基于服务的软件架构与云计算

Zhenyan Ji
zhyji@bjtu.edu.cn
Beijing Jiaotong University

Contact information

Zhenyan Ji
Associate Professor
- Office Room: YF706
- Email: zhyji@bjtu.edu.cn
- Telephone: 13621070959

云计算

- 云计算（cloudcomputing）是基于互联网的相关服务的增加、使用和交付模式，通常涉及通过互联网来提供动态易扩展且经常是虚拟化的资源。
- 美国国家标准与技术研究院（NIST）定义：云计算是一种按使用量付费的模式，这种模式提供可用的、便捷的、按需的网络访问，进入可配置的计算资源共享池（资源包括网络，服务器，存储，应用软件，服务），这些资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互。

云计算

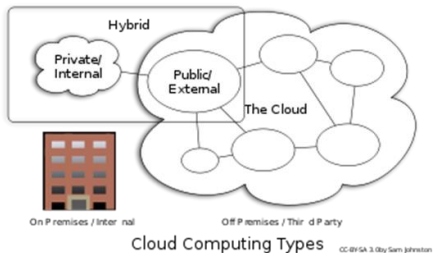
- 云计算常与网格计算、效用计算、自主计算相混淆。
- 网格计算：分布式计算的一种，由一群松散耦合的计算机组成的一个超级虚拟计算机，常用来执行一些大型任务；
- 效用计算：IT资源的一种打包和计费方式，比如按照计算、存储分别计量费用，像传统的电力等公共设施一样；
- 自主计算：具有自我管理功能的计算机系统。
- 事实上，许多云计算部署依赖于计算机集群（但与网格的组成、体系结构、目的、工作方式大相径庭），也吸收了自主计算和效用计算的特点。

Benefits

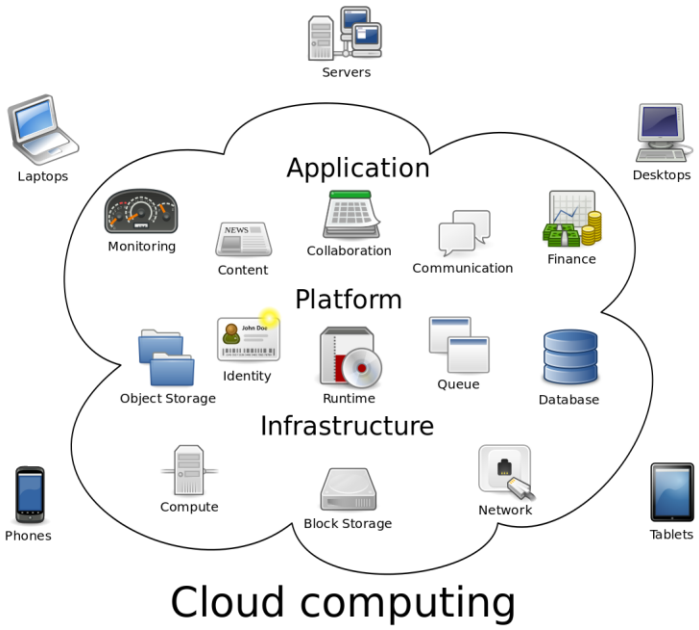
- Cost & management
- Reduced Time to deployment
- Scaling
- Reliability
- Sustainability
  - Hardware not owned

Types of Cloud Computing

- **Public Cloud**: Computing infrastructure is hosted by the vendor.
- **Private Cloud**: Computing architecture is dedicated to the customer and is not shared with other organizations.
- **Hybrid Cloud**: Organizations host some critical, secure applications in private clouds. Non-critical applications are hosted in the public cloud
  - **Cloud bursting**: the organization uses its own infrastructure for normal usage, but cloud is used for peak loads.



Cloud Computing Types



wikipedia:Cloud Computing

1.2云计算——大数据的计算

云计算按服务类型大致分为三类：

专用

通用

将软件作为服务
SaaS（Software as a Service）

针对性更强，它将某些特定应用软件功能封装成服务
如：Salesforce online CRM

将平台作为服务
PaaS（Platform as a Service）

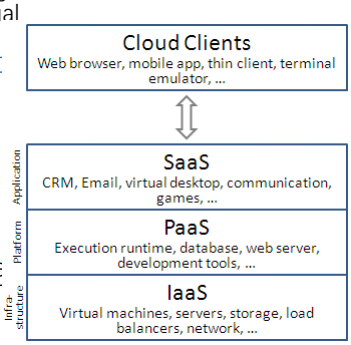
对资源的抽象层次更进一步，提供用户应用程序运行环境
如：Google App Engine
Microsoft Windows Azure

将基础设施作为服务
IaaS（Infrastructure as a Service）

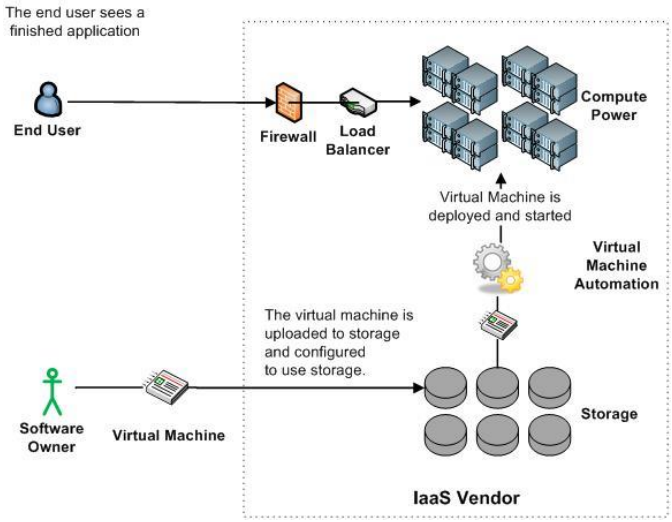
将硬件设备等基础资源封装成服务供用户使用
如：Amazon EC2/S3

Classification of Cloud Computing based on Service Provided

- Infrastructure as a service (IaaS)
  - Offering hardware related services using the principles of cloud computing. These could include storage services (database or disk storage) or virtual servers.
  - Amazon EC2, Amazon S3, Rackspace Cloud Server and Flexiscale.
- Platform as a Service (PaaS)
  - Offering a development platform on the cloud.
  - Google's Application Engine, Microsofts Azure, Salesforce.com's force.com .
- Software as a service (SaaS)
  - Including a complete software offering on the cloud. Users can access a software application hosted by the cloud vendor on pay-per-use basis.
  - Salesforce.coms' offering in the online Customer Relationship Management (CRM) space, Googles gmail and Microsofts hotmail, Google docs.



Infrastructure as a Service (IaaS)



Key Ingredients in Cloud Computing

- Service-Oriented Architecture (SOA)
- Utility Computing (on demand)
- Virtualization
- SAAS (Software As A Service)
- PAAS (Platform AS A Service)
- IAAS (Infrastructure AS A Servie)
- Web Services in Cloud

Cloud in Real World

- Amazon Elastic Compute Cloud
- Google App Engine
- Microsoft Azure
- 百度云
- 阿里云

Hadoop

- Hadoop是一个由Apache基金会开发的分布式系统基础架构。
- 用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。
- Hadoop实现了一个分布式文件系统（Hadoop Distributed File System），简称HDFS。HDFS有高容错性的特点，并且设计用来部署在低廉的（low-cost）硬件上；而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。
- Hadoop的框架最核心的设计就是：HDFS和MapReduce。HDFS为海量的数据提供了存储，则MapReduce为海量的数据提供了计算。

Hadoop

- **What Is Apache Hadoop?**
- The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.
- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.
- The project includes these modules:
- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

What is Spark

- Apache Spark™ is a fast and general engine for large-scale data processing.
- Apache Spark is an open source cluster computing system that aims to make data analytics fast --both fast to run and fast to write
- It builds on the top of Hadoop distributed file system (HDFS).

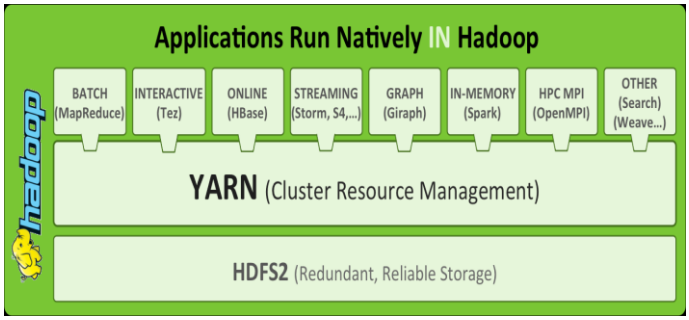
Spark

- Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark是UC Berkeley AMP lab (加州大学伯克利分校的AMP实验室)所开源的类Hadoop MapReduce的通用并行框架，Spark，拥有Hadoop MapReduce所具有的优点；不同于MapReduce的是——Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法。
- Spark 是一种与 Hadoop 相似的开源集群计算环境，两者不同之处使 Spark 在某些工作负载方面表现得更加优越，Spark 启用了内存分布数据集，能够提供交互式查询，还可以优化迭代工作负载。

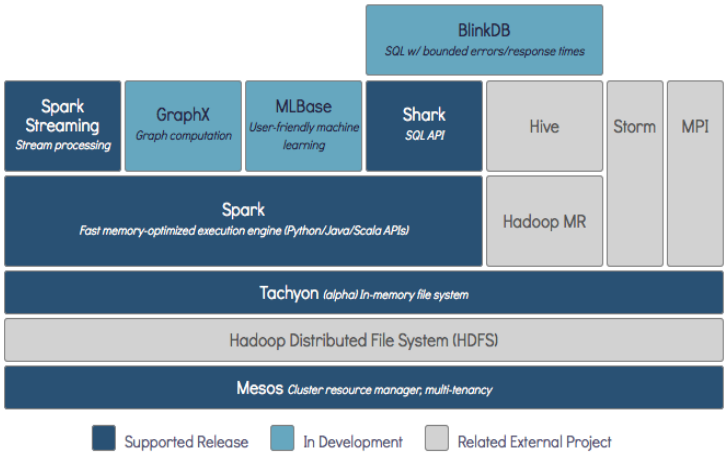
Spark

- Spark 是在 Scala 语言中实现的，它将 Scala 用作其应用程序框架。与 Hadoop 不同，Spark 和 Scala 能够紧密集成，其中的 Scala 可以像操作本地集合对象一样轻松地操作分布式数据集。
- Spark 是为了支持分布式数据集上的迭代作业，它是对 Hadoop 的补充，可以在 Hadoop 文件系统中并行运行。通过名为 Mesos 的第三方集群框架可以支持此行为。Spark 由加州大学伯克利分校 AMP 实验室 (Algorithms, Machines, and People Lab) 开发，可用来构建大型的、低延迟的数据分析应用程序。
- 部署：Local、Standalone、Mesos、YARN…..

Spark on YARN



Spark on Mesos

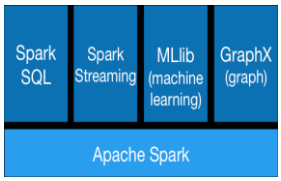


The feature of Spark

Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of high-level tools including Spark SQL, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these frameworks seamlessly in the same application.

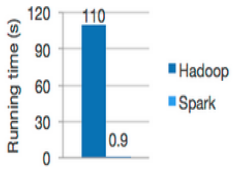


That is "All-In-One"

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

The feature of Spark

Ease of Use

Write applications quickly in Java, Scala or Python.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala and Python shells.

```
file = spark.textFile("hdfs://...")

file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

```
val file = sc.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

light and thin!

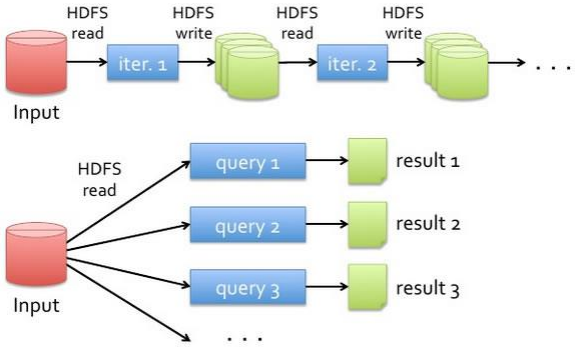
- core code of Spark

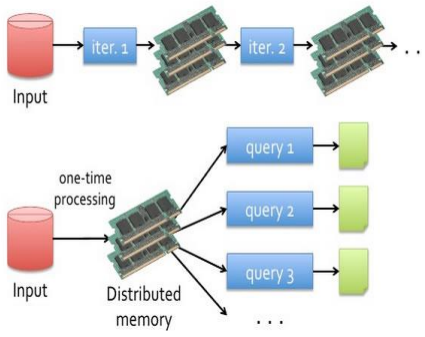
Spark: 20,000 LOC

Hadoop 1.0: 90,000 LOC

Hadoop 2.0: 220,000 LOC

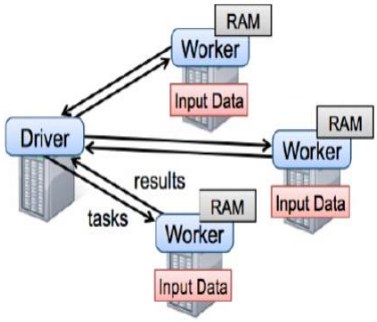
Why so fast? Hadoop VS Spark





More reasons...

- Driver starts many workers, and workers read data from HDFS and turn the data to RDD.
- Then.... deal with RDD in memory !!!



Python

- Python是一种面向对象的解释型计算机程序设计语言，由荷兰人Guido van Rossum于1989年发明，第一个公开发行版发行于1991年。
- Python是纯粹的自由软件， 源代码和解释器遵循 GPL(GNU General Public License)协议。Python语法简洁清晰，特色之一是强制用空白符(white space)作为语句缩进。
- Python具有丰富和强大的库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块（尤其是C/C++）很轻松地联结在一起。常见的一种应用情形是，使用Python快速生成程序的原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写，比如3D游戏中的图形渲染模块，性能要求特别高，就可以用C/C++重写，而后封装为Python可以调用的扩展类库。需要注意的是在您使用扩展类库时可能需要考虑平台问题，某些可能不提供跨平台的实现。
- 7月20日，IEEE发布2017年编程语言排行榜：Python高居首位[2] 。

代码例子

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

for letter in 'Python':     # 第一个实例
   if letter == 'h':
      break
   print '当前字母 :', letter

var = 10                    # 第二个实例
while var > 0:
   print '当前变量值 :', var
   var = var -1
   if var == 5:   # 当变量 var 等于 5 时退出循环
      break

print "Good bye!"
```

Scala

High-level language for JVM

>> Object-oriented + Functional programming (FP)

Statically typed

>> Comparable in speed to Java

>> no need to write types due to type inference

Interoperates with Java

>> Can use any Java class, inherit from it, etc;

>> Can also call Scala code from Java

- Scala 运行在Java虚拟机上，并兼容现有的Java程序。
- Scala 源代码被编译成Java字节码，所以它可以运行于JVM之上，并可以调用现有的Java类库。
- Scala Tutorial: http://www.runoob.com/scala/scala-tutorial.html

Quick Tour

Declaring variables:

```
var x: Int = 7
var x = 7  // type inferred

val y = "hi"  // read-only
```

Java equivalent:

```
int x = 7;

final String y = "hi";
```

Functions:

```
def square(x: Int): Int = x*x

def square(x: Int): Int = {
  x*x
}

def announce(text: String) {
  println(text)
}
```

Last expression in block returned

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

29

Quick Tour

Generic types:

```
var arr = new Array[Int](8)

var lst = List(1, 2, 3)
// type of lst is List[Int]
```

Java equivalent:

```
int[] arr = new int[8];

List<Integer> lst =
  new ArrayList<Integer>();
lst.add(...)
```

Indexing:

```
arr(5) = 7

println(lst(5))
```

Java equivalent:

```
arr[5] = 7;

System.out.println(lst.get(5));
```

30

Quick Tour

Processing collections with functional programming:

```
val list = List(1, 2, 3)

list.foreach(x => println(x))  // prints 1, 2, 3
list.foreach(println)          // same

list.map(x => x + 2)  // => List(3, 4, 5)
list.map(_ + 2)       // same, with placeholder notation

list.filter(x => x % 2 == 1)  // => List(1, 3)
list.filter(_ % 2 == 1)       // => List(1, 3)

list.reduce((x, y) => x + y)  // => 6
list.reduce(_ + _)            // => 6
```

Function expression (closure)

All of these leave the list unchanged (List is Immutable)

Scala Closure Syntax

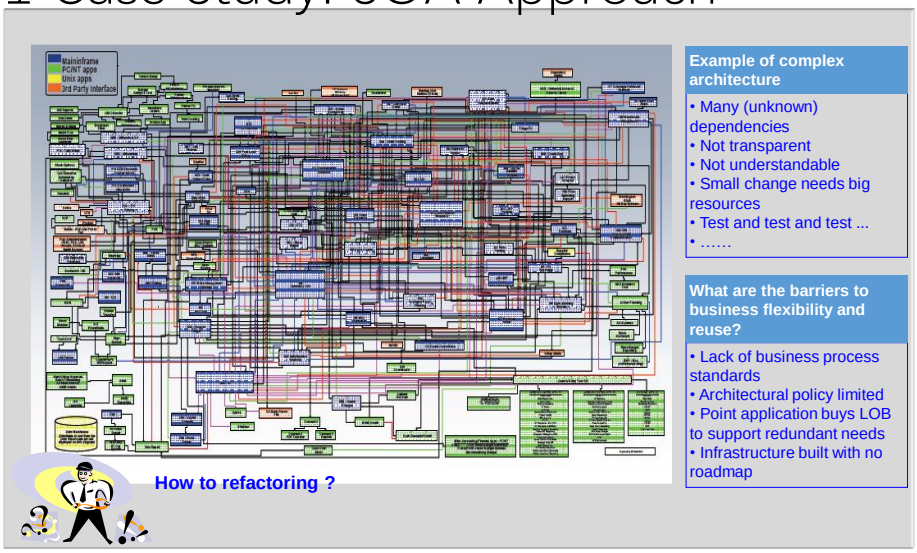
```
(x: Int) => x + 2   // full version

x => x + 2   // type inferred

_ + 2        // when each argument is used exactly once

x => {       // when body is a block of code
  val numberToAdd = 2
  x + numberToAdd
}

// If closure is too long, can always pass a function
def addTwo(x: Int): Int = x + 2

list.map(addTwo)
```

Scala allows defining a "local function" inside another function

32

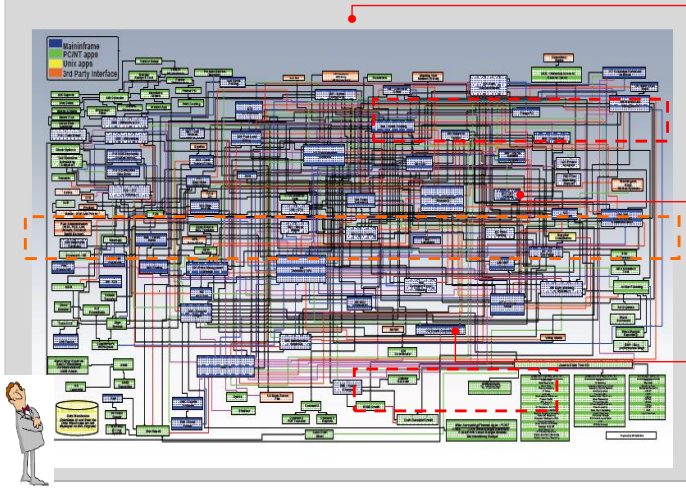
服务（Services）

1.1.1 Case Study: SOA Approach



1.1.1 Case Study: SOA Approach

- In SOA approach



Enterprise Level

- SOA a strategic initiative at the enterprise level
- Suggests a broader Component Business Modeling consulting engagement

Line of Business

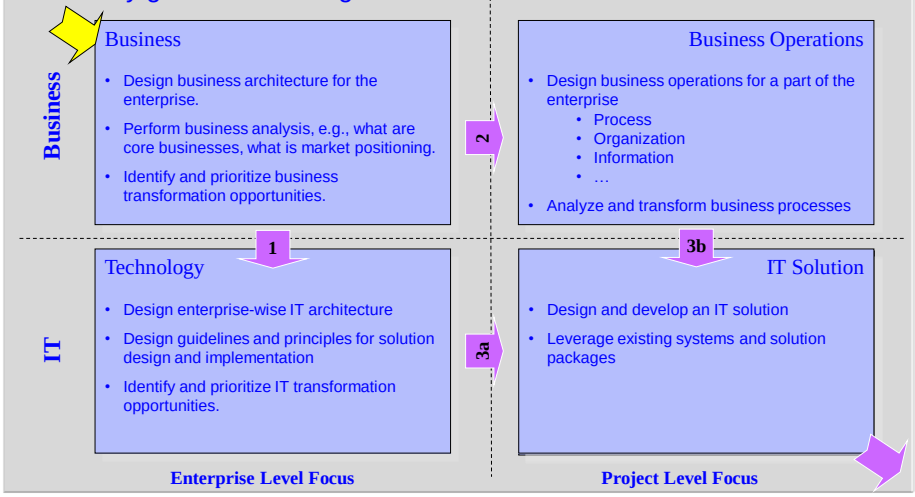
- SOA across a set of related projects under a LOB decision maker
- Client architecture readiness evaluation

Single Project

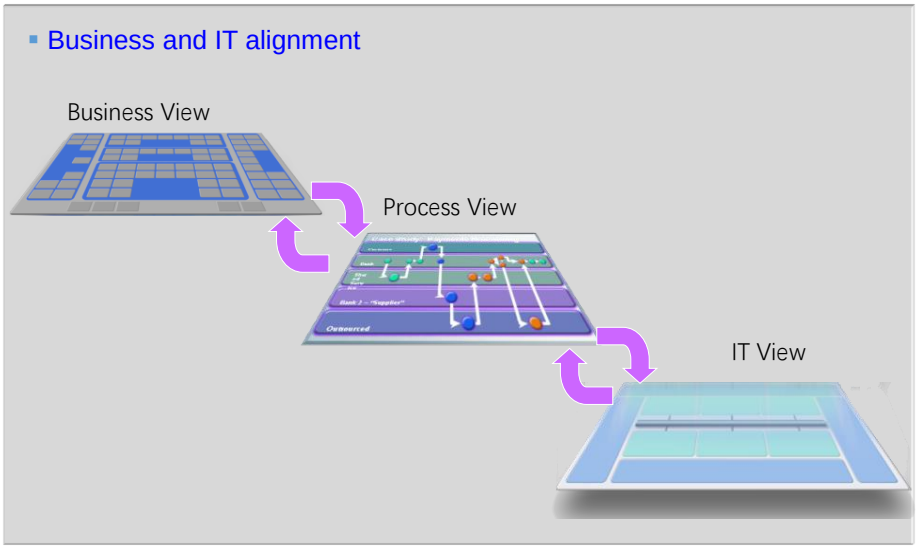
- A way to test the waters and gain experience
- SOA Jumpstart

1.1.1 Case Study: SOA Approach

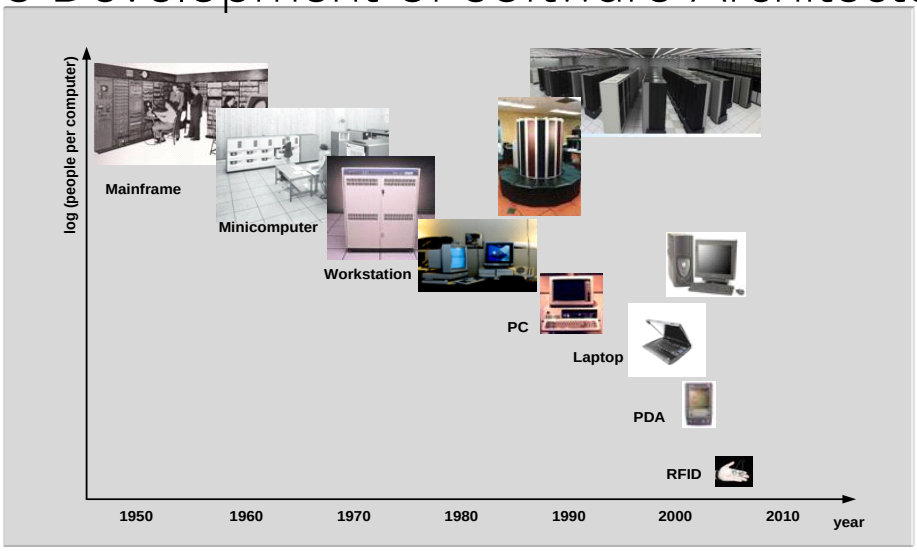
- Primary goal of SOA: align business & IT



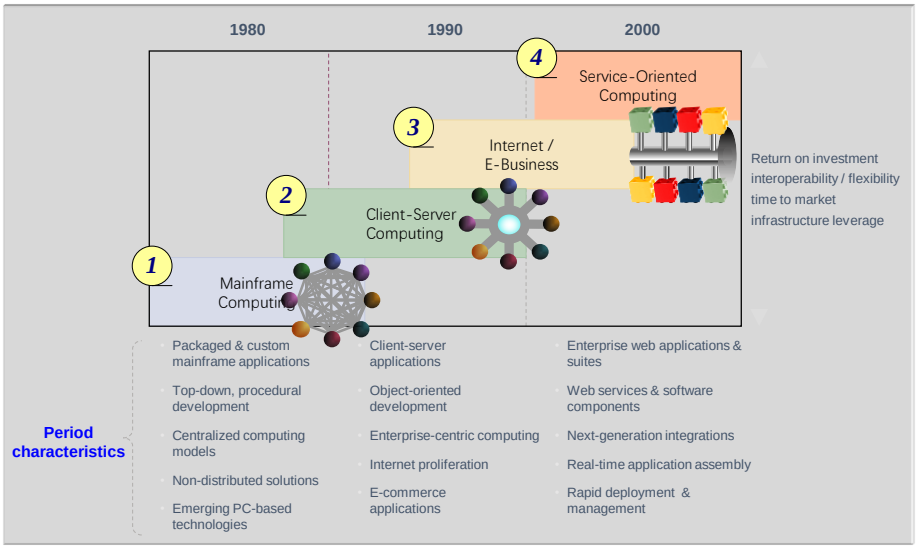
1.1.1 Case Study: SOA Approach



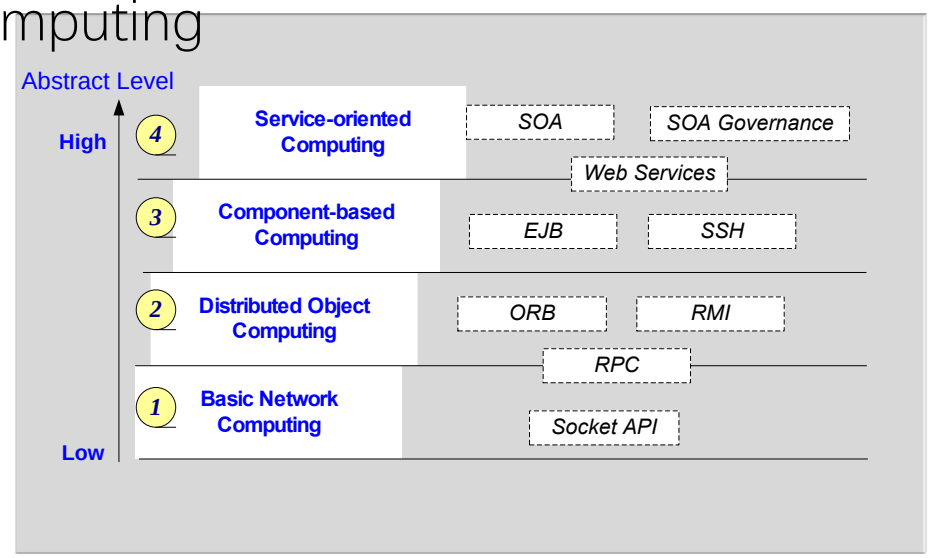
1.1.3 Development of Software Architecture



1.1.3 Development of Software Architecture

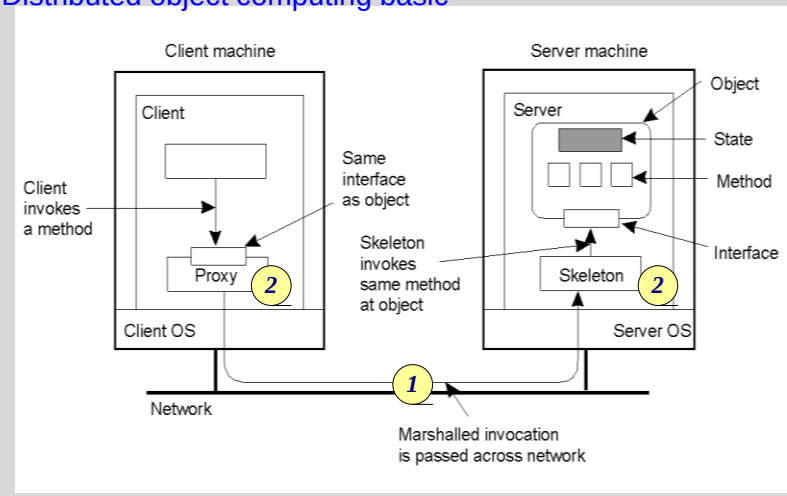


1.1.4 Development of Distributed Computing



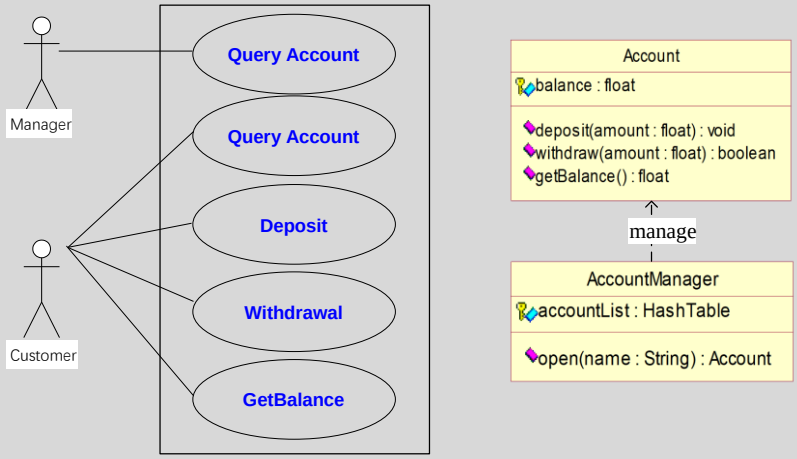
1.1.4 Development of Distributed Computing

- Distributed object computing basic

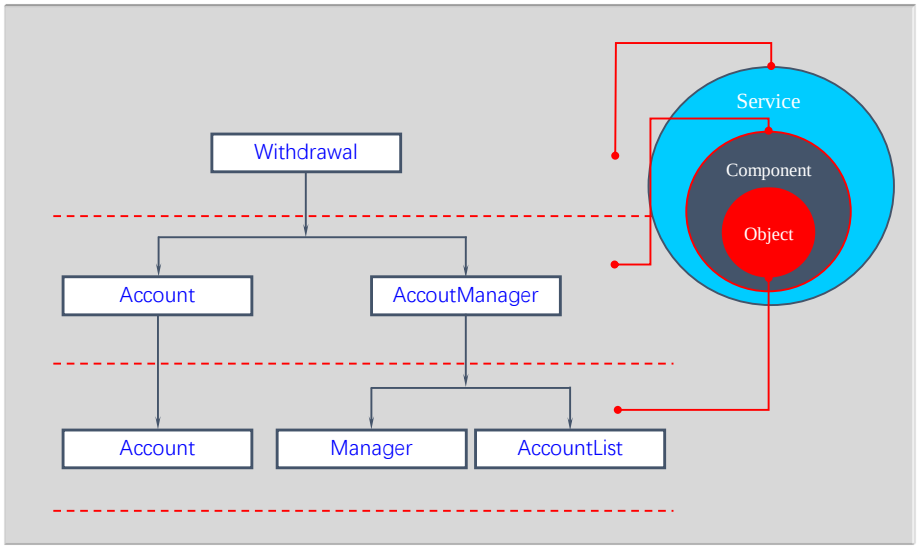


1.1.4 Development of Distributed Computing

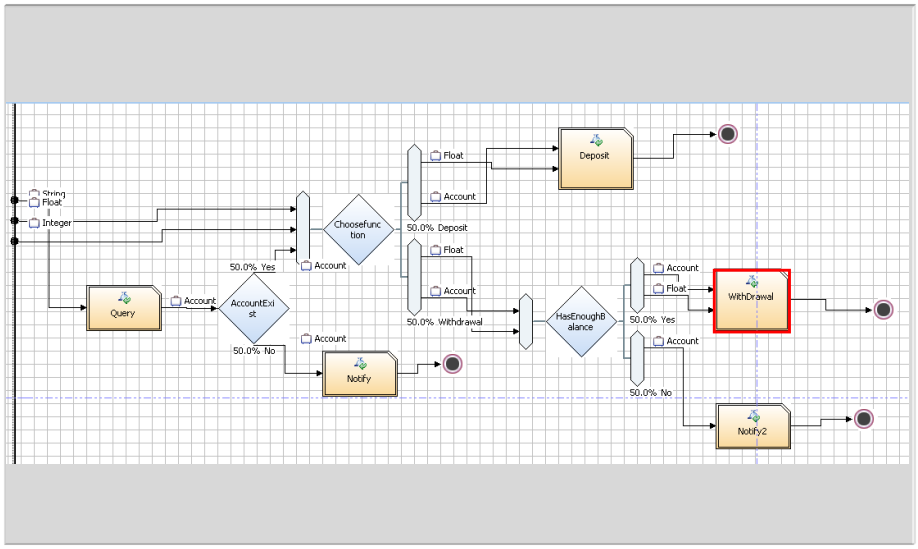
- From OO to Service-oriented



1.1.4 Development of Distributed Computing



1.1.4 Development of Distributed Computing



1.2.1 SOA Entry Points

Information centric approach
Greater value through SOA d*elivering information as a service to people & processes*

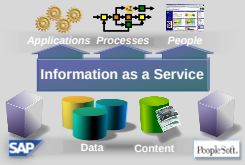
1

Value
Improve business operations and reduce risk with trusted information services delivered in-line and in-context

Why SOA?
Trusted information packaged as services are embedded inline within processes or delivered to people

Start with
Discover and understand information sources, relationships & business context– choose reusable high value data for first services

Next steps
Expand number and scope of services across internal and external processes

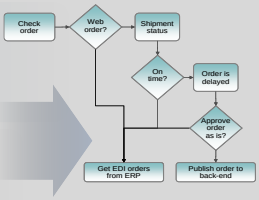


1.2.1 SOA Entry Points

Process Centric Approach
Greater Value through SOA *Business Process Management for Continuous Innovation*

2

Value
Innovative business models deployed quickly with flexible and optimized processes. Measure performance to drive improvement.

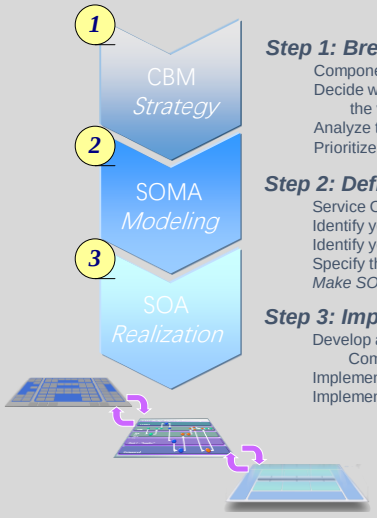


Why SOA?
Modeled processes, converted into services, are re-used, connected and re-deployed more flexibly and quickly with SOA

Start with
A single process – Model an underperforming process. Optimize and deploy as enhanced process.

Next steps
Flexibly link multiple processes across the enterprise & to suppliers / partners. Monitor the process to measure & track performance.

1.2.3 Service-Oriented Modeling and Architecture



Step 1: Break down your business into components

Component Business Modeling
Decide what is strategically important, and what is just operations in the value chain domains
Analyze the different KPIs attached to these components
Prioritize and scope your transformation projects

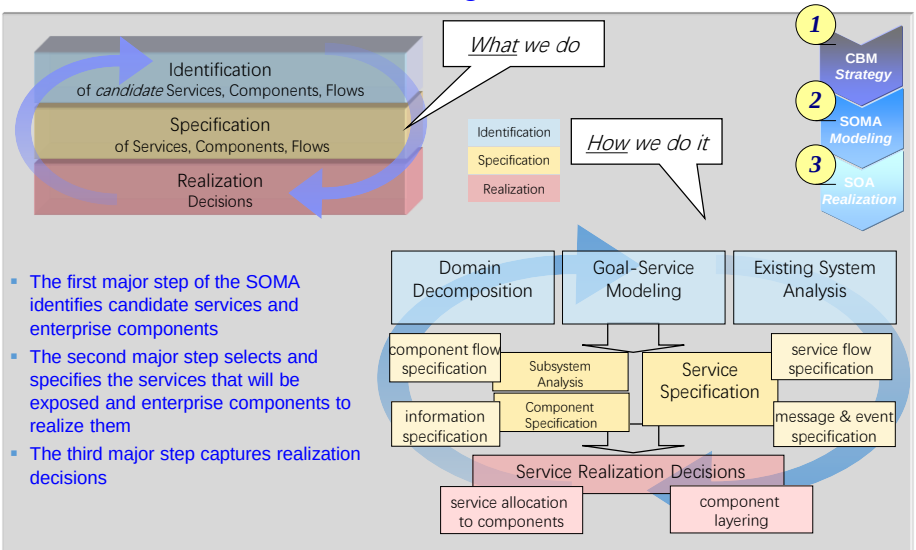
Step 2: Define a Service Model

Service Oriented Model and Architecture
Identify your business services based on your business components
Identify your business processes
Specify the services, processes and components accordingly
Make SOA realization decisions based on architectural decisions

Step 3: Implement a Service Model

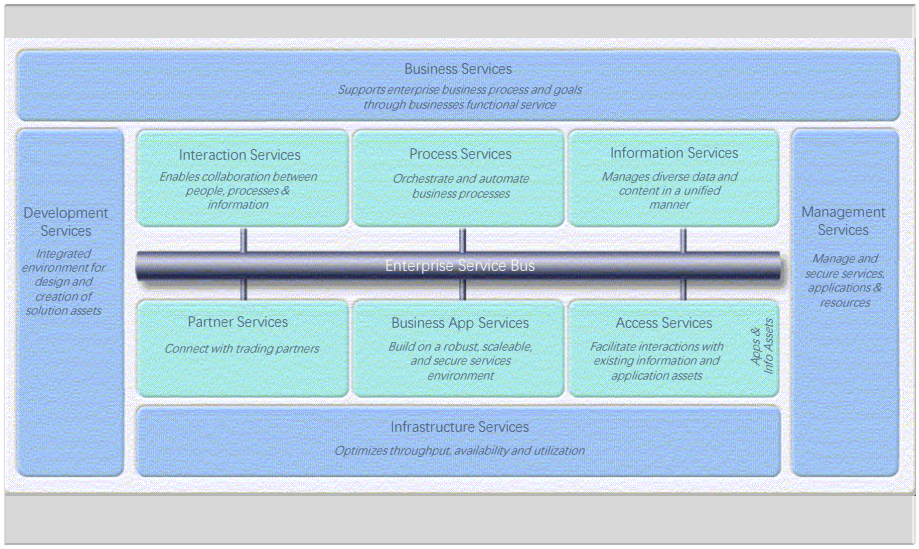
Develop a service-oriented architecture to support the Componentized Business
Implement service based scoping policy for projects
Implement appropriate governance mechanism

1.2.3 Service-Oriented Modeling and Architecture

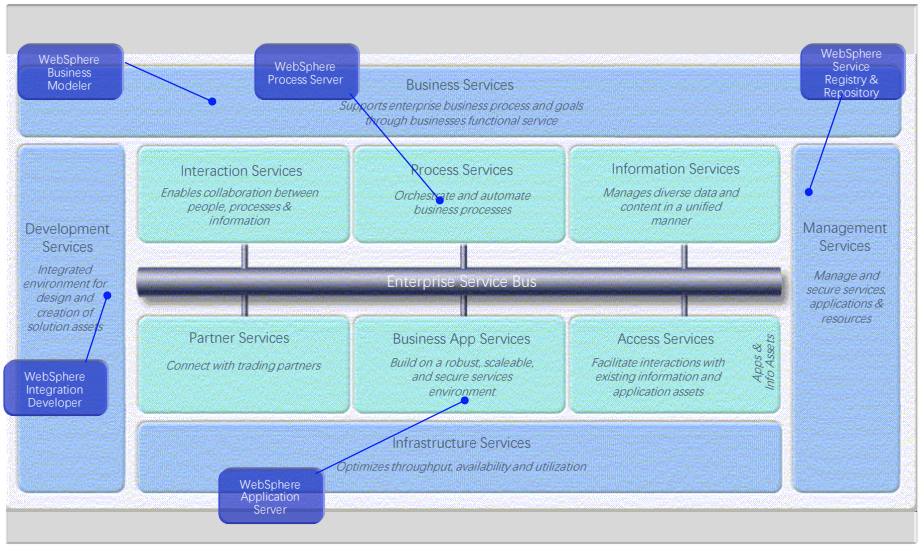


- The first major step of the SOMA identifies candidate services and enterprise components
- The second major step selects and specifies the services that will be exposed and enterprise components to realize them
- The third major step captures realization decisions

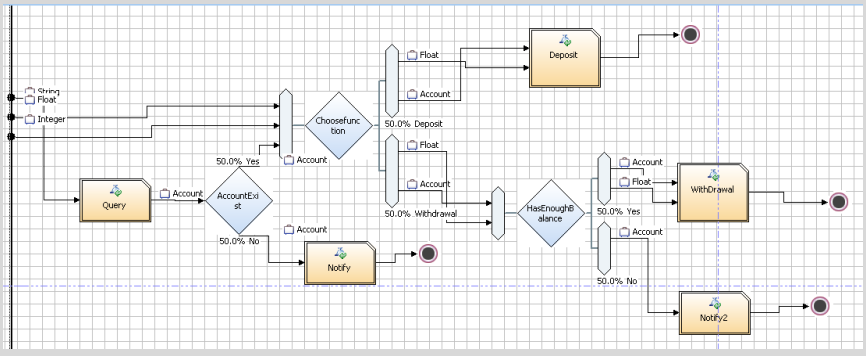
1.2.4 Reference Framework of SOA



1.3.1 SOA Products Supported by IBM

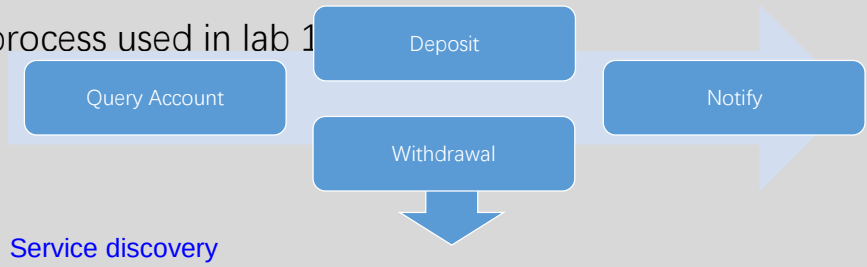


1.3.2 Case Study: Simple SOA Development



1.3.2 Case Study: Simple SOA Development

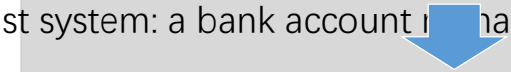
- Top-down method
- The process used in lab 1



- Service discovery
  - Query Account
  - Deposit
  - Withdrawal
  - Notify

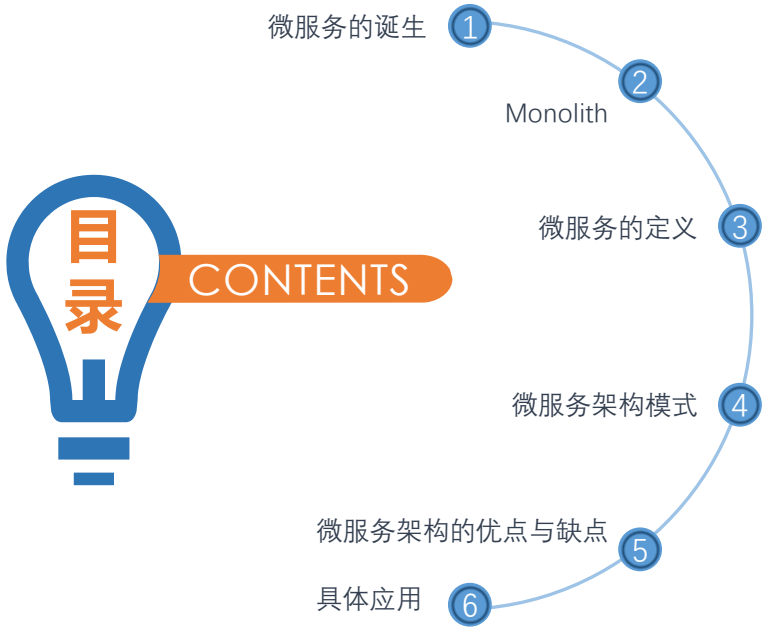
1.3.2 Case Study: Simple SOA Development

- Bottom-up method
- Exist system: a bank account management module



  - Service discovery
    - Query Account
  - Service identification
    - Account Query Service
    - Deposit Service
    - Withdrawal Service
    - Notify Service

微服务架构

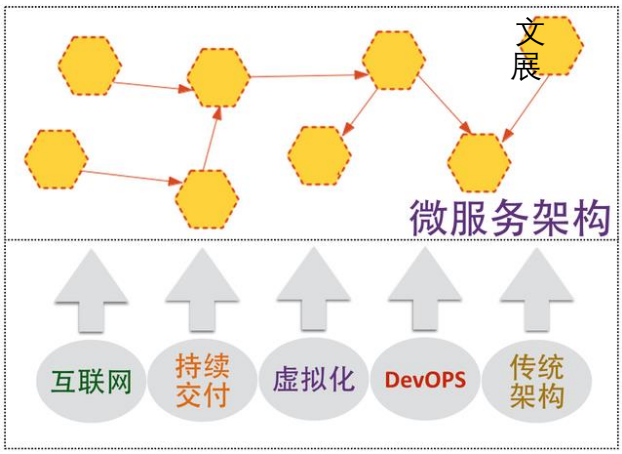

目录
CONTENTS
微服务的诞生 1
Monolith 2
微服务的定义 3
微服务架构模式 4
微服务架构的优点与缺点 5
具体应用 6

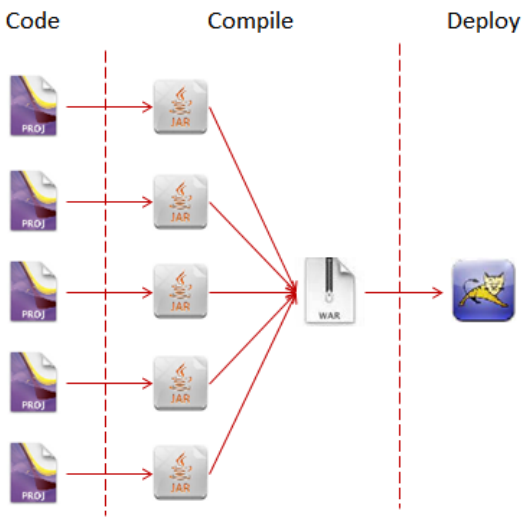

1
微服务的诞生

微服务架构（Microservice Architect）是一种架构模式，它提倡将单块架构的应用划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通。每个服务都围绕着具体业务进行构建，并且能够被独立的部署到生产环境、类生产环境等。

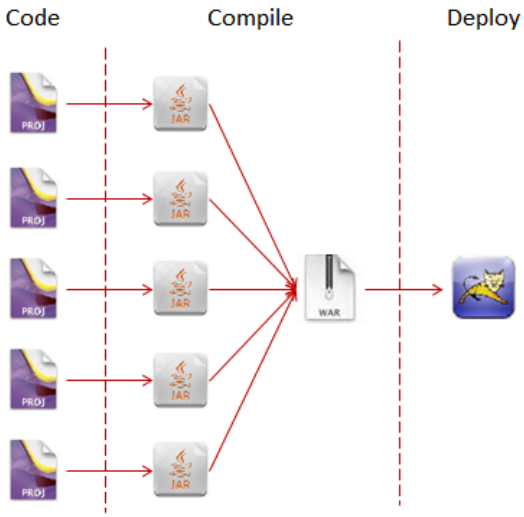
背景

微服务的诞生并非偶然。它是互联网高速发展，敏捷、精益、持续交付方法论的深入人心，虚拟化技术与DevOps化的快速发以及传统单块架构无法适应快速变化等多重因素的推动下所诞生的产物

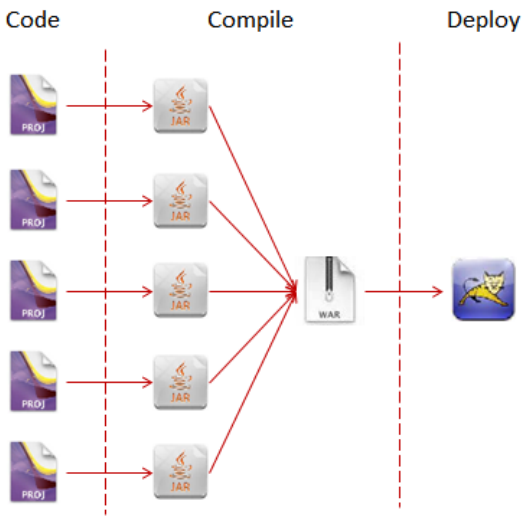




通常，要开发的服务所对应的代码由多个项目所组成，各个项目会根据自身所提供功能的不同具有一个明确的边界。



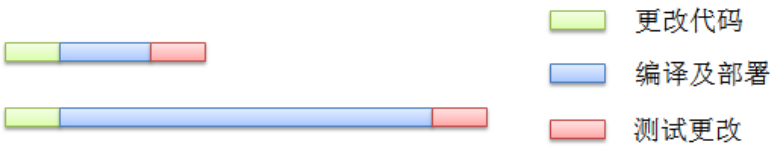
在编译时，这些项目将被打包成为一个个JAR包，并最终合并在一起形成一个WAR包。接下来，我们需要将该WAR包上传到Web容器中，解压该WAR包，并重新启动服务器。



在执行完这一系列操作之后，我们对服务的编译及部署就已经完成了

这种将所有的代码及功能都包含在一个WAR包中的项目组织方式被称为Monolith。在项目较小的情况下，这种代码组织方式还是可以接受的：更改完代码后，编译器编译代码，然后软件开发人员花费一分钟部署刚刚编译出来的WAR包以便测试自己刚刚所做的更改。但随着项目的逐渐变大，整个开发流程的时间也会变得很长：即使在仅仅更改了一行代码的情况下，软件开发人员需要花费几十分钟甚至超过一个小时的时间对所有代码进行编译，并接下来花费大量的时间重新部署刚刚生成的产品，以验证自己的更改是否正确。

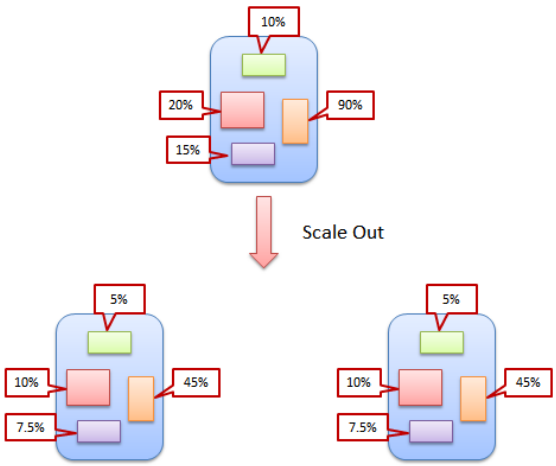
如果应用的部署非常麻烦，那么为了对自己的更改进行测试，软件开发人员还需要在部署前进行大量的环境设置，进而使得软件开发人员的工作变得繁杂而无趣



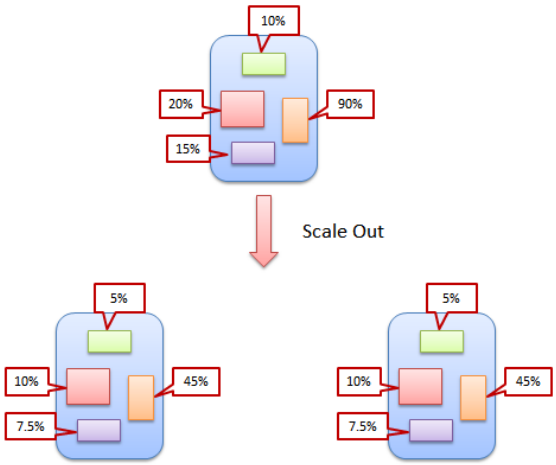
从上面的示意图中可以看到，在应用变大之后，软件开发人员花在编译及部署的时间明显增多，甚至超过了他对代码进行更改并测试的时间，效率已经变得十分低下。

在变得越来越大的同时，我们的应用所使用的技术也会变得越来越多。这些技术有些是不兼容的，就比如在一个项目中大范围地混合使用C++和Java几乎是不可能的事情。在这种情况下，我们就需要抛弃对某些不兼容技术的使用，而选择一种不是那么适合的技术来实现特定的功能。

除此之外，由于按照Monolith组织的代码将只产生一个包含了所有功能的WAR包，因此在对服务的容量进行扩展的时候，我们只能选择重复地部署这些WAR包来扩展服务能力，而不是仅仅扩展出现系统瓶颈的组成



但是这种扩展方式极大地浪费了资源。就以上图所展示的情况为例：在一个服务中，某个组成的负载已经达到了90%，也就是到了不得不对服务能力进行扩容的时候了。而同一服务的其它三个组成的负载还没有到其处理能力的20%。



由于Monolith服务中的各个组成是打包在同一个WAR包中的，因此通过添加一个额外的服务实例虽然可以将需要扩容的组成的负载降低到了45%，但是也使得其它各组成的利用率更为低下。

可以说，所有的不便都是由于Monolith服务中一个WAR包包含了该服务的所有功能所导致的。而解决该问题的方法就是Microservice架构模式。

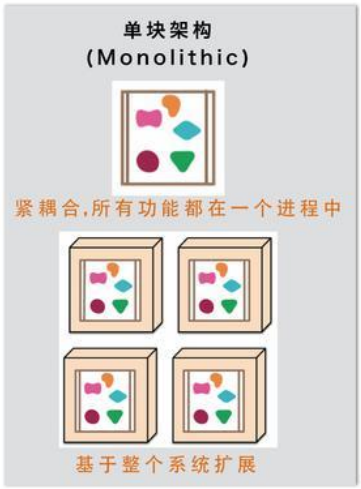
实际上，从业界的讨论来看,微服务本身并没有一个严格的定义。不过，ThoughtWorks的首席科学家，马丁 -福勒先生对微服务的这段描述，似乎更加具体、贴切，通俗易懂：

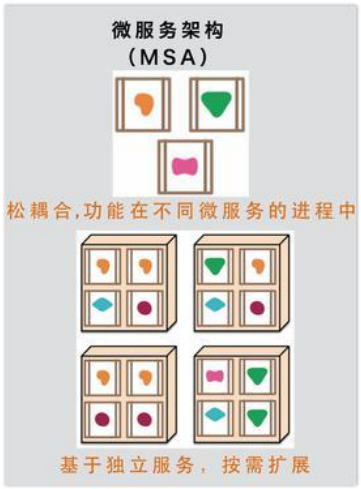
Microservice

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

微服务架构

微服务架构是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP协议的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够被独立的部署到生产环境、类生产环境等。另外，应当尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建。

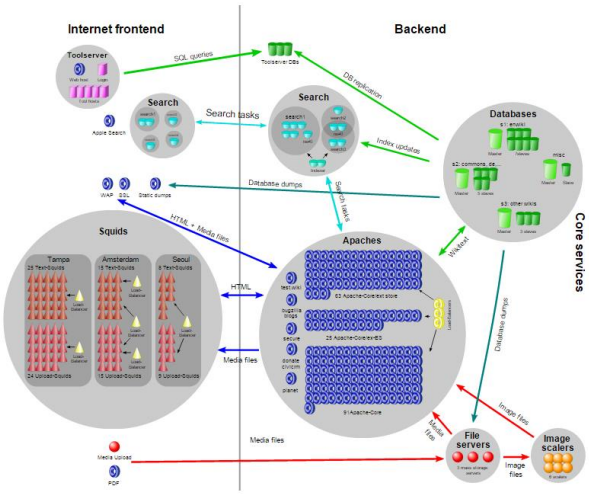




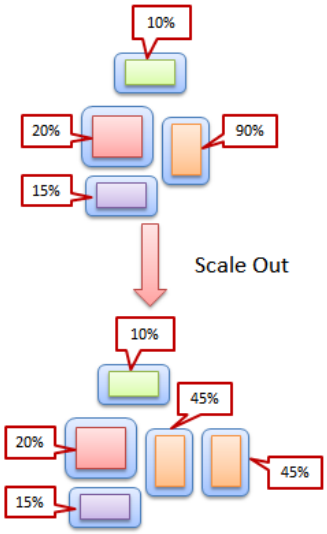
将功能分散到各个离散的服务中然后实现对方案的解耦。服务更原子，自治更小，然后高密度部署服务

简单地说，Microservice架构模式就是将整个Web应用组织为一系列小的Web服务。这些小的Web服务可以独立地编译及部署，并通过各自暴露的API接口相互通讯。它们彼此相互协作，作为一个整体为用户提供功能，却可以独立地进行扩容。

以下图所示的WikiPedia服务架构为例



WikiPedia包含了一系列服务，如数据访问服务Databases，搜索服务Search等。这些服务都包含了数量不等的服务实例，以确保能在不同负载的情况下为用户提供优质的服务。在用户的请求到达时，它们将协同工作，一起完成对用户请求的响应



在使用Microservice架构模式的情况下，软件开发人员可以通过编译并重新部署单个子服务的方式来验证自己的更改，而不再需要重新编译整个应用，从而节省了大量的时间。同时由于每个子服务是独立的，因此各个服务内部可以自行决定最为合适的实现技术，使得这些子服务的开发变得更为容易。最后如果当前系统的容量不够了，那么我们只需要找到成为系统瓶颈的子服务，并扩展该子服务的容量即可。

微服务架构的优点与缺点

优点

1. 每个服务足够内聚，足够小，代码容易理解、开发效率提高
2. 服务之间可以独立部署，微服务架构让持续部署成为可能
3. 每个服务可以各自进行扩展，而且，每个服务可以根据自己的需要部署到合适的硬件服务器上
4. 容易扩大开发团队，可以针对每个服务（service）组件开发团队
5. 提高容错性（fault isolation），一个服务的内存泄露并不会让整个系统瘫痪
6. 系统不会被长期限制在某个技术栈上

缺点

1. 开发人员要处理分布式系统的复杂性；
开发人员要设计服务之间的通信机制，对于需要多个后端服务的user case，要在没有分布式事务的情况下实现代码非常困难；涉及多个服务直接的自动化测试也具备相当的挑战性
2. 服务管理的复杂性，在生产环境中要管理多个不同的服务的实例，这意味着开发团队需要全局统筹（PS：现在docker的出现适合解决这个问题）
3. 应用微服务架构的时机如何把握？对于业务还没有理清楚、业务数据和处理能力还没有开始爆发式增长之前的创业公司，不需要考虑微服务架构模式，这时候最重要的是快速开发、快速部署、快速试错

目前国外使用微服务架构的知名厂商中不乏Amazon、Twitter、Netflix等这样的科技巨头，但是国内在微服务领域实践这块，真正成功的案例屈指可数，好雨云平台强调应用一键部署，整个平台的核心正是基于微服务的架构去搭建，可以说，好雨云在微服务领域有着成功的经验和技术

好雨云是基于“云帮”2.0及主流 IaaS 厂商资源搭建的、面向业务的新一代公有云服务，同时也是拥有“云帮”2.0全部特性的企业级云应用快速交付及管理平台。用好雨云，让云落地。

微服务在好雨云的解决方案

好雨微服务架构

好雨云平台的底层是通过Docker实现的，只是用户感受不到Docker。通过这种内部封装,用户不用管理计算资源和网络资源，把复杂技术包装在内部。通过服务整体对外

感谢