and managed as a combination of smaller services that work cohesively together for larger, application-wide functionality.
* Tools are rising to meet the need to think about and build apps using a piece-by-piece methodology that is, frankly, less mind-boggling than considering the whole of the application at once.
* Microservices are a form of service-oriented architecture style (one of the most important skills for Java developers) wherein applications are built as a collection of different smaller services rather than one whole app.
* Instead of a monolithic app, you have several independent applications that can run on their own and may be created using different coding or programming languages.
* Big and complicated applications can be made up of simpler and independent programs that are executable by themselves.
* These smaller programs are grouped together to deliver all the functionalities of the big, monolithic app.
* Microservices captures your business scenario, answering the question “What problem are you trying to solve?” It is usually developed by an engineering team with only a few members and can be written in any programming language as well as utilize any framework.
* Each of the involved programs is independently versioned, executed, and scaled.
* These microservices can interact with other microservices and can have unique URLs or names while being always available and consistent even when failures are experienced.

**Monolithic Architecture**:



* Refer to the above diagram. We can infer that all the features such as the search, user-info, recommendations, video playlist and others are put on a single database using single code.
* Now, let me tell you the challenges faced by the developers while using a monolithic framework by using some scenarios.

**Challenges of Monolithic Architecture:**

* **Scenario 1:** **Scalability:**
  + Let’s assume that the developers want to update the playlist according to most popular tv shows and also simultaneously want to update all videos to HD quality.
  + The developers cannot scale the application simultaneously. New instances of the same application have to be created every time a new feature has to be developed or deployed.
* **Scenario 2: Agility:**
  + Assume that developers want to make immediate changes in the application.
  + The monolithic application can definitely accommodate these changes. But, the problem here is that the developers have to rebuild the code for every small change.
* **Scenario 3: Hybrid Technologies:**
  + Suppose developers of this application are comfortable with various technologies like JAVA, C++, .NET, C#.
  + Even though they are comfortable with various technologies, they still have to build large and complex applications on a single technology.
* **Scenario 4: Fault Tolerance:**
  + Let’s suppose that a specific feature is not working in the application.
  + The complete system goes down because of this problem. In order to tackle this problem, the application has to be re-built, re-tested and also re-deployed.
  + So, how did the developers of mediamore overcome these complexities?
  + Developers thus decided to re-architect their monolithic application into multiple individual deployable components, called as microservices.
  + Here lies the million-dollar question!

**Reasons for using Microservice:**

In monolith application, there are few challenges:

* For a large application, it is difficult to understand the complexity and make code changes fast and correctly, sometimes it becomes hard to manage the code.
* Applications need extensive manual testing to ensure the impact of changes.
* For small change, the whole application needs to be built and deployed.
* The heavy application slows down start-up time.
* There are several benefits to using microservices. For one, because these smaller applications are not dependent on the same coding language, the developers can use the programming language that they are most familiar with.
* That helps developers come up with a program faster with lower costs and fewer bugs.
* The agility and low costs can also come from being able to reuse these smaller programs on other projects, making it more efficient.

**Benefits of Microservices:**

* **Small Modules:**
  + Application is broken into smaller modules which are easy for developers to code and maintain.
* **Easier Process Adaption:**
  + By using microservices, new Technology & Process Adaption becomes easier. You can try new technologies with the newer microservices that we use.
* **Independent scaling:**
  + Each microservice can scale independently via X-axis scaling (cloning with more CPU or memory) and Z-axis scaling (sharding), based upon their needs.
* **Unaffected:**
  + Large applications remain largely unaffected by the failure of a single module.
* **DURS:**
  + Each service can be independently DURS (deployed, updated, replaced, and scaled).

**Advantages of Microservices:**

Some of the major advantages of using Microservices are:

* It gives developers the freedom to independently develop and deploy services.
* Improves the scalability with the ability to individually scale services as needed.
* Better fault tolerance, which means that if one microservice fails, the others will continue to deliver.
* Code for different services can be written in different languages.
* The microservice architecture enables continuous delivery.

**Restrictions of Microservices:**

* **Configuration Management:**
  + As it becomes granular the headache comes for configuring the services and monitoring those. You need to maintain configurations for hundreds of components across environments.
* **Debugging:**
  + Tracking down the service failure is painstaking job. You might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.
* **Automation:**
  + Because there are a number of smaller components instead of a monolith, you need to automate everything – Builds, Deployment, Monitoring etc.
* **Testing:**
  + Needs a greater effort for end to end testing as it needs all the dependent services to be up and running.

**Microservice Frameworks for Java:**

There are several microservices frameworks that you can use for developing for Java. Some of these are:

1. **Spring Boot:**
   * This is probably the best Java microservices framework that works on top of languages for Inversion of Control, Aspect Oriented Programming, and others.
2. **Jersey:** 
   * This open source framework supports JAX-RS APIs in Java is very easy to use.
3. **Swagger:** 
   * Helps you in documenting API as well as gives you a development portal, which allows users to test your APIs.
4. **Dropwizard:**
   * Dropwizard pulls together stable, mature libraries from the Java ecosystem into a simple, light-weight package that lets you focus on getting things done.
5. **Restlet:**
   * Restlet Framework helps Java developers build better web APIs that follow the REST architecture style.
6. **Spark:**
   * A micro-framework for creating web applications in Kotlin and Java 8 with minimal effort.
7. Others that you can consider include: Dropwizard, Ninja Web Framework, Play Framework, RestExpress, Restlet, Restx, and Spark Framework.

**Microservices with Spring Boot:**

Spring Boot projects include:

1. **Spring IO Platform:** Enterprise grade distribution for versioned applications.
2. **Spring Framework:** For transaction management, dependency injection, data access, messaging, and web apps.
3. **Spring Cloud:** For distributed systems and used for building or deploying your microservices.
4. **Spring Data:** For microservices that are related to data access, be it map-reduce, relational or non-relational.
5. **Spring Batch:** For high levels of batch operations.
6. **Spring Security:** For authorization and authentication support.
7. **Spring REST** Docs: For documenting RESTful services.
8. **Spring Social:** For connecting to social media APIs.
9. **Spring Mobile:** For mobile Web apps.