Component-based Software Development

Introduction

Dr. Vinod Dubey
SWE 645
George Mason University

Agenda

- **Icebreaker**
  - Introduction
- **Syllabus and class website**
- **Common Challenges in Software Industry**
  - How to solve them
- **Course Preview**
- **Questions/Answers**

Common Challenges in Software Industry

- **Monolithic Systems (in legacy environments)**
  - **All in one approach**
    - **Most capabilities of the application packaged and deployed together**
  - **Adds complexity**
    - **Tightly coupled**
    - **Hard to change**
    - **Difficult to scale**
  - **Longer testing life cycle**
    - **Requires regression testing of the entire application even for a small change**
  - **Slow/delayed software delivery process**

Good Software Characteristics

- Key characteristics of modern applications:
  - Meet functional requirements correctly!
  - Meet non-functional requirements:
    - Resiliency/ Fault Tolerance
    - Security
    - Modularity – Microservices
    - Portability - Containerized microservices
    - Event-driven processing
    - Automation – CI/CD
    - Scalability/Elasticity
    - Availability/Reliability
    - Interoperability

Good Software Characteristics

- Key characteristics of modern applications:
  - Meet functional requirements correctly!
  - Meet non-functional requirements:
    - Resiliency/ Fault Tolerance
    - Security
    - Modularity – Microservices
    - Portability - Containerized microservices
    - Event-driven processing
    - Automation – CI/CD
    - Scalability/Elasticity
    - Availability/Reliability
    - Interoperability
  - While these are all prerequisites to a finished product, not a single one is specific to any one business domain!

Good Software Characteristics

- **Not only should the developed software do its job correctly,** but it must **also** do it well!

- **Example: the healthcare.gov launch**
  - **Initial deployment met the functional requirements, such as**
    - **registering with healthcare.gov**
    - **choosing desired health insurance coverage**
  - **But failed to perform well when thousands of users tried logging in to sign up for the insurance!**

What is desired?

- **Adopt an efficient and effective software development approach**
- **Design and develop software so that they can be reused**
  - **Reuse can save design, coding, and testing cost**
- **Need to pay attention to both - functional and non-functional requirements**
- **Leverage component-based frameworks that are already developed and tested**
  - No need to re-invent the wheel!
- ***The goal**: to learn skills/approaches to build modular, portable, scalable, and resilient enterprise applications*

What is a Software Component?

- **A component is a way to modularize software in smaller packages**
- **You should be able to replace or upgrade it independently without impacting other parts of the application**
- **Basic features of software components**
  - Identity
  - Modularity
  - Contract-based interface
  - Independent deliverability
  - Reusability

SWE645 – Course Preview

SWE645 – Key Topics

- **Resilient Architectures (in AWS)**
- **DevOps-CI/CD/Microservices/Containers**
- **Container Orchestration (Kubernetes)**
- **Message Streaming Platform– Apache Kafka**
- **Angular/TypeScript**
- **Web Services (RESTful)**
- **Java Persistence API (JPA)/Hibernate**
- **Data Warehousing (Redshift)**
- **NoSQL Database (DynamoDB)**
- **Python Programming**
- **Serverless Computing(Lambda, SQS, SNS)**
- **AWS cloud platform for assignments**
  - Amazon S3, Amazon EC2, Amazon RDS
  - And tons of container-based technology

Resilient Architectures (in AWS)

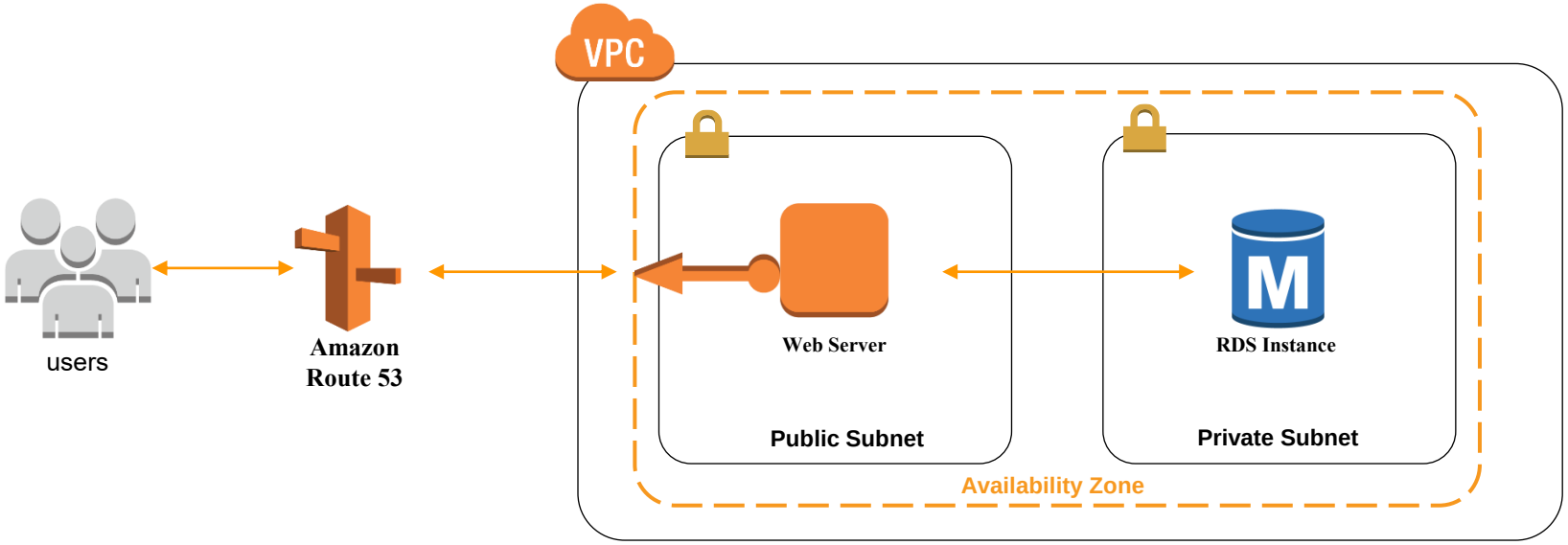
Resiliency

- **Ability of a system/service to automatically recover from failures**
  - Fault tolerant
  - Highly available
  - Scalable/performant
  - Secure

System Failure types

- **It's a long list!**
  - Hardware/Infrastructure Failures
    - Hardware failures/VM failures
    - Power outage/Availability Zones failure
  - Failures due to Performance Degradation
    - Application component failure
    - System crash due to excessive workload
    - Unacceptable Response time/throughput
  - Security Risks
    - Unauthorized Access (Human/system)
    - System Vulnerabilities /Compliance Issues
  - Data Security Issues
    - Data Integrity/Durability/Availability/Confidentiality
  - Network Connectivity Issues
  - Natural Disaster requiring Disaster Recovery
- **How do we design a system to withstand all these failures?**

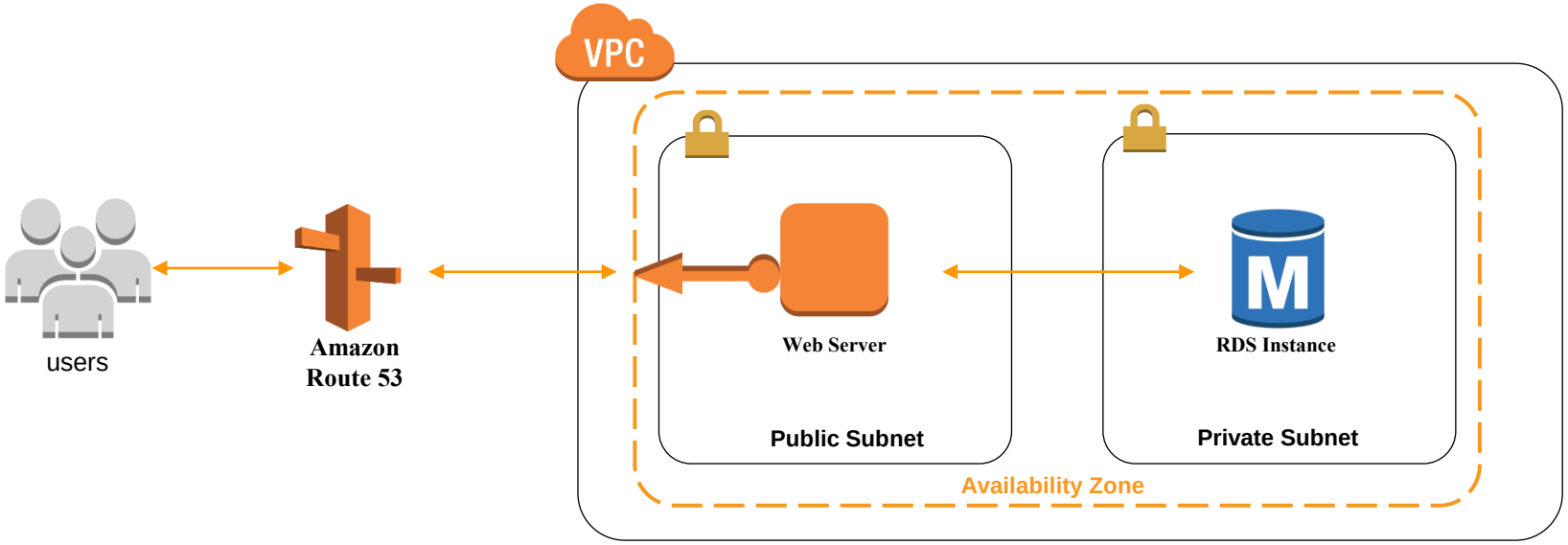
What's wrong with this architecture?



A typical multi-tier web application architecture in AWS

What's wrong with this architecture?

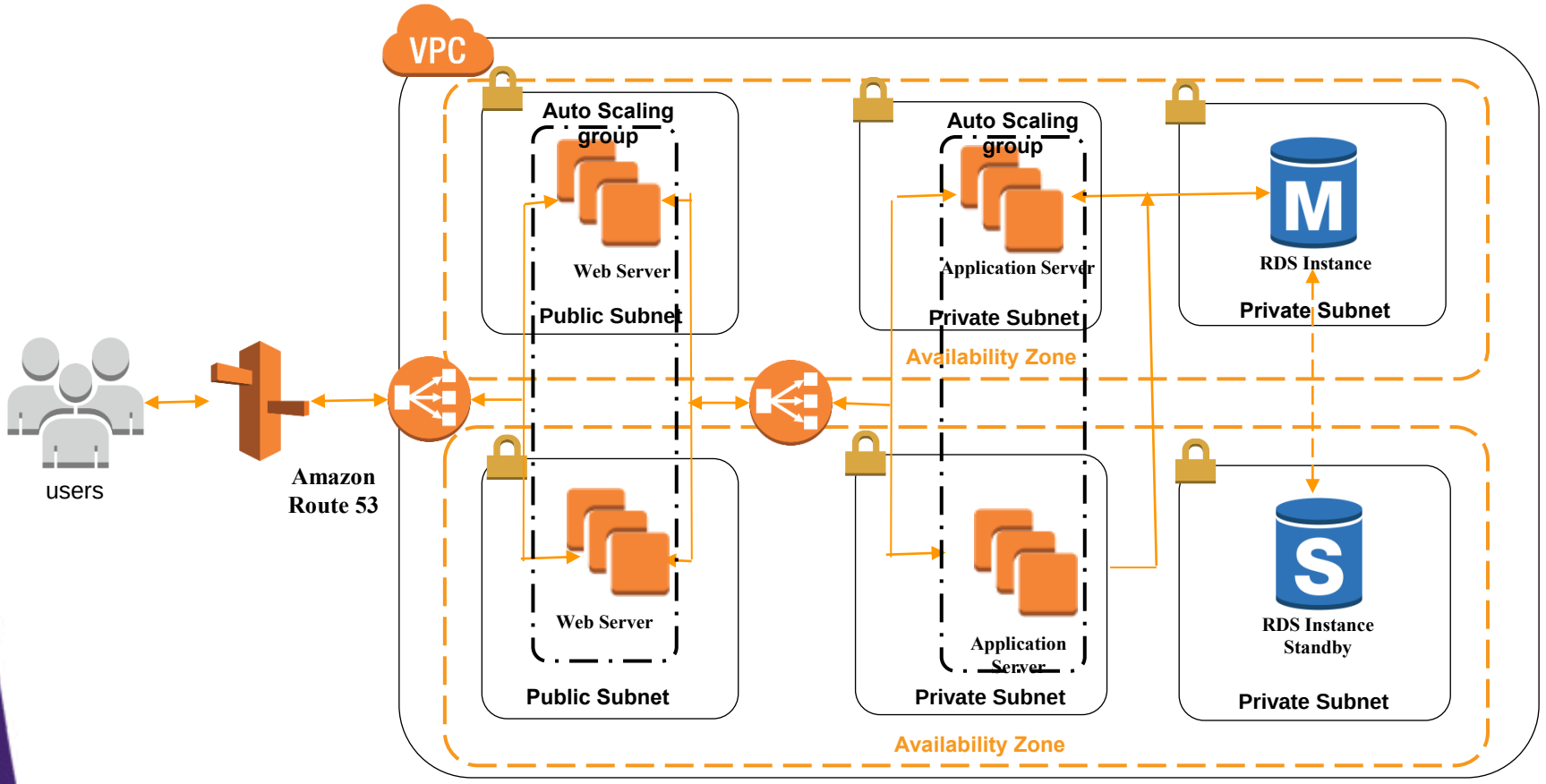
- **Multiple Single Point of Failures**
  - If the Database fails, the entire system fails
  - If the Web Server fails, the entire system fails
  - If the Availability Zone fails, the entire system fails



A typical multi-tier web application architecture in AWS

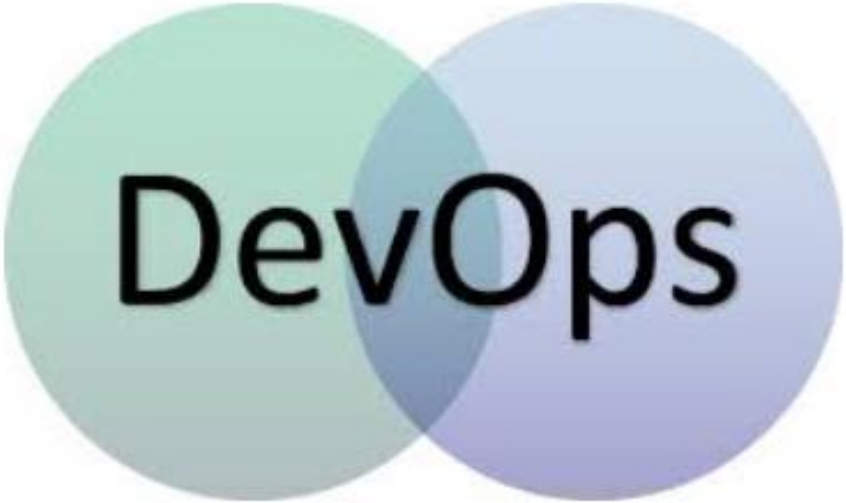
How do we design apps in the cloud?

- Need to build applications that are highly available, resilient/fault tolerant, scalable, and secure



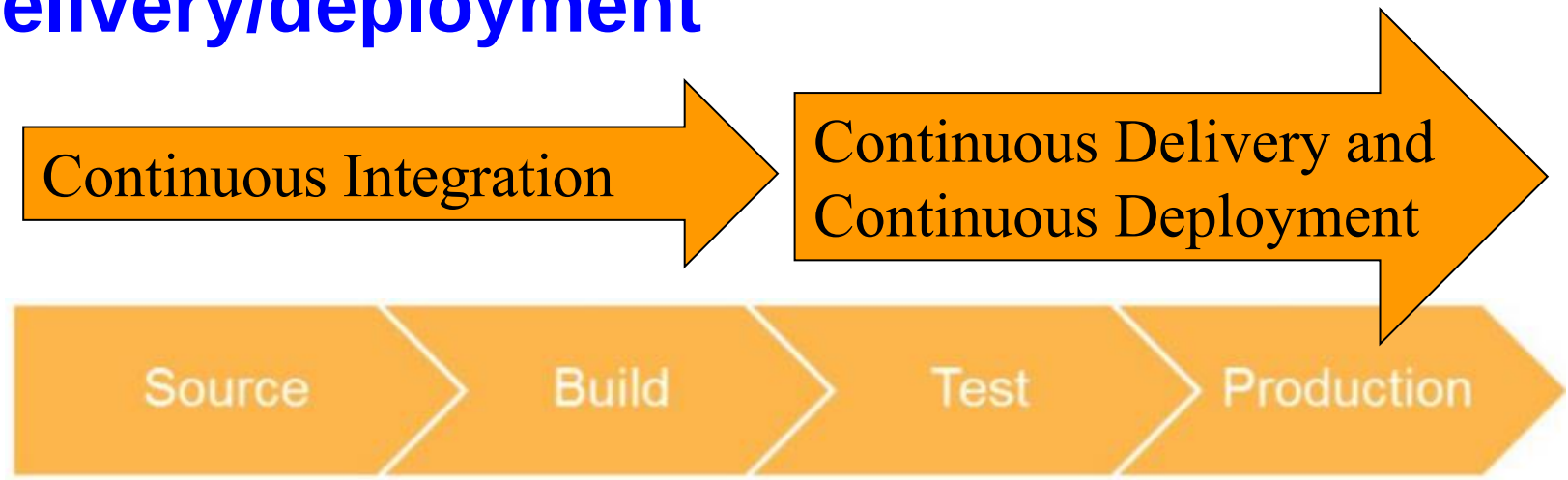
DevOps

- **Describes a culture and processes that bring development and operations teams together**
  - to complete software development.
- **Allows organizations to create and improve products at a faster pace using Agile methodology**



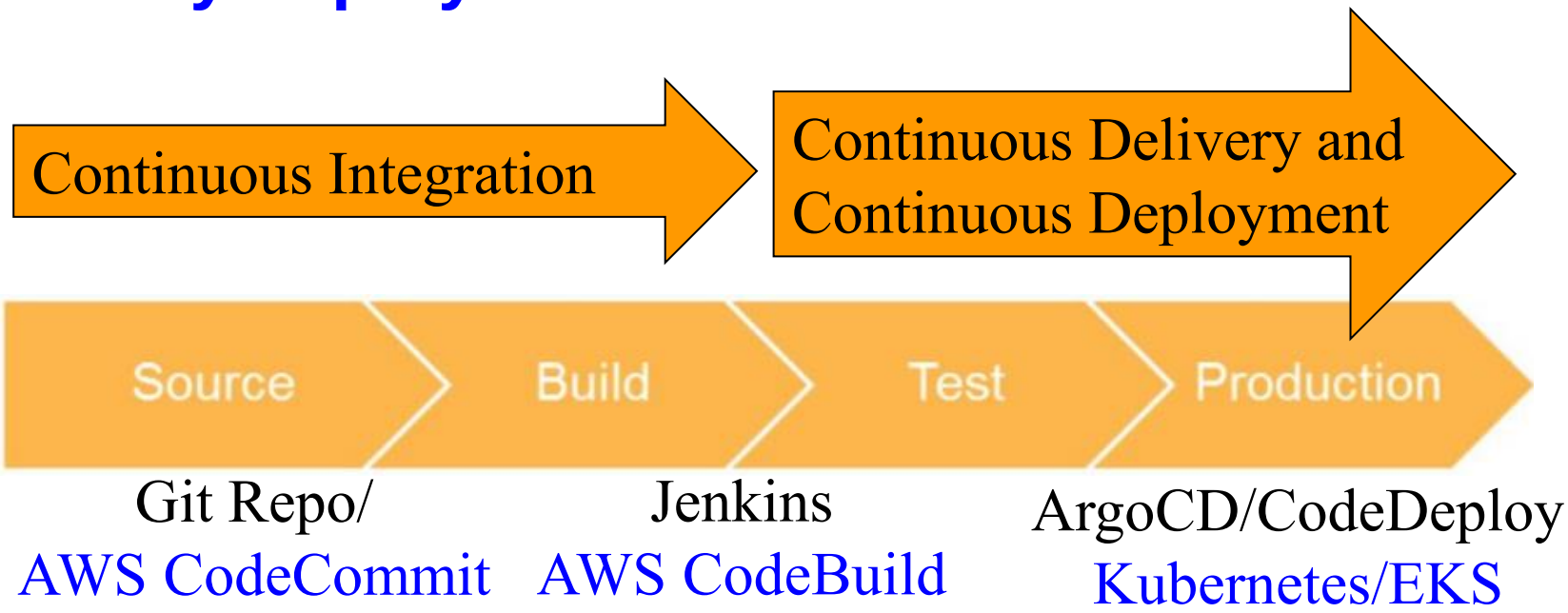
DevOps

- **Enhanced collaboration**
  - of Development and Operations teams
- **Automation**
  - of build, test, and deployment processes
- **Facilitates continuous integration, continuous delivery/deployment**



DevOps

- **Enhanced collaboration**
  - of Development and Operations teams
- **Automation**
  - of build, test, and deployment processes
- **Facilitates continuous integration, continuous delivery/deployment**



Microservices

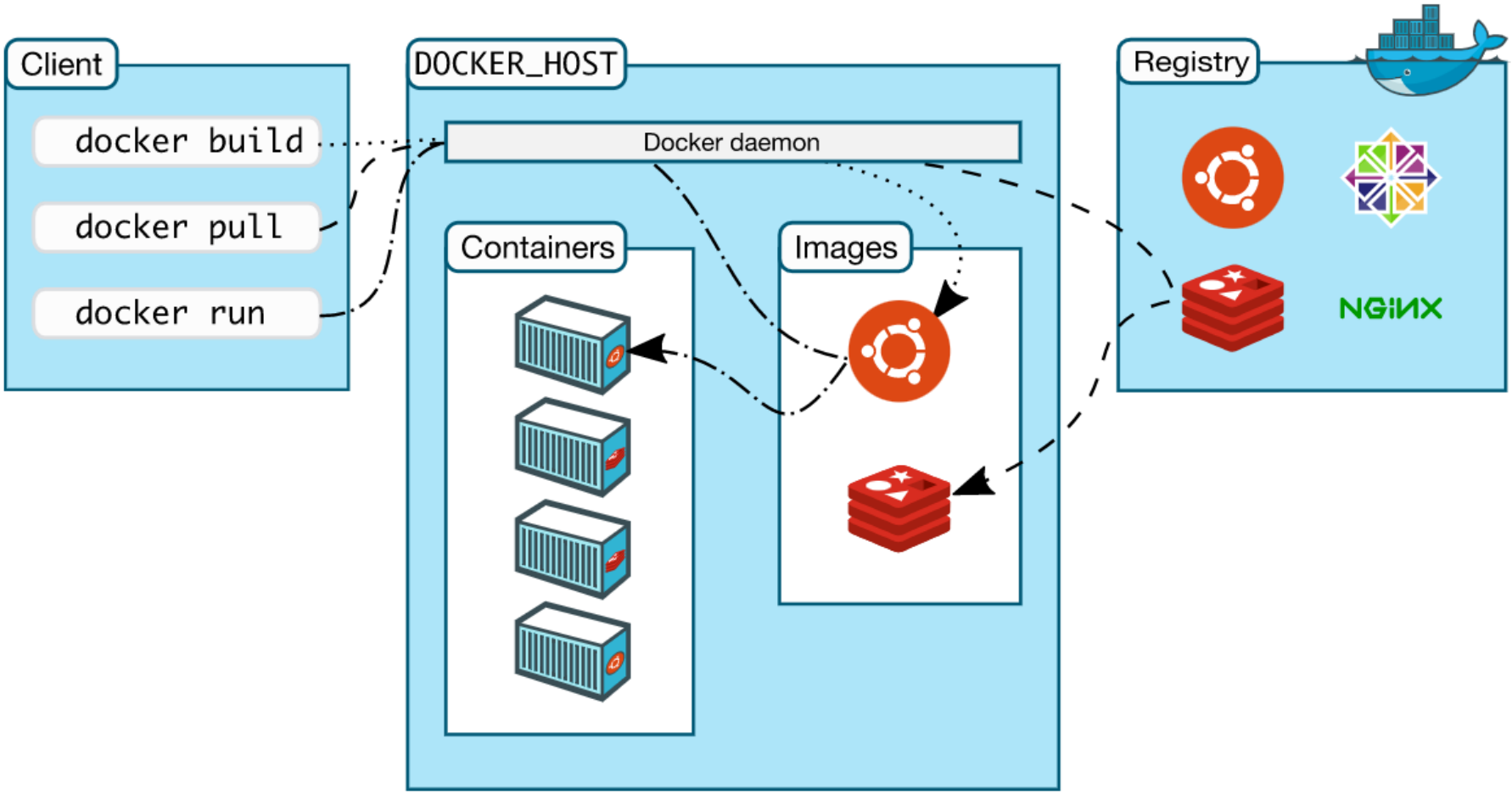
- **An architecture style that promotes developing modular applications that can be deployed independently and that can be scaled independently**
  - Service oriented – Communicate over API
  - Loosely coupled – Updating one service does not impact other services
  - Bounded Context – Do one thing and do it well
  - Modular – Domain/business capability driven design
- **Moves away from monolithic applications**
  - Key Drivers
    - Scalability challenges; Lack of efficiencies
    - Difficulties with adopting new technologies
    - Slow developer velocity

Container

- **Container technology is used to build containerized applications**
  - Allows packaging of an application and its dependencies in a way that can run consistently the same on any platform
- **Promotes portability and flexibility on where the application can run**
- **You can start with publicly accessible Base Images (e.g., for Nginx,Tomcat, MySQL)**
  - and add your applications
- **Very light weight – containers start and stop in seconds!**
- **Docker made containers popular**
  - "Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.

Container

- **Docker Architecture**



•https://docs.docker.com/get-started/overview

Container Orchestration – Kubernetes

Issue

- **Starting and stopping one container in a development environment on your laptop is easy!**
- **Managing a cluster of containerized applications in production is a hard problem**
- **We need to ask/consider**
  - How do we make containers resilient?
  - How do we achieve horizontal scalability across multiple servers?
  - How do we roll out a new version of my software without any service interruption?
  - How do we group related containers so that they run on the same host in order to work?

Issue

- **Starting and stopping one container in a development environment on your laptop is easy!**
- **Managing a cluster of containerized applications in production is a hard problem**
- **We need to ask consider**
  - How do we make containers resilient?
  - How do we achieve horizontal scalability across multiple servers?
  - How do we roll out a new version of my software without any service interruption?
  - How do we group related containers so that they run on the same host in order to work?
- **Answer: Container Orchestration**

What is Container Orchestration?

- **A general term for technologies that enable you to manage a large collection of containers easily**
  - Makes it easy to deploy containerized applications on a cluster and scale to meet workload demands
  - Automates container lifecycle
- **Key Responsibilities**
  - Provisioning and deployment of containers
  - Resiliency and availability of containers
  - Scalability/Elasticity and load balancing
  - Container access control (ingress/egress)
  - Allocation of container resources
  - Health and monitoring of containers and hosts

What is Container Orchestration?

- **Popular container orchestrators**
  - Kubernetes (**K8s**) by Google, now open sourced
  - Docker Swarm
    - The official orchestration platform by Docker
  - Google Kubernetes Engine (GKE)
    - runs Kubernetes under the hood
  - EC2 Container Service (ECS) by AWS
  - Elastic Kubernetes Service (EKS) by AWS
  - Azure Kubernetes Service (AKS)
  - Mesosphere DC/OS
  - Marathon

What is Kubernetes?

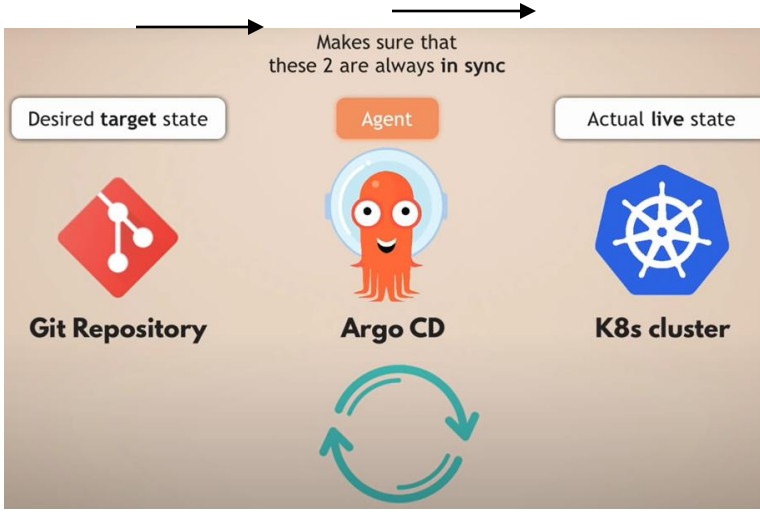
- **Kubernetes is an open-source container-orchestration system**
  - for automating deployment, scaling, and management of containerized applications
  - It was originally designed by Google
  - Supported by a vibrant and growing community of users and con**tri**butors
  - Kuber**netes** can run anywhere – on premises or in a cloud!

What is Kubernetes?

- **Kubernetes takes declared state of how you would like to configure your containerized application in YAML format and it makes that happen, even across computers.**
  - It will deploy your containers and make them **pu**blicly available.

Deploying on Kubernetes: Argo CD

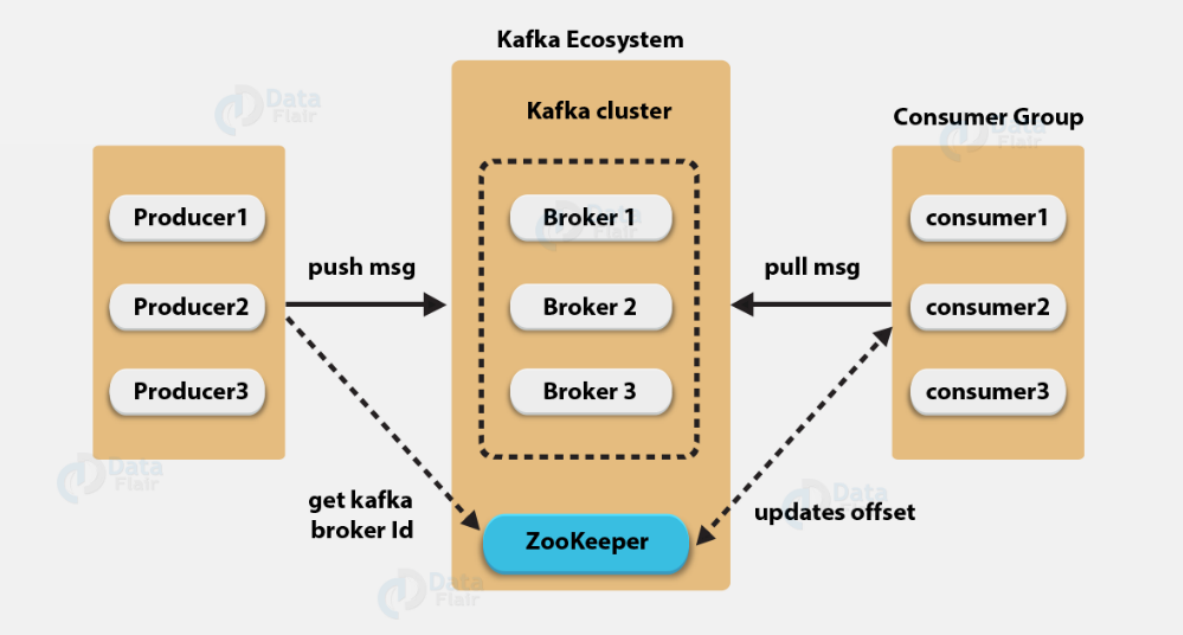
- **Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes**
- **Argo CD follows the GitOps pattern**
  - Uses Git repositories as the source of truth for defining the desired application state
  - Automates the deployment of the desired application states in the specified target environments.
  - Kubernetes manifests can be specified in several ways:
    - helm charts
    - YAML/json manifests
    - Kustomize applications
  - Application deployments can track updates to branches, tags, or pinned to a specific version of manifests at a Git commit.
  - *Source: https://argoproj.github.io/argo-cd/*



Apache Kafka

Apache Kafka

- **Apache Kafka is an open-source distributed message (event) streaming platform that enables data producers and consumers to exchange information efficiently**
  - A high-throughput, low-latency platform for handling real-time data feeds.
  - Written in Scala and Java.
  - Originated at LinkedIn



Source: Google/Data Flair

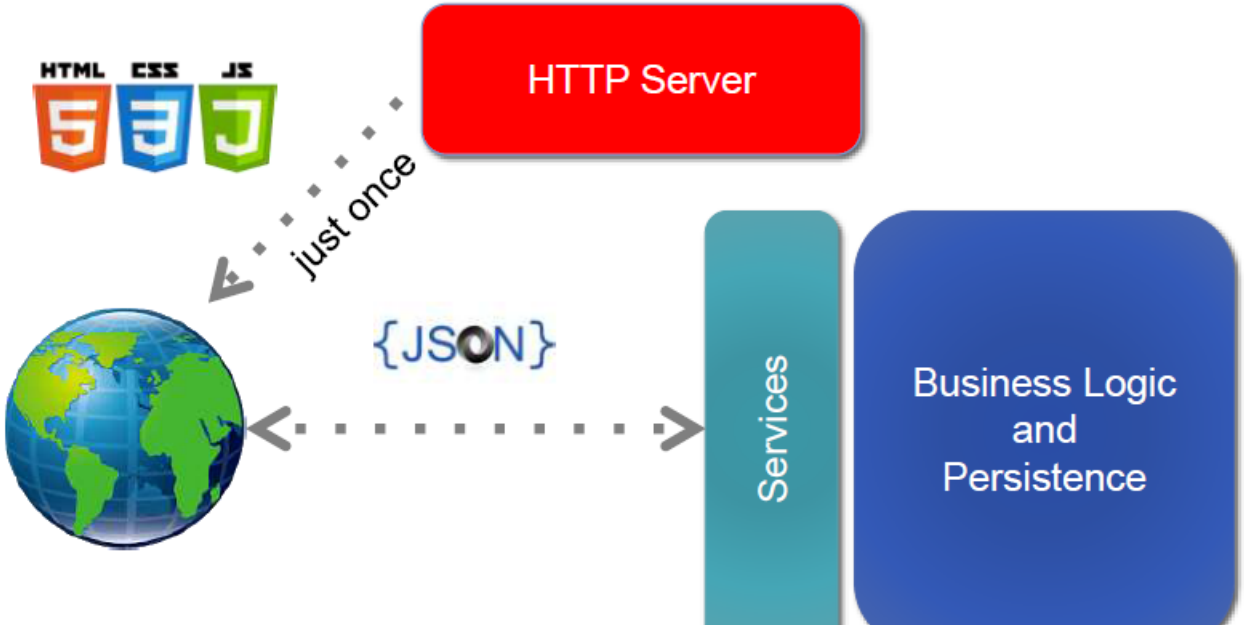
Angular

What is angular?

- **Angular is an open-source JavaScript-based Web application development framework**
  - Originally developed by Google
  - Uses a hierarchy of components as its primary architectural characteristic.
  - Uses Microsoft's TypeScript language
    - Static Typing, including Generics
    - Annotations and many more
    - TypeScript is a superset of JavaScript
  - Dynamic loading of content

What is angular?

- **Angular uses the concept of Single Page Application (SPA)**
  - Advantage of using SPA is that it results in an efficient and faster application
    - Eliminates the need to download html, JS, and CSS code in each request



Agenda for Angular

- Angular Project
- Angular component
- Data binding
- Directives
- Styles for components
- Creating/Using Modules
- Services/Dependency Injection
- HttpClient/Making RESTful calls
- Routes
- Forms
- TypeScripts

Java Persistance API (JPA) / Hibernate

Data Persistence

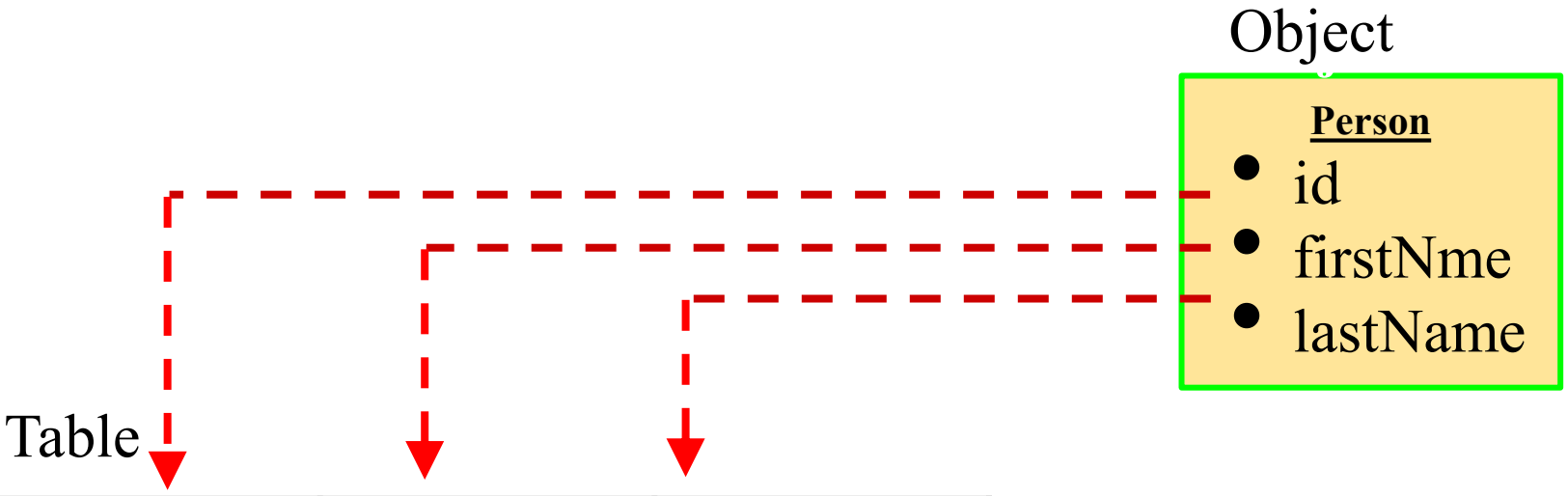
- How do we preserve application data into a database?

Data Persistence

- **How do we preserve application data into a database?**
- **Two key options on Java platform**
  - JDBC/SQL
  - Java Persistence API (JPA)
    - Hibernate - Object/Relational Mapping (ORM)

Java Perisistence API (JPA)

- **JPA utilizes the O/R mapping design**



PERSON_ID	FIRST_NAME	LAST_NAME
1	Greg	Martin
2	John	Doe
3	George	Smith

A single JavaBean can be mapped to a database record. This type of design allows for data manipulation using Objects.

Goal

The Goal is to learn:

- **How to use JPA annotations to configure POJOs into JPA entities to provide O/R mapping**

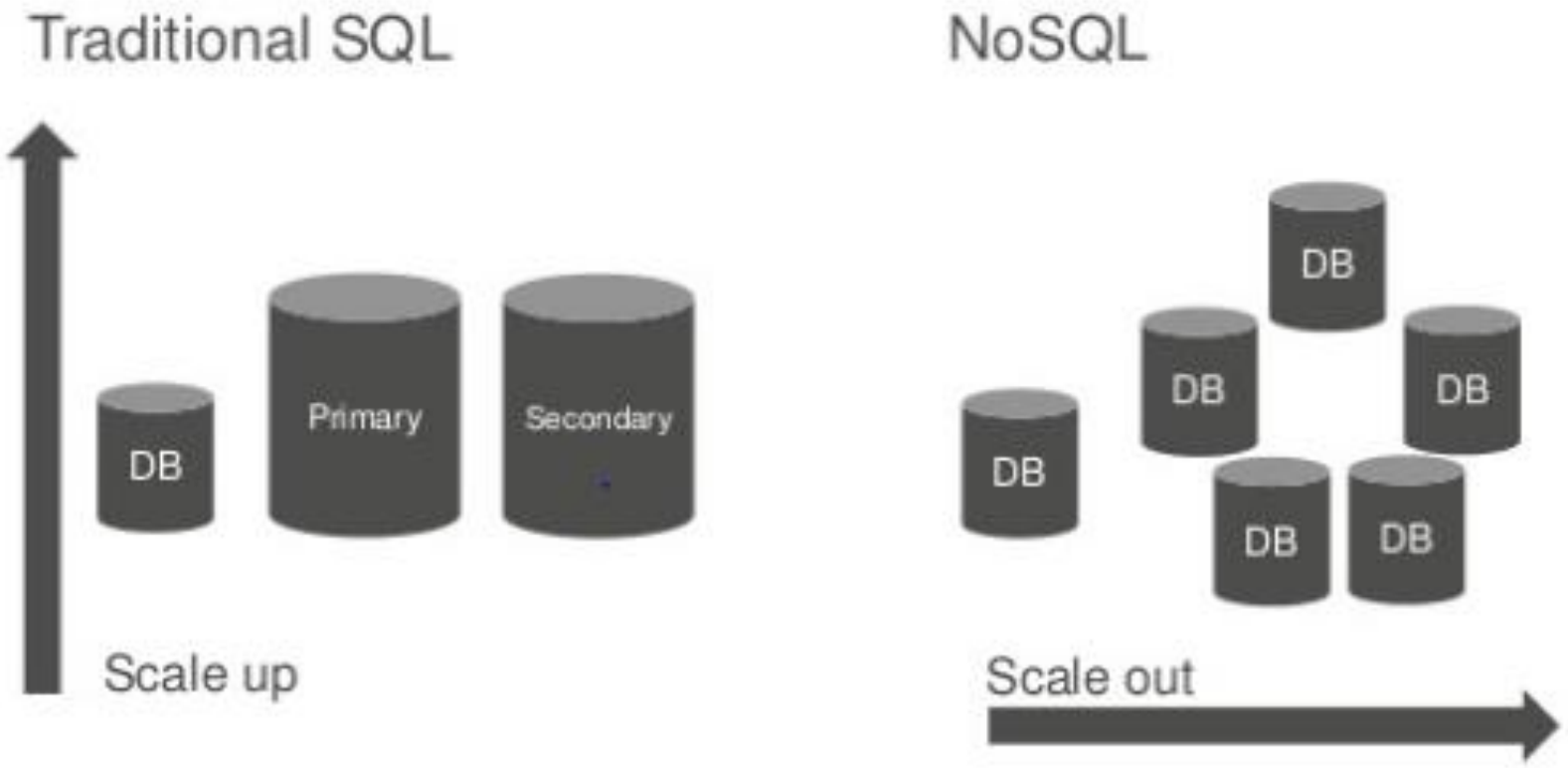
- **How to use Persistence APIs (EntityManager) to implement CRUD operations**

JPA Providers

- **Hibernate**
- **EclipseLink -- reference implementation**
- **Apache OpenJPA**

NoSQL Database / Amazon DynamoDB

Relational (SQL) vs. Non-Relational (NoSQL)



NoSQL Solutions

- **Popular NoSQL Databases**
  - MongoDB
  - Cassandra
  - MarkLogic
  - Couchbase
  - DynamoDB

Amazon DynamoDB

- **Amazon DynamoDB is a fully managed NoSQL database service**
  - for applications that need consistent, single-digit millisecond latency at any scale.
- **Key Characteristics:**

Data Warehousing / Amazon Redshift

What is a Data Warehouse?

- **A large store of data accumulated from a wide range of sources within a company**
- **Designed for query and analysis**
  - rather than for transaction processing
- **Used for guiding decision making by management**
- **It usually contains historical data**
  - derived from transactional data
- **A data warehouse, typically, is a relational database**

Source: http://www.ibmbigdatahub.com/blog/ensuring-data-warehouse-quality-business-mandate

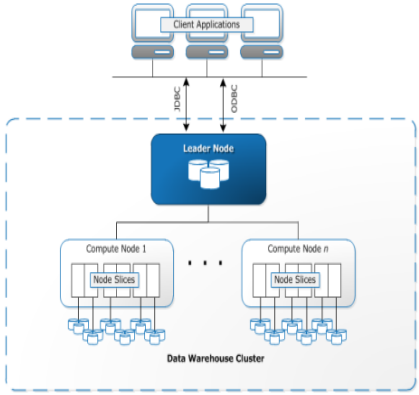
Amazon Redshift

- **Amazon Redshift is Amazon's relational data warehouse service in AWS**
- **Fully Managed**
  - Most administrative tasks are automated
    - Provisioning, configuration, backups, and patching
  - Continuous monitoring and recovery from failures
- **Petabyte Scale**
  - Optimized for datasets ranging from a few hundred gigabytes to a **p**etabyte or more
  - 1PB = 1000 terabytes = 1000000 gigabytes
    - Think the size of 10 billion photos on FACEBOOK

Amazon Redshift

Characteristics that make Redshift unique and high performing

- **Variety of innovations to reduce disk I/O**
  - Columnar Storage
  - Data Compression
  - Zone Maps
  - Data Sorting
    - Sort Keys
  - Node Slices
    - Support for massive parallelism
  - Data Distribution
    - Dist Key



Web Services

Question

- **How do we communicate and interoperate between disparate platforms, such as .NET, Java, Python, mainframe?**

Web Service

- **Web services provide a mechanism to remotely execute business operations using standard based technologies and protocols such as XML/JSON and HTTP**
  - A software component stored on one computer (Server)
  - Accessed by an application (or other software component) on another computer (Client) over a network.
- **In Java, Web services are implemented using classes**
  - The class that represents the web service resides on a Web server—it's not part of the client application.

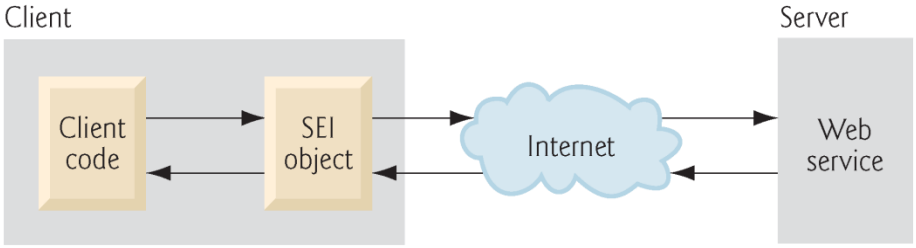


Fig. 31.7 | Interaction between a web service client and a web service.

Web Service Advantages

- **Language neutrality**
  - A client does not know the language used by the service; the service does not know the language used by the client
  - Web Services define the message format, not the **pro**gramming language used
- **Interoperability**
  - Definition of request and response data lets any Web service interact with any other web service.
- **Industry support**
  - Very widely adopted. No danger of investing in short-lived fad ala Ada and CORBA.
  - Very simple tools for Java, Python, Microsoft languages, etc.

Two Prevailing Styles of Web Services

- **RESTful Web Services**
  - Web Services that are based on REST architecture
- **SOAP Web Services**
  - Web Services that use SOAP to exchange XML formatted data
  - In this course we will focus on RESTful Web Services!

Cloud Computing Platform

Cloud Computing Platform

- **Leveraging Cloud Computing Platform**
  - Reduces the administrative burden of provisioning and maintaining the infrastructure and servers
    - Shared responsibility
  - Allows teams to focus on building and delivering capabilities
  - Managed services that yield speed and agility
  - Pay as you go pricing model
  - Define **Infra**structure-as-code

Amazon AWS

- Amazon Web Services (AWS) provides on-demand cloud computing platforms on a paid subscription basis
  - Amazon Web Services is a subsidiary of Amazon.com
- Offers compute power, storage, website hosting and many other functionalities to help businesses scale and grow

Source: https://en.wikipedia.org/wiki/Amazon_Web_Services

Amazon AWS

- **Key AWS cloud services that are used and discussed in this class**
  - Elastic Compute Cloud (EC2)
  - Simple Storage Service (S3)
  - Amazon Relational Database Service (RDS)
  - Amazon DynamoDB
  - Amazon Redshift
  - AWS Lambda
- **Well Architected Frameworks**
  - Availabilty / Reliability / Security
  - Resiliency / Fault Tolerance / Performance

Serverless Computing

Serverless Computing

- **A cloud-computing execution model in which the cloud provider provisions, manages, and runs the server, and dynamically manages the allocation of machine resources.**
  - Pricing is based on the actual amount of resources consumed by an application,
    - rather than on pre-purchased units of capacity.
    - It's a true pay-as-you-go model
  - It could be considered a form of utility computing.

Source: https://en.wikipedia.org/wiki/Serverless_computing

Serverless Computing

- **The term "serverless" is a misnomer!**
  - The cloud providers still use servers, but customers have no role in provisioning and maintaining them.
- **Helps customers focus on their core businesses**
  - Leave the worries of maintaining and patching servers to the cloud provider.

Other topic(s) to be covered

- **Python programming**
- **Detailed discussions of each of the topics introduced before**

Questions?