Introduction to Micro-Services with Spring Boot

Prakash Badhe
prakash.badhe@vishwasoft.in

About Me: Prakash Badhe

- Technology Practices Trainer, Consultant.
- Enterprise Technologies Consultant-Trainer for 19+ years
- Passion for technologies and frameworks
- Proficient in application frameworks, libraries and tools.
- Proficient in DevOps tools and technologies.
- Proficient in Docker, Kubernetes etc.
- Supports agile technical practices and setup in agile development and production environments.
- Worked with Symantec,Samsung,PTC,Cognizant etc. for setting up the DevOps environments.

About you..

- **Prerequisite** : Experience with server side java applications development and deployment.
- Linux system commands exposure.
- Tell about your
- **Objectives for this Training**
- **Job Role and skill-Set : In web application development.**
- **Experience in different Technologies and tools.**
- **Prior experience with Virtual machines, Docker or any other tools on Linux platforms.**

Setup

- Windows 10/Linux 64 bit
- Adobe PDF Reader
- JDK1.8 64 bit
- Spring Tool Suite
- Apache Kafka
- MongoDB
- Live Internet connection

SOA : Service Oriented Architecture

Why SOA..?

- Distributed Processing..
- Applications need data sharing and communication in the network across different machines.
- Heterogeneous applications with different programming and different Operating systems.
- Lots of data compatibility issues
- Standard protocol and data format needed.

Applications as services

- At abstract level if the applications are defined as platform independent services, then they can share the data and understand the data exchange...
- The SOA is the answer..
- Service Oriented Architecture.

Service Oriented Architecture

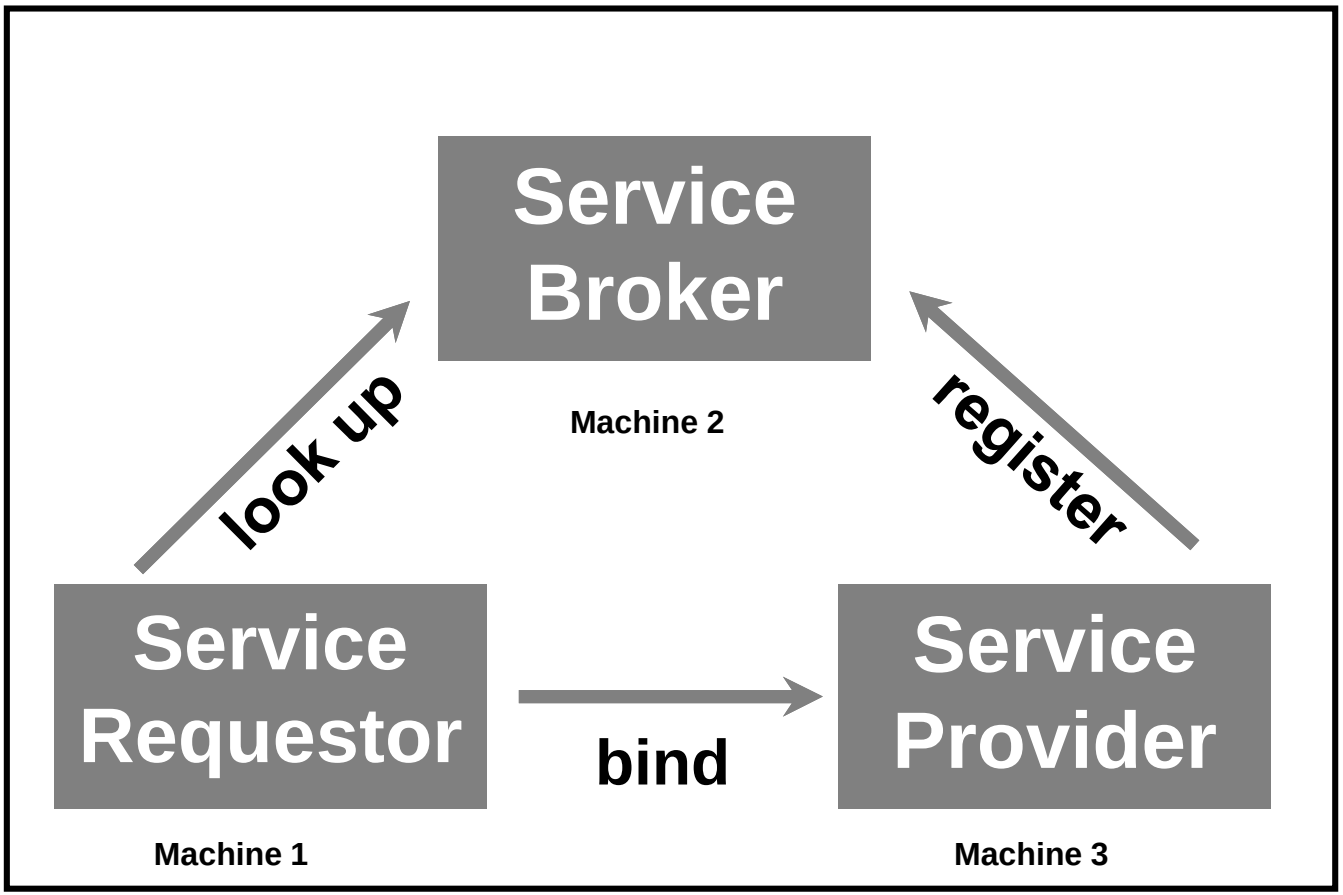
- In a service-oriented architecture, applications are made up from loosely coupled software services, which interact to provide all the functionality needed by the application.
- The applications exchange information across the platforms.
- Each service is generally designed to be very self-contained and stateless to simplify the communication that takes place between them.

Roles Involved In A SOA

Three main roles involved in a service-oriented architecture

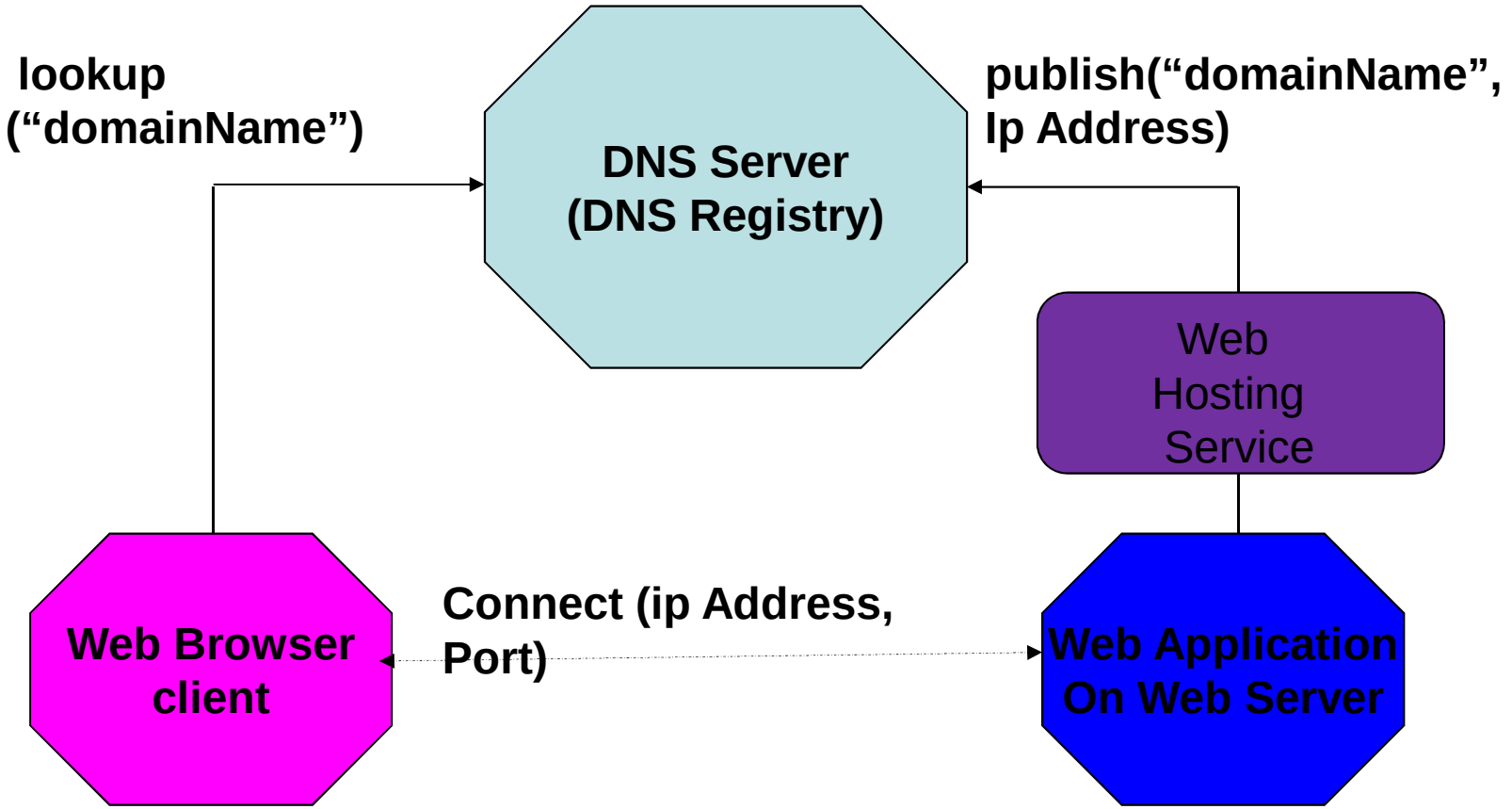
- **Service provider**
- **Service broker**
- **Service requestor**

SOA Communication Model

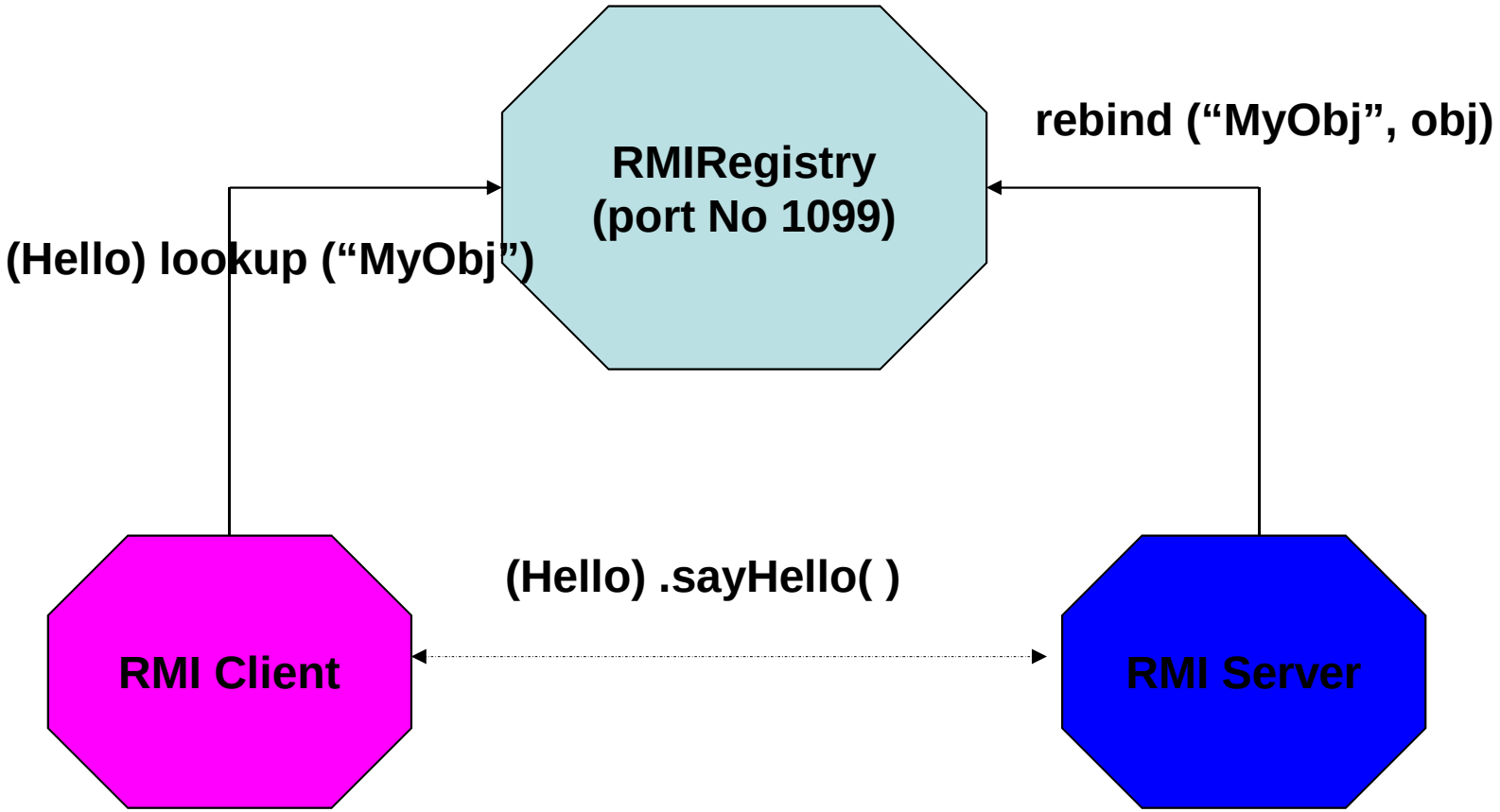


Service Oriented Architecture

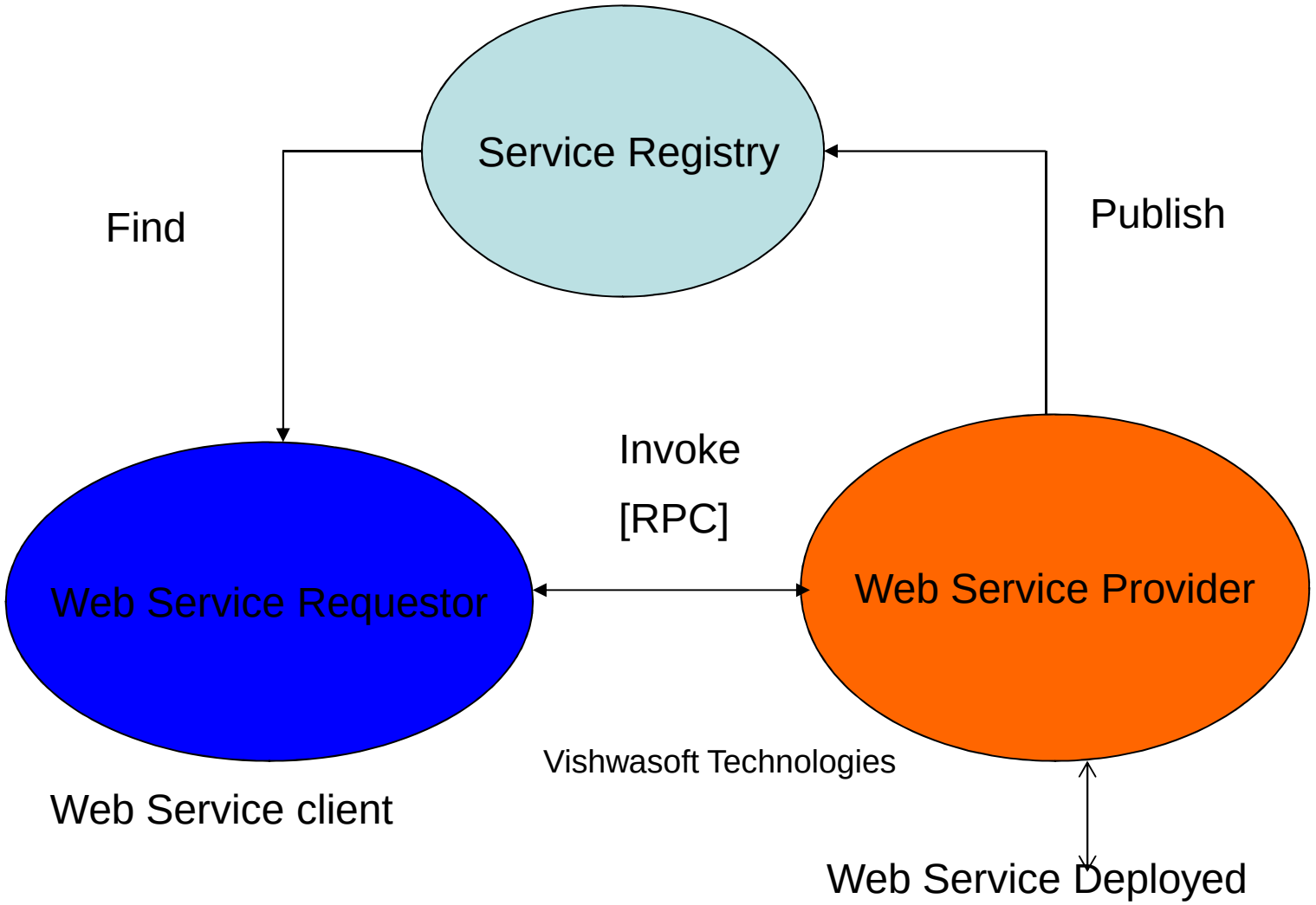
Web Application Process



RPC/RMI Architecture



Web Services Conceptually



Web Programming Model

- Involves client-server interaction on web.
- Uses http protocol and html for information exchange
- Exchange Messages that carry MIME-typed data
- Web applications are loosely coupled than the traditional distributed programming models like RPC, DCOM, and CORBA.

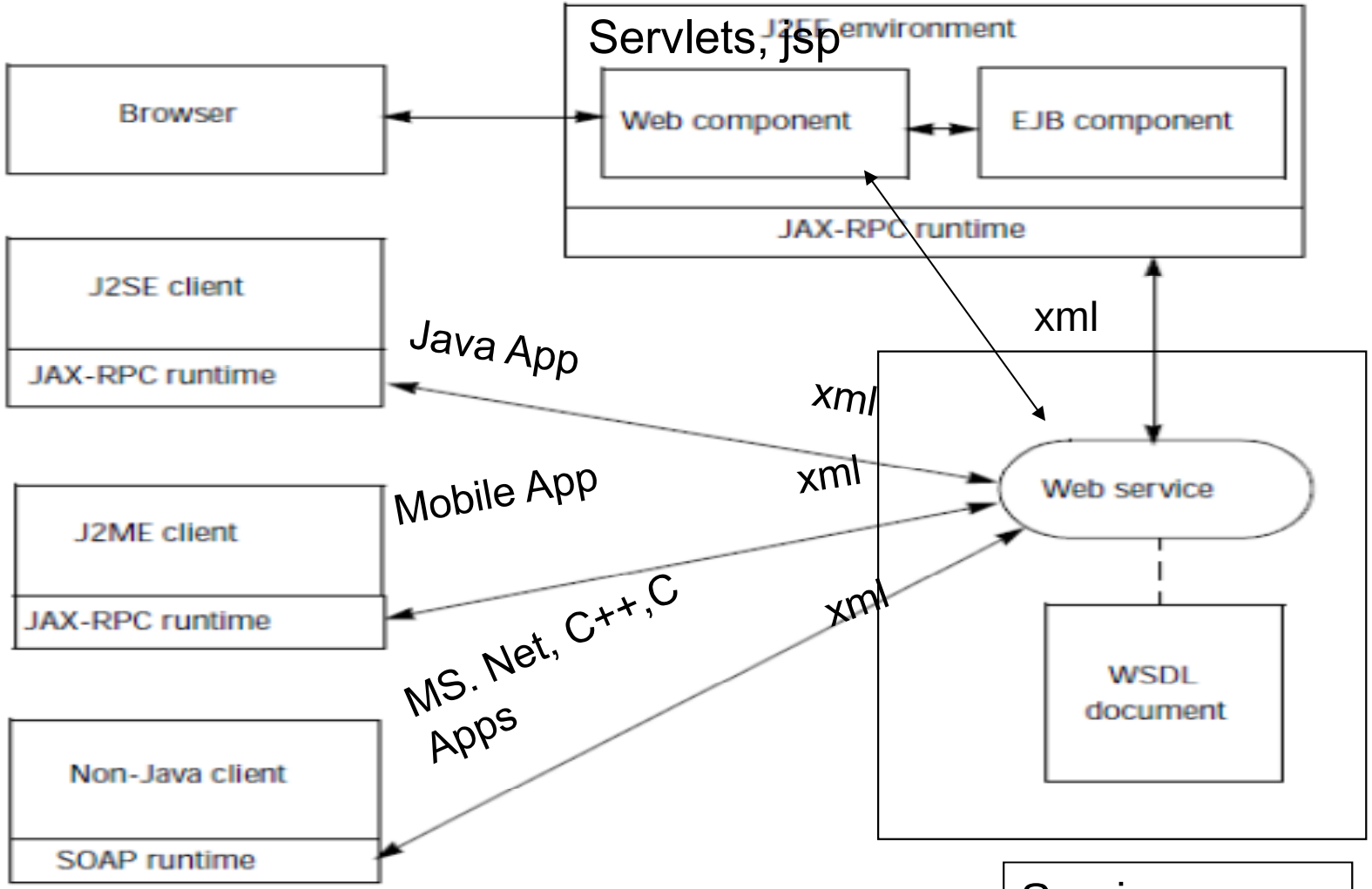
Web Service As SOA

- A Web service is a software application designed to support **interoperable** machine-to-machine interaction over a network
- *Generally web services are small task /service operations supported by applications on the server/s.*
- *These tasks are reusable across different OS platforms and programming language applications.*
- *Web services operate in a distributed environment.*

Web service features

- Loosely coupled Web programming model for use in applications that may not be browser-based.
- Provides a platform for building distributed applications using software components that are..
  - running on different operating systems and devices,
  - written using different programming languages and tools from multiple vendors,
  - all developed and deployed independently

Clients for Web Service



XML Usage

- Web Services fundamentally use XML to standardize the behaviors and data.
- Key XML standards to understand
  - XML : extensible mark up across programming languages.
  - XML Schema specifies xml structure.
  - XML Namespace avoids naming conflicts and categorizes elements as per the functionality

Web Service Standards

- SOAP
- – Simple Object Access Protocol for web service and clients to communicate.
- WSDL
- – Web Service Definition Language for web services to describe their services and other information.
- UDDI
- – Universal Description, Discovery, and Integration protocol for Web Services to discover web services.
- All these are specified in XML format.

Soap Web Services

- Manage language independent interactions across different platforms and applications
- Different versions evolved one by one to support additional features.
- Interoperability is still an issue across different implementations and versions.
- Heavy dependencies on the soap library and other xml implementations.
- Non-xml data format support is limited only in the form of attachments.
- Performance is an issue for mission critical applications.

RESTing the SOAP ?

Enter ‘REST’ Services

- REST is ‘Representational State Transfer’..
- REST is an architectural style rather than a protocol which removes the dependencies on soap and xml standards as wsdl,uddi etc.
- REST supports any understandable data formats across applications.
- REST works currently only with Http.
- REST specifies Resources/data on server rather than actions on them..
- REST specifies transfer of the state of Resource across applications.

Resources on the Web

Resources are not only represented as XML

- XML formats (HTML, XHTML, RSS, etc.)
- JPG, GIF, PNG
- MP3, WAV, OGG
- Anything else that can be on the web.

Resource Nouns

- Important ‘things’ (nouns) are Resources
  - Addressed through a URI
- Uniform interface (verbs)
  - In HTTP: GET, PUT, POST, DELETE
- Verb-noun séparation makes intégration easier
  - GET /customer/45 Instead of getCustomer(45)

REST Nouns and Verbs

- REST works with resources as nouns with their identities.
- The state of these nouns is shared with clients over http methods as verbs(operations)
- The http methods as verbs in REST
  - Get: get the resource state/values
  - Post : post new resource
  - Put : update the resource state
  - Delete :delete the resource on server
  - Options: read the options available with server.
  - Head : set the http headers on server.

REST Commandments

- Give every “thing” an ID
- Link things together
- Use standard methods
- Communicate statelessly

SOAP and REST

- REST is ready for the enterprise
- REST is strong at:
  - Internet scale computing
  - High levels of interoperability
  - Resource Oriented operations
- SOAP/WS is strong at:
  - Complex security (Trust and Federation)
  - Multi-transport services
  - Occasionally connected applications
- In the real world they are typically enabled by a combination of Soap and REST

Java support for REST

- Jax-RS is the java specification standard for implementing and consuming REST web services.
- Implementations a are Jboss RestEasy,Jersey platforms.
- Spring with Spring-Boot
- Oracle and IBM SOA Suite
- Eclipse Micro-Profile

Application Layers

- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
- Business logic — the application’s business logic.
- Database access — data access objects responsible for access the database.
- Application integration — integration with other services (e.g. via messaging or REST API).
- Even with this modular layered architecture, the application is packaged and deployed as a single monolith. (All in One)

Monolith Benefits

- Simple to develop.
- Simple to test. For example you can implement end-to-end testing by simply launching the application and testing it.
- Simple to deploy. You just have to copy the packaged application to a server.
- Simple to scale horizontally by running multiple copies behind a load balancer.

Monolith Drawbacks

- Limitation in size and complexity.
- Application is too large and complex to fully understand and to be able to update with new changes fast and correctly.
- The size of the application can **slow down the start-up** time.
- The **response time and performance** becomes slow because of all execution happening in same process.
- The entire application has to be re-deployed on each update and during down time nothing of it can be accessible.
- Difficult to manage team based development.

Monolith - Reliability

Bug in any module (e.g. memory leak) can potentially bring down the entire process. Moreover, since all instances of the application are identical, that bug impact the availability of the entire application

Scaling the Monolith

Challenging to scale when different modules have conflicting resource requirements.

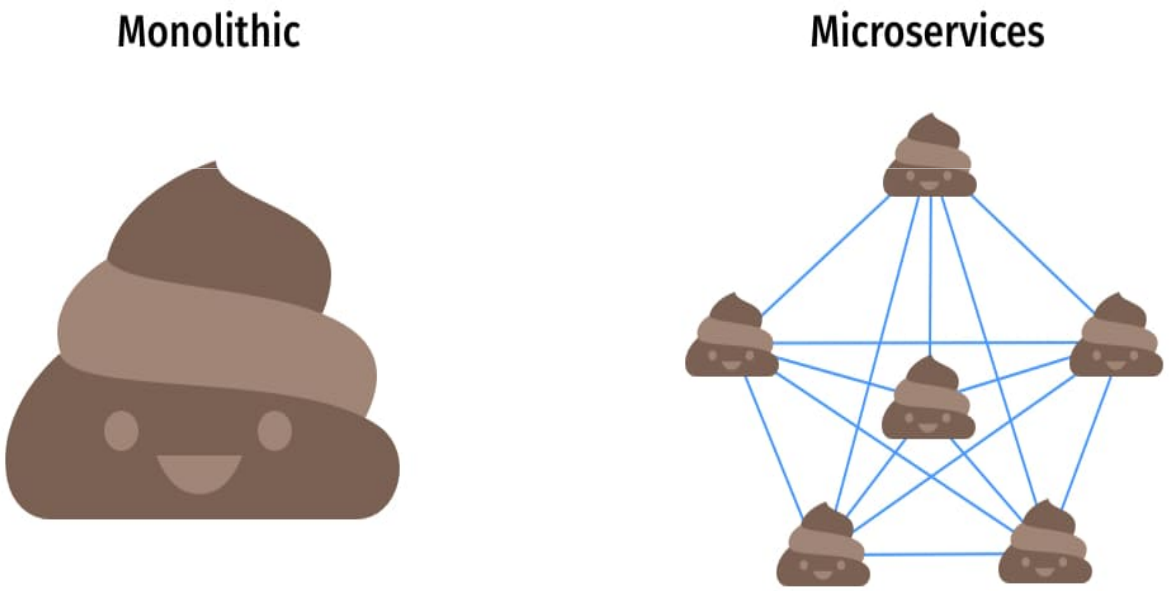
Monolith- Sharing and Reuse

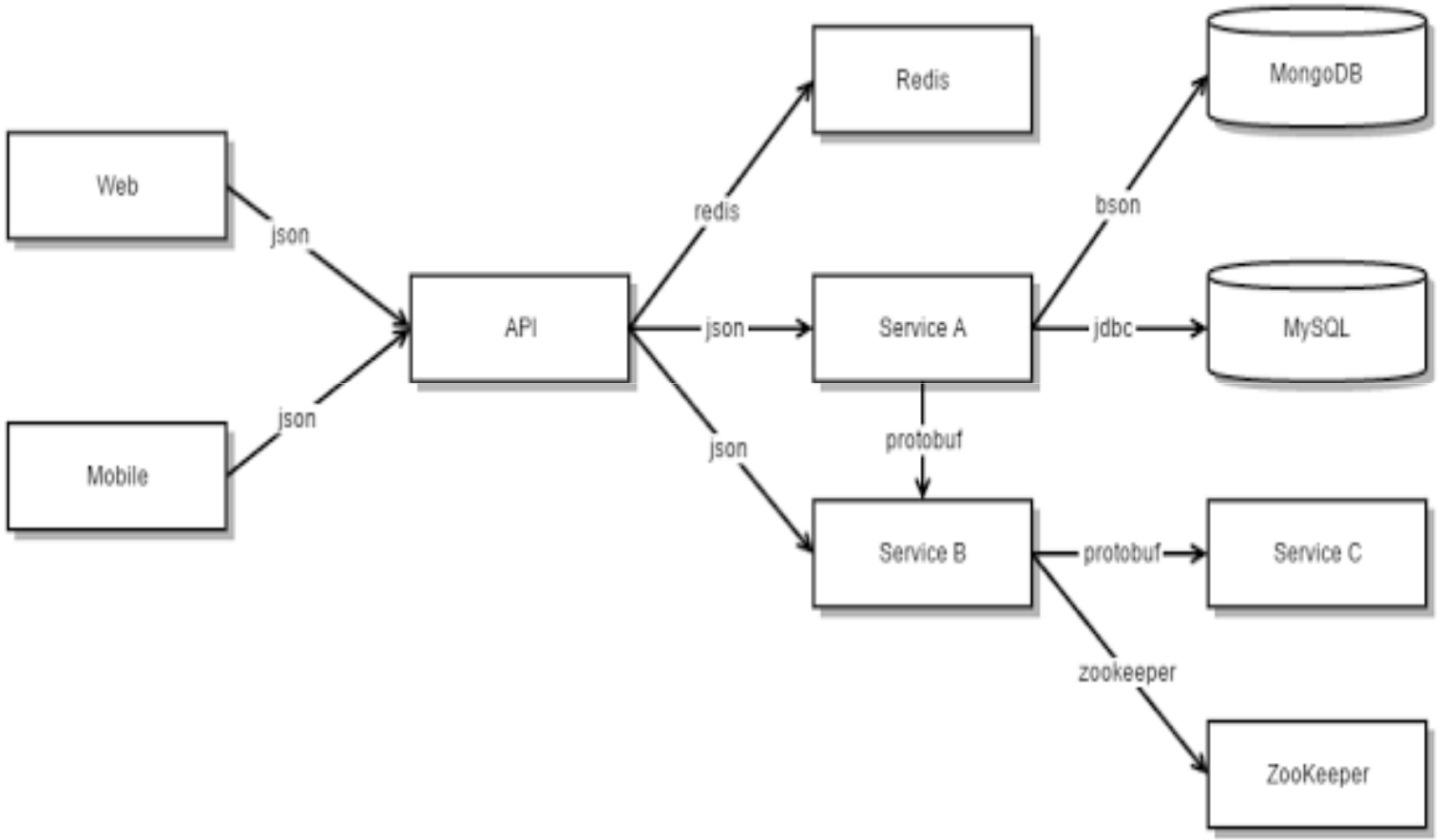
- The good parts of one application cannot be shared or re-used by other applications.
- Sharing of database for other processes and parallel updates is a challenge.

Breaking the Monolith

- The application is architecture is broken into a set of smaller, interconnected services (isolated processes) called as micro-services.
- Some micro-services expose a REST, RPC or message-based API and most services consume APIs provided by other services.
- Some micro-services might implement a web UI.
- Micro-Services are loosely coupled applications as services.

Monolith to Micro-Services




Web
Mobile
json
json
API
redis
json
json
Redis
Service A
Service B
bson
jdbc
protobuf
protobuf
zookeeper
MongoDB
MySQL
Service C
ZooKeeper

MicroService Benefits

- MicroServices reinforce modular structure, which is particularly important for larger teams.
- Services are easier to deploy, and since they are autonomous, are less likely to cause entire system failures when they go wrong.
- With MicroServices you can mix multiple technologies, languages, development frameworks and data-storage technologies.
- Reduced complexity.
- Each MicroService is to be deployed independently. As a result, it makes continuous deployment possible for complex applications

MicroService Benefits

- MicroService architecture enables each service to be scaled independently.
- Performance is improved.
- Better testability — services are smaller and faster to test.
- Better deploy-ability — services can be deployed independently.
- Improved fault isolation; for memory leak in one service then only that service is affected.

Sharing of Database

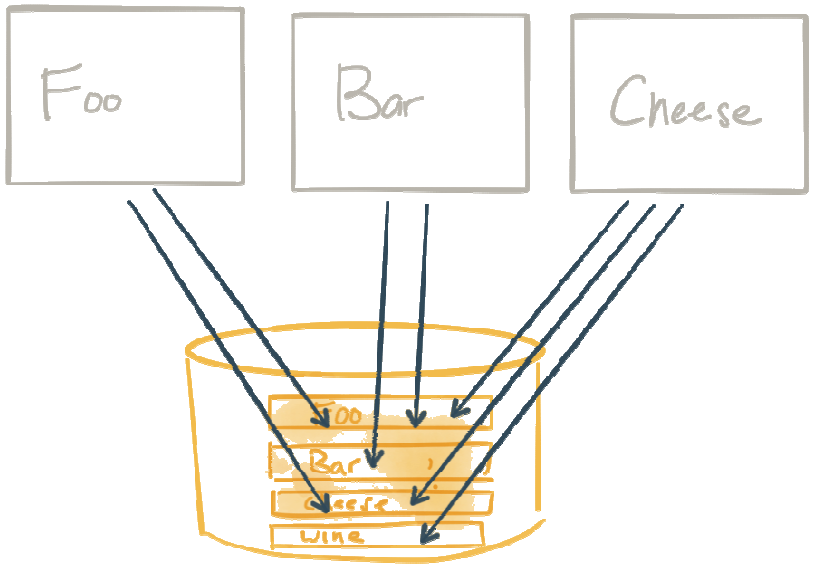
- Instead of sharing a single database schema with other services, **each service has its own database schema.**
- **T**this can result in duplication of some data. However, having a **database schema per service is essential** to get the benefit from micro-services, because it **ensures loose coupling**.

Cost of MicroServices

- Distributed systems are harder to implement, since remote calls are slow and are always at risk of failure. Uniform programming model can reduce the difficulties.
- Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency. With high availability nodes in clusters, this effect can be reduced.
- Need a mature operations team to manage lots of services, which are being redeployed regularly, with automation CI/CD this can be eased.
- The network, technology, bandwidth limits the performance and flexibility.

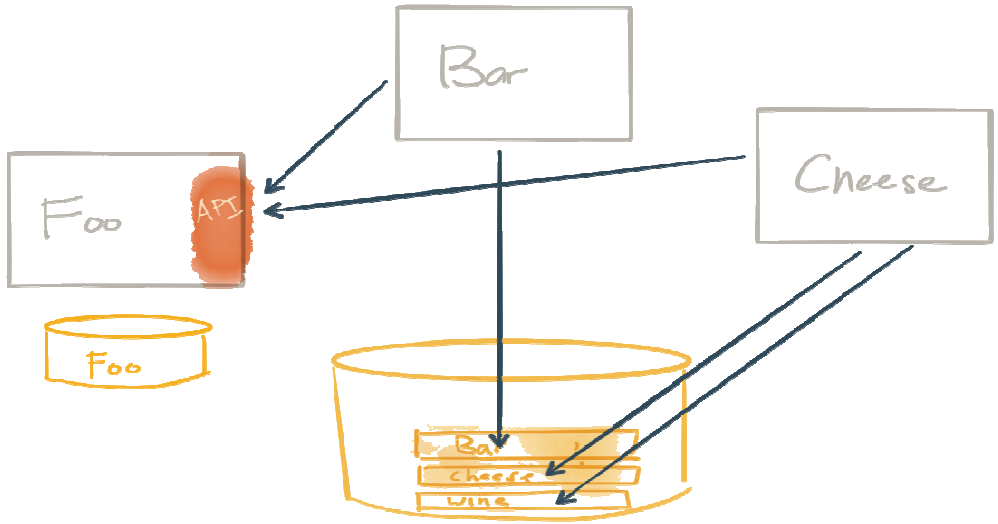
How to de-compose

Identify Modules by domain, categories, business use, related dependencies.

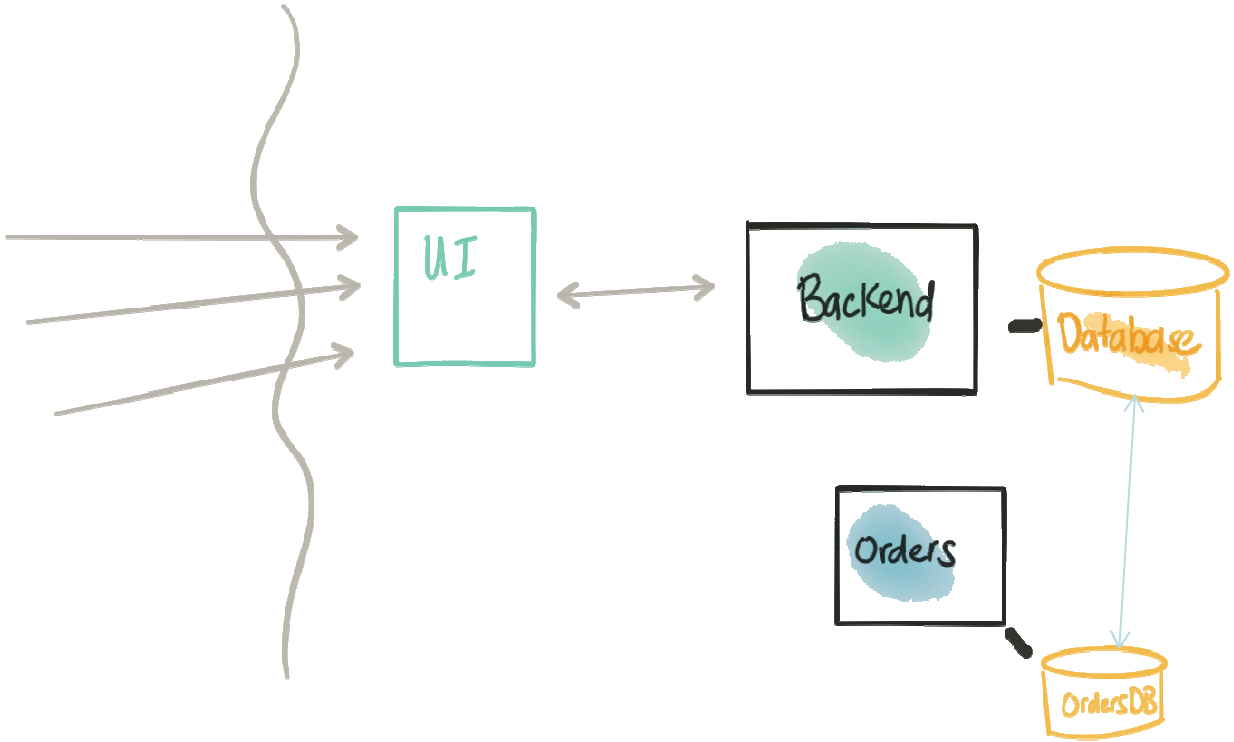


Database break-up

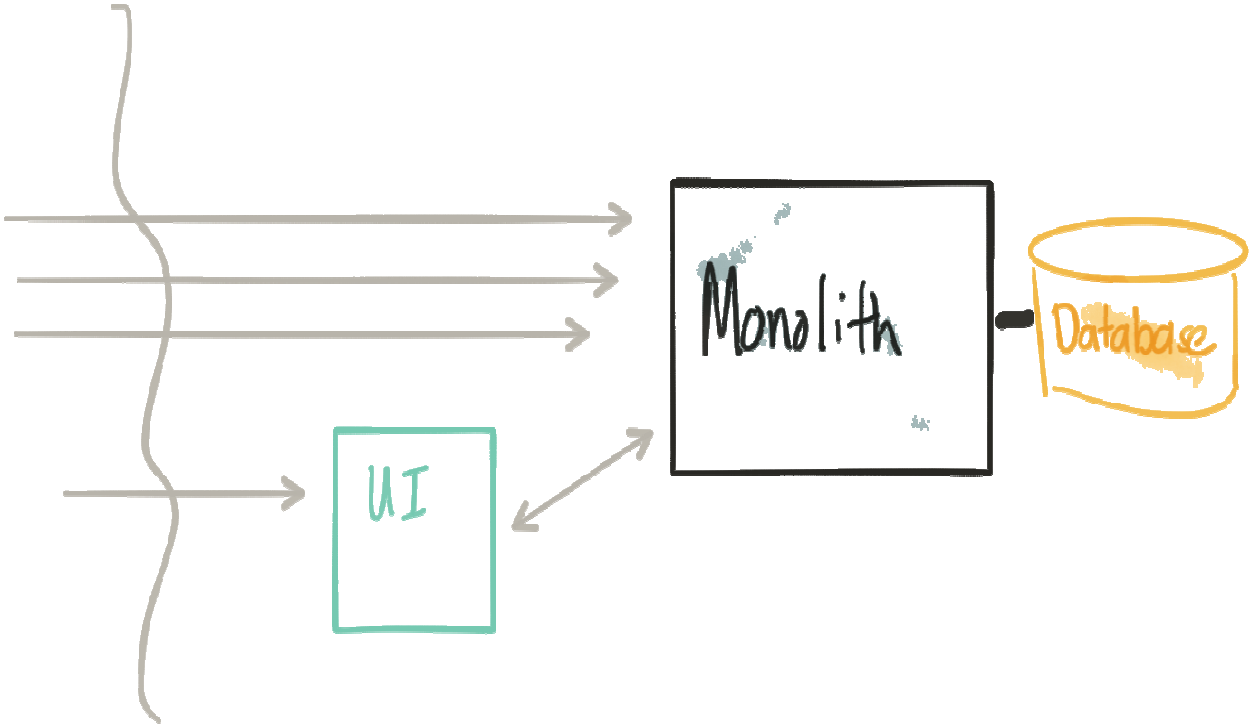
Break out database tables, wrap with service and, update dependencies



Service Modules



UI Extraction



Load Balancing