Syllabus -> Where do we stand?

- ✓ Introduction to Cloud Computing, Cloud Concepts & Technologies, Cloud Services & Platforms
- ➢ Hadoop & MapReduce - Concepts, Cloud Application Design
- ✓ Python for Cloud, Cloud Application Development in Python
- ➢ Big Data Analytics, Multimedia Cloud, Cloud Security, Cloud Application Benchmarking & Tuning

CS351 - Cloud Computing

Lecture #12

Cloud Application Design

Dr. Ferdous Ahmed Barbhuiya
Indian Institute of Information Technology Guwahati

Outline

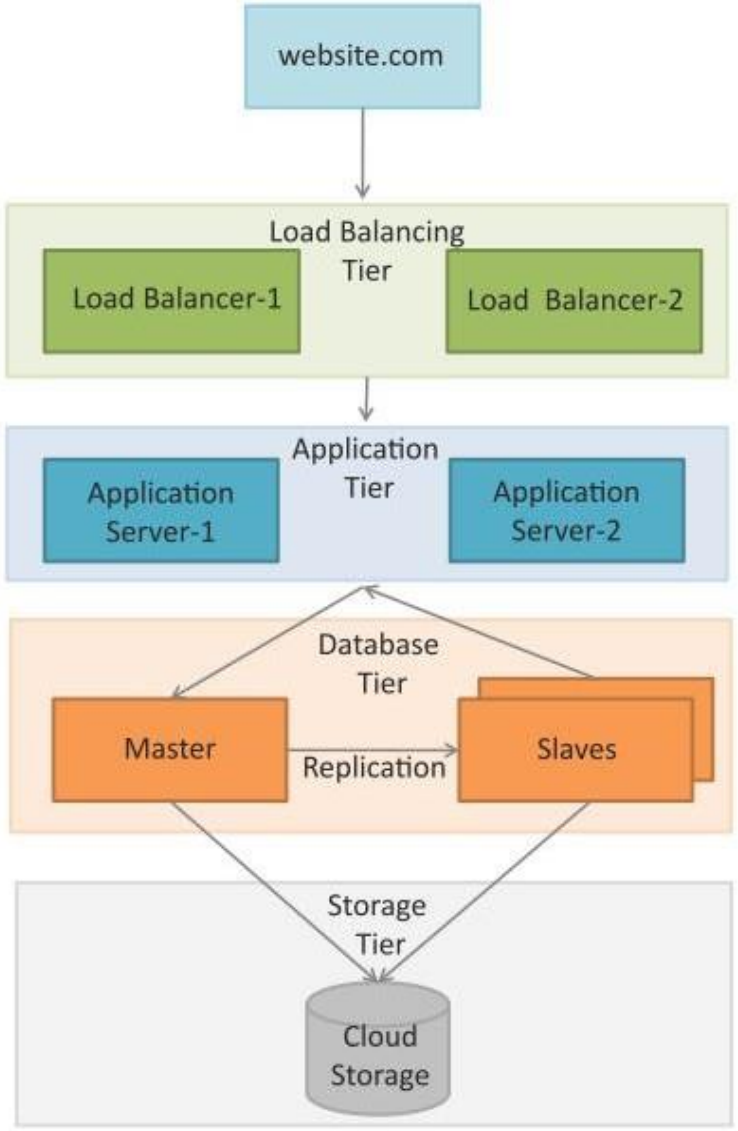
- Cloud Application Design Considerations
- Cloud Application Reference Architectures
- Design Methodologies
- Data Storage
- Data Analytics
- Deployment & Management

Design Considerations for Cloud Applications

- Scalability
  - Drives the application designers to move to cloud computing environments.
- Reliability & Availability
  - Ddefined as the probability that a system will perform the intended functions under stated conditions for a specified amount of time.
- Security
  - Security is an important design consideration for cloud applications given the outsourced nature of cloud computing environments.
- Maintenance & Upgradation
  - Design applications with low maintenance and upgradation costs.
- Performance
  - Applications should be designed while keeping the performance requirements in mind.

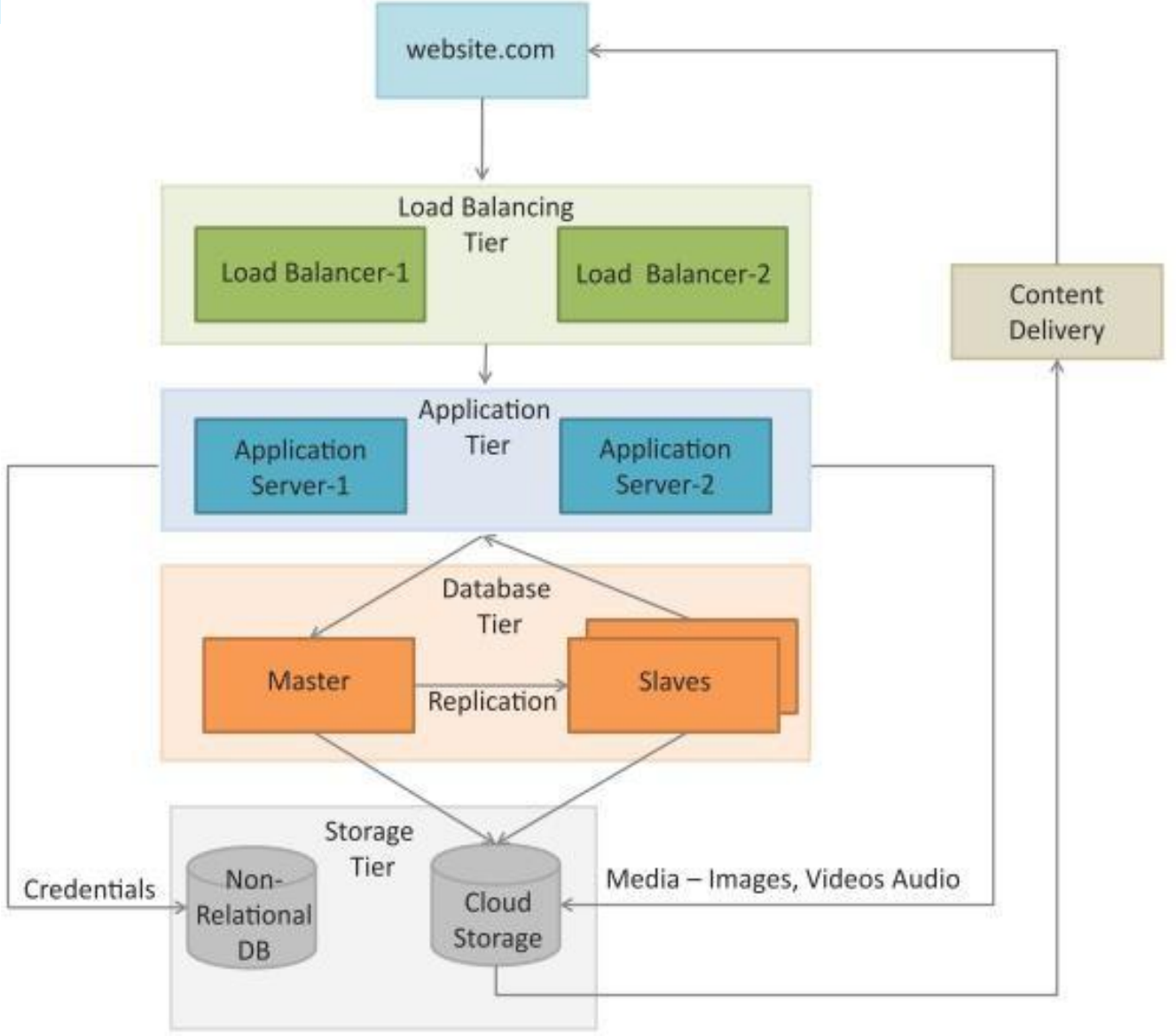
Reference Architectures – e-Commerce, B2B, Banking and Financial apps

- Load Balancing Tier
  - Load balancing tier consists of one or more load balancers.
- Application Tier
  - For this tier, it is recommended to configure auto scaling.
  - Auto scaling can be triggered when the recorded values for any of the specified metrics such as CPU usage, memory usage, etc. goes above defined thresholds.
- Database Tier
  - The database tier includes a master database instance and multiple slave instances.
  - The master node serves all the write requests and the read requests are served from the slave nodes.
  - This improves the throughput for the database tier since most applications have a higher number of read requests than write requests.



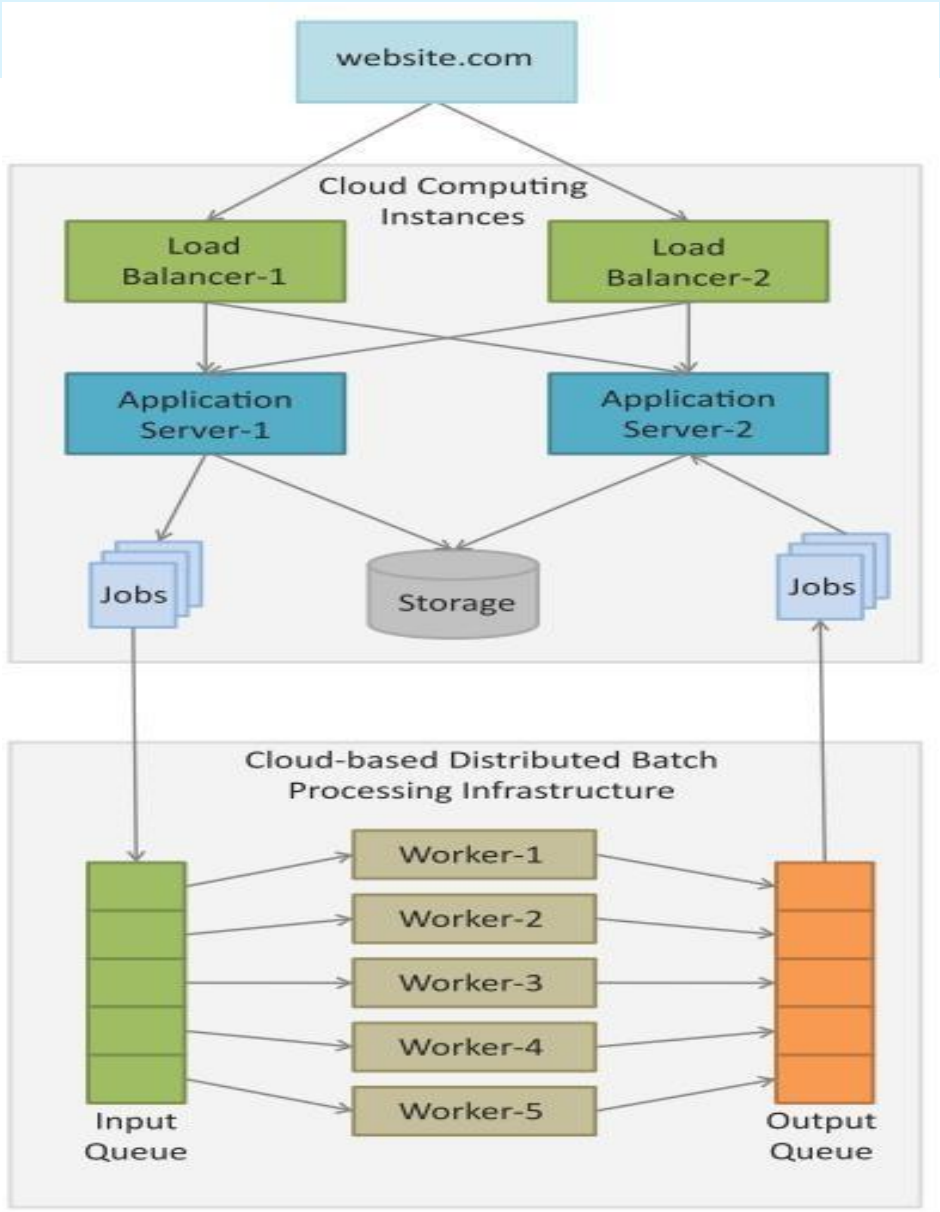
Reference Architectures –
Content delivery apps

- Figure shows a typical deployment architecture for content delivery applications such as online photo albums, video webcasting, etc.
- Both relational and non-relational data stores are shown in this deployment.
- A content delivery network (CDN) which consists of a global network of edge locations is used for media delivery.
- CDN is used to speed up the delivery of static content such as images and videos.



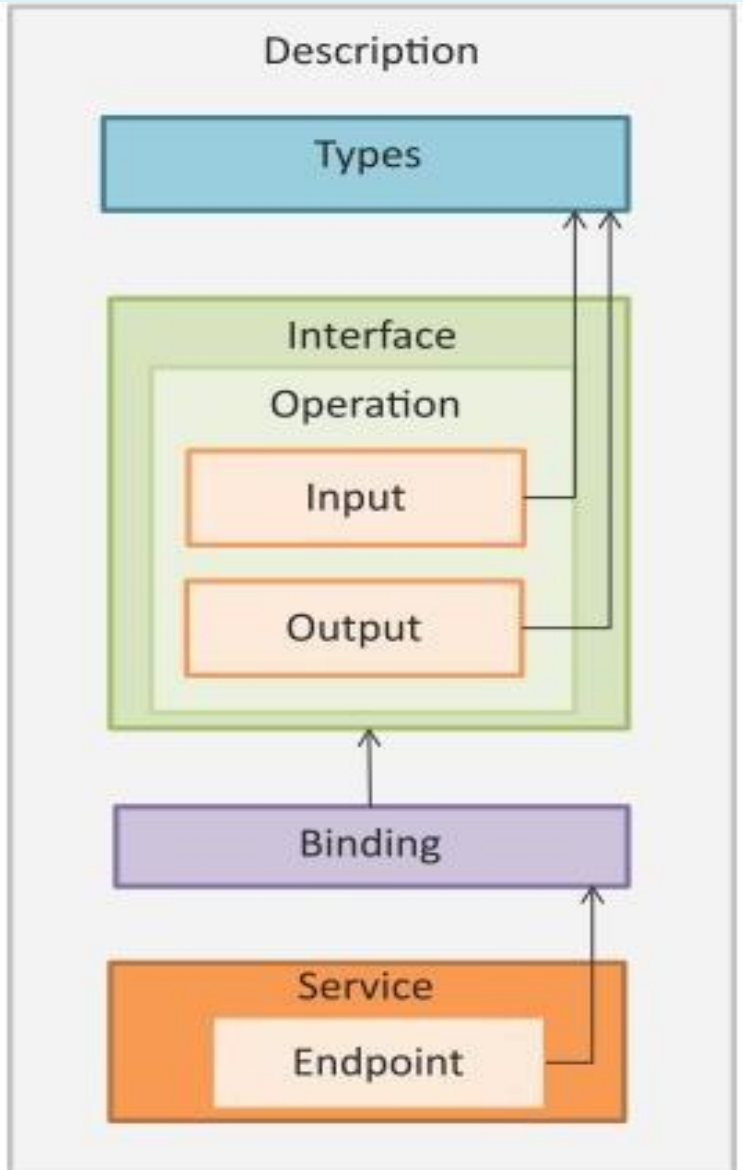
Reference Architectures –
Analytics apps

- Comprises of web, application, storage, computing/analytics and database tiers.
- The analytics tier consists of cloud-based distributed batch processing frameworks such as Hadoop which are suitable for analyzing big data.
- Data analysis jobs (such as MapReduce) jobs are submitted to the analytics tier from the application servers.
- The jobs are queued for execution and upon completion the analyzed data is presented from the application servers.



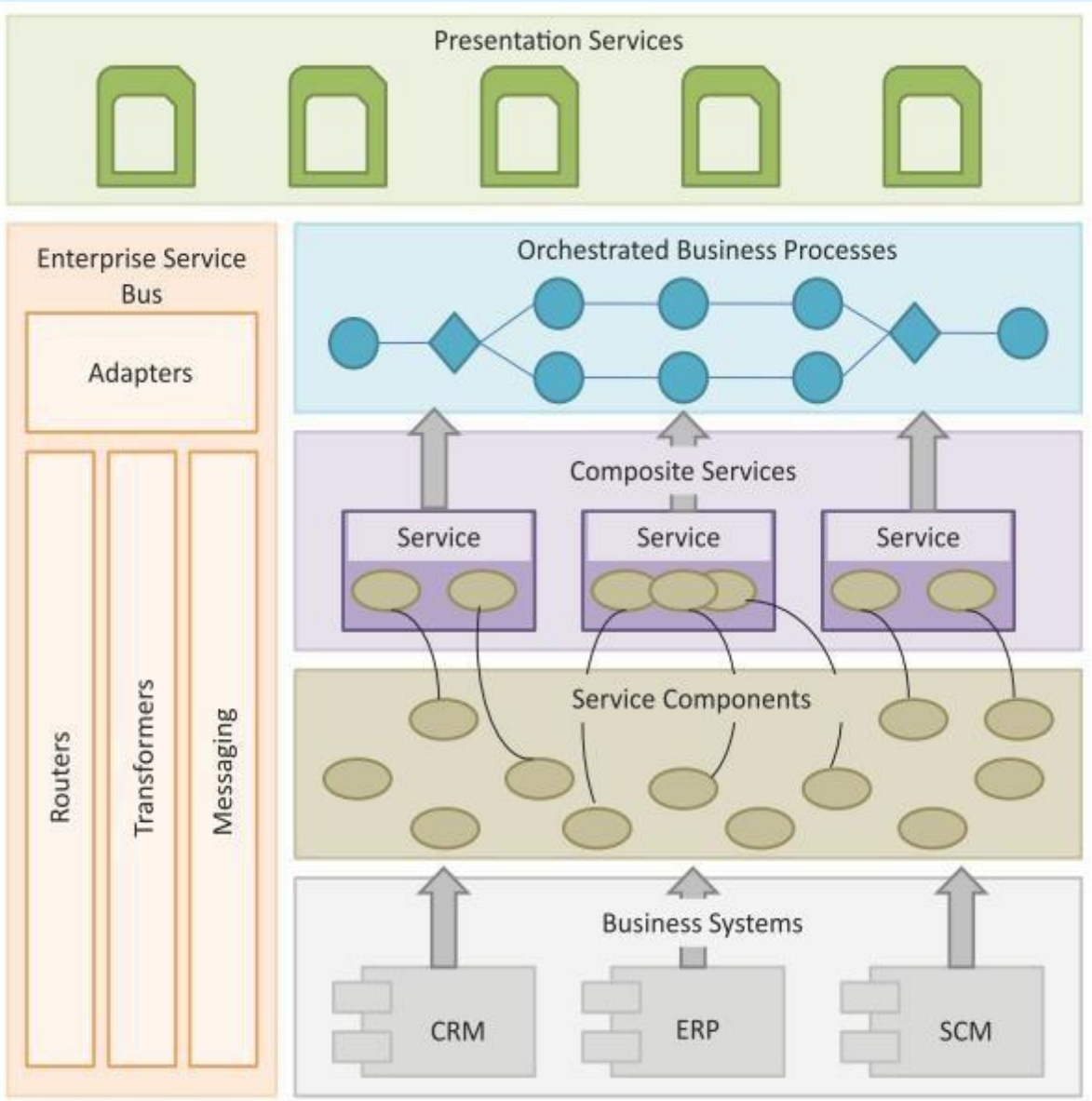
Service Oriented Architecture

- Approach for designing and developing applications in the form services that can be shared and reused.
- Collection of discrete software modules or services that form a part of an application and collectively provide the functionality of an application.
- SOA services are developed as loosely coupled modules with no hard- wired calls embedded in the services.
- Communicate with each other by passing messages.
- Services are described using the Web Services Description Language (WSDL).
  - WSDL is an XML-based web services description language, used to create service descriptions containing information on the inputs, outputs & functions performed by a service, of the service.



SOA Layers

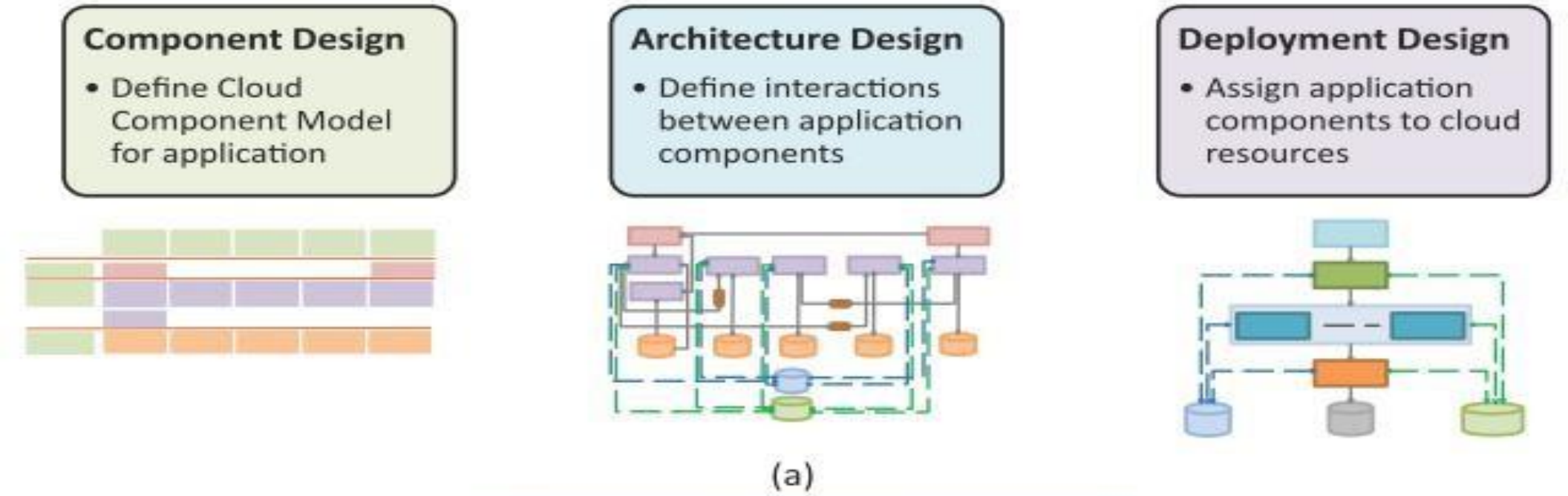
- Business Systems
  Custom built applications and legacy systems like ERP, CRM, SCM etc.
- Service Components
  Allow the layers above to interact with the business systems
  Responsible for realizing the functionality of the services .
- Composite Services
  Coarse-grained services composed of two or more service components, can be used to create enterprise scale components or business-unit specific components.
- Orchestrated Business Processes
  Composite services can be orchestrated to create higher level business processes. In this layers the compositions and orchestrations of the composite services are defined to create business processes.
- Presentation Services
  This is the topmost layer that includes user interfaces that exposes the services and the orchestrated business processes to the users.
- Enterprise Service Bus
  This layer integrates the services through adapters, routing, transformation and messaging mechanisms.



Cloud Component Model

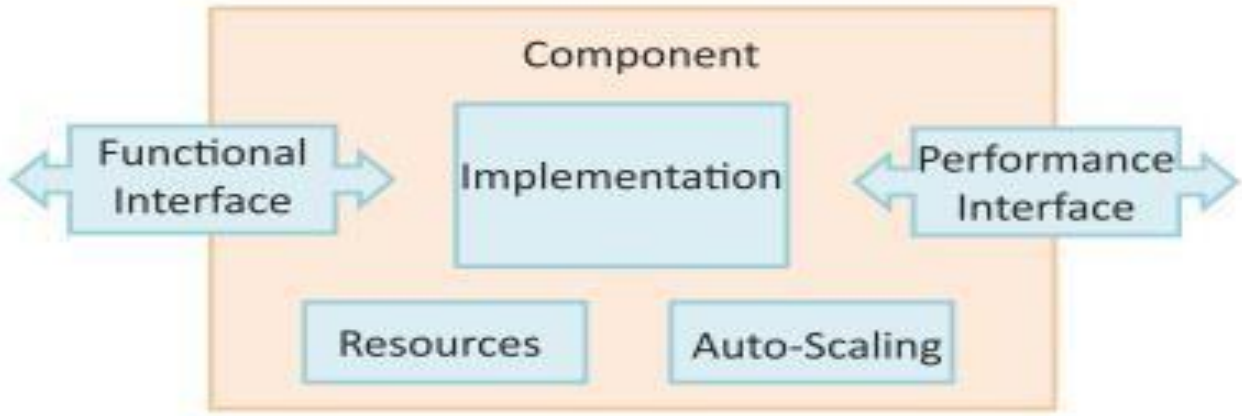
- Cloud Component Model is an application design methodology that provides a flexible way of creating cloud applications in a rapid, convenient and platform independent manner.
- CCM is an architectural approach for cloud applications that is not tied to any specific programming language or cloud platform.
- Cloud applications designed with CCM approach can have innovative hybrid deployments in which different components of an application can be deployed on cloud infrastructure and platforms of different cloud vendors.
- Applications designed using CCM have better portability and interoperability.
- CCM based applications have better scalability by decoupling application components and providing asynchronous communication mechanisms.

CCM Application Design Methodology



(a)

- CCM approach for application design :
  - Component Design
  - Architecture Design
  - Deployment Design

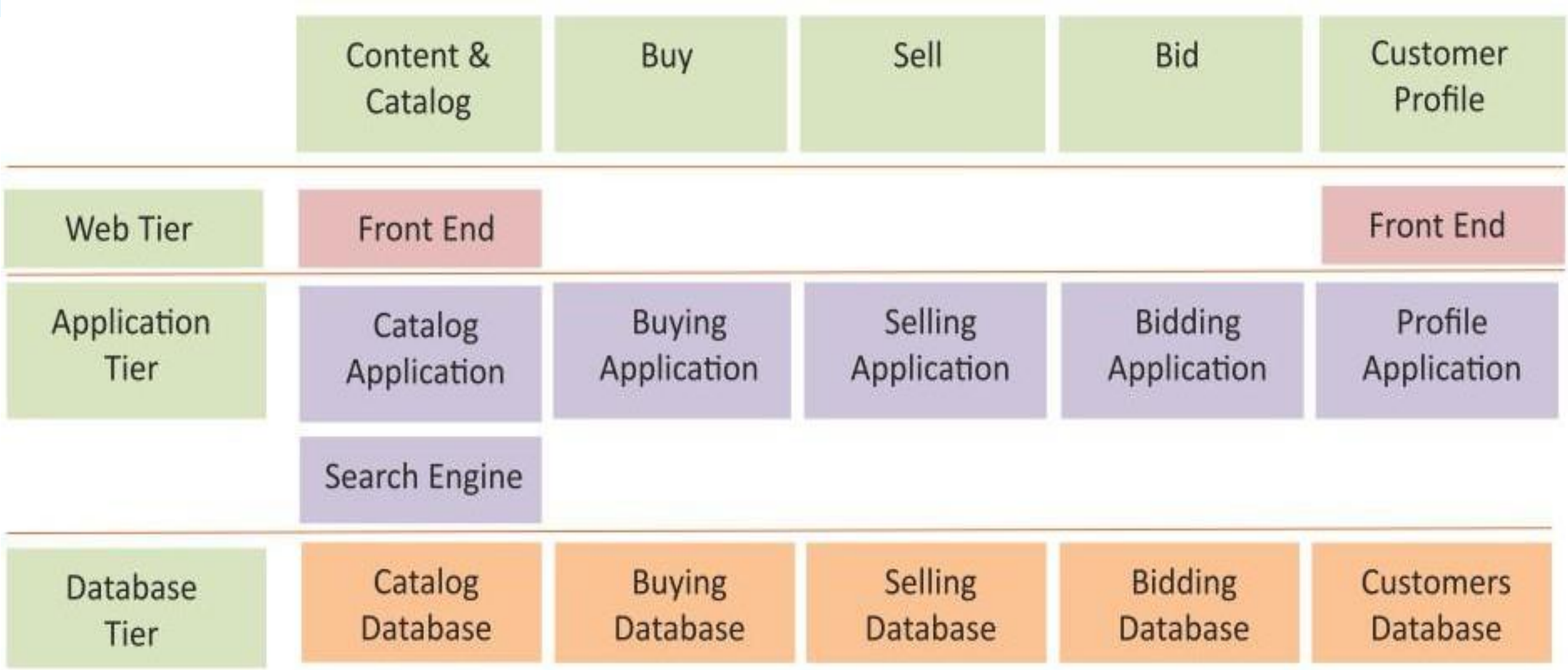


(b)

CCM Component Design

- Created for the application based on comprehensive analysis of the application's functions and building blocks.
- Allows identifying the building blocks of a cloud application which are classified based on the functions performed and type of cloud resources required.
- Each building block performs a set of actions to produce the desired outputs for other components.
- Each component takes specific inputs, performs a pre- defined set of actions and produces the desired outputs.
- Components offer their functions as services through a functional interface which can be used by other components.
- Components report their performance to a performance database through a performance interface.

CCM Component Design

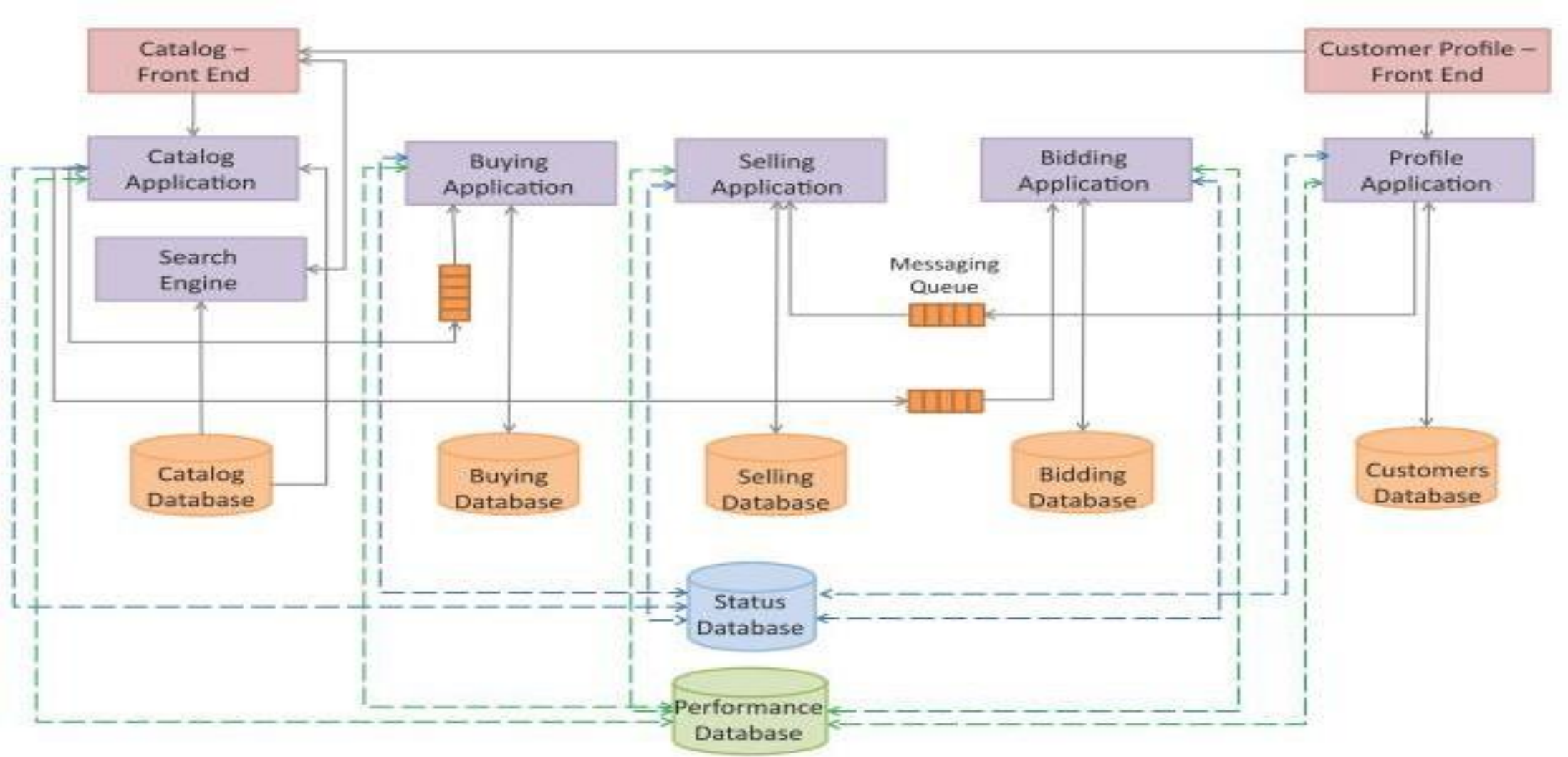


CCM map for an e-Commerce application

CCM Architecture Design

- Interactions between the application components are defined.
- CCM components have the following characteristics:
  - **Loose Coupling**
  - **Asynchronous Communication**
    - It is possible to add capacity by adding additional servers when the application load increases. Asynchronous communication is made possible by using messaging queues.
  - **Stateless Design**
    - Components in the Cloud Component Model are stateless. By storing session state outside of the component (e.g. in a database), stateless component design enables distribution and horizontal scaling.

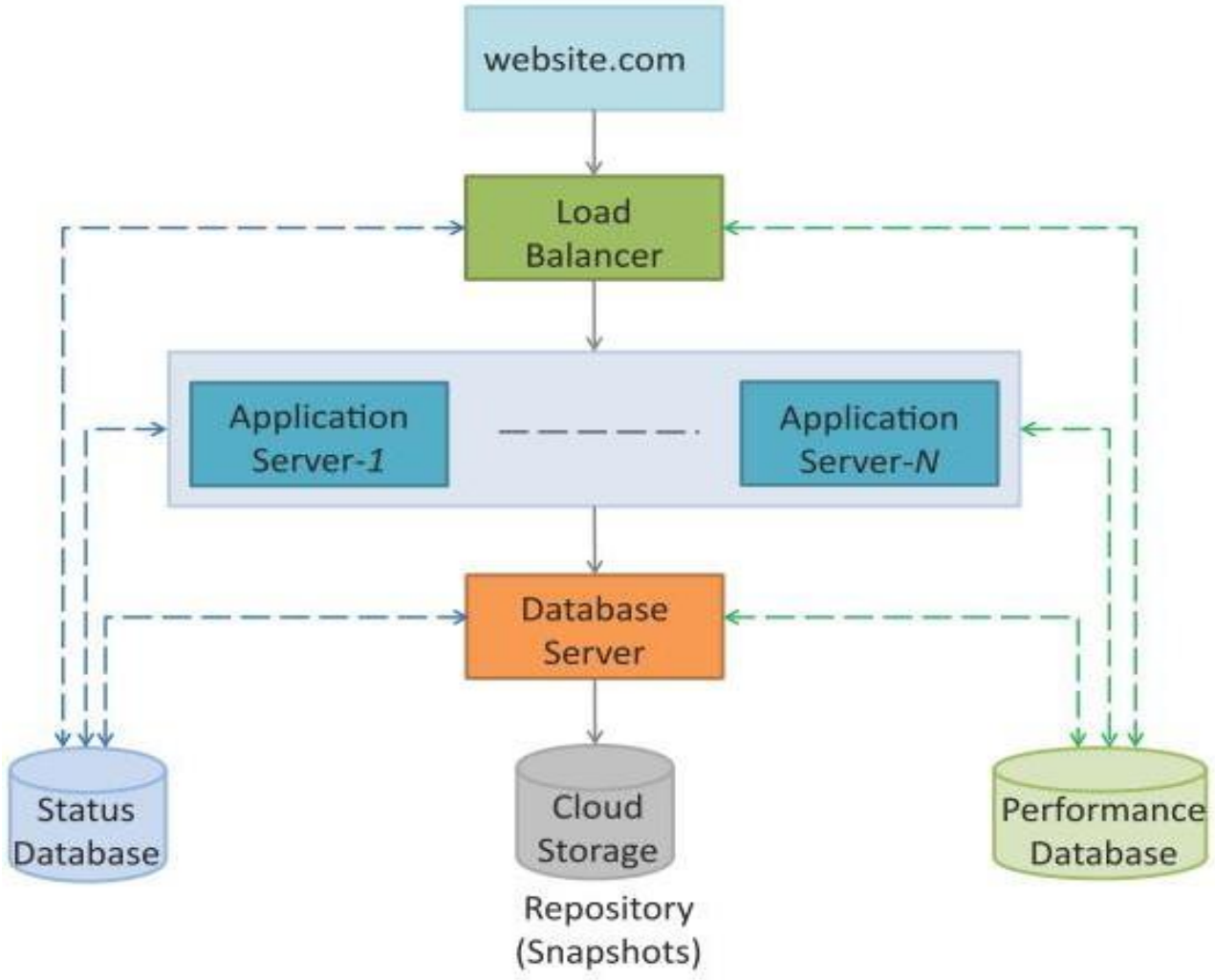
CCM Architecture Design



Architecture design of an e-Commerce application.

CCM Deployment Design

- Application components are mapped to specific cloud resources such as web servers, application servers, database servers, etc.
- Components can be deployed independently of each other.
- This approach makes it easy to migrate application components from one cloud to the other.
- With this fiexibility in application design and deployment, the application developers can ensure that the applications meet the performance and cost requirements with changing contexts.



Deployment design of an e-Commerce application

SOA vs CCM

Similarities

	SOA	CCM
Standardization & Re-use	SOA advocates principles of reuse and well defined relationship between service provider and service consumer.	CCM is based on reusable components which can be used by multiple cloud applications.
Loose coupling	SOA is based on loosely coupled services that minimize dependencies.	CCM is based on loosely coupled components that communicate asynchronously
Statelessness	SOA services minimize resource consumption by deferring the management of state information.	CCM components are stateless. State is stored outside of the components.

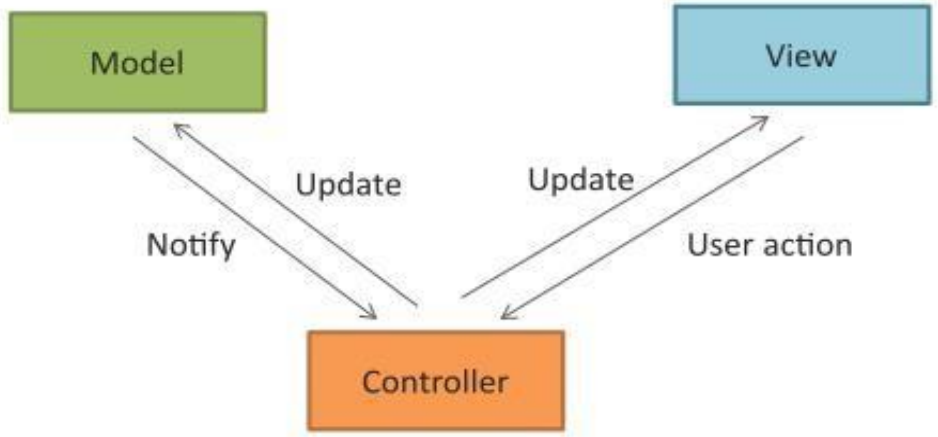
SOA vs CCM

Differences

	SOA	CCM
End points	SOA services have small and well-defined set of endpoints through which many types of data can pass.	CCM components have very large number of endpoints. There is an endpoint for each resource in a component, identified by a URI.
Messaging	SOA uses a messaging layer above HTTP by using SOAP which provide prohibitive constraints to developers.	CCM components use HTTP and REST for messaging.
Security	Uses WS-Security , SAML and other standards for security	CCM components use HTTPS for security.
Interfacing	SOA uses XML for interfacing.	CCM allows resources in components represent different formats for interfacing (HTML, XML, JSON, etc.).
Consumption	Consuming traditional SOA services in a browser is cumbersome.	CCM components and the underlying component resources are exposed as XML, JSON (and other formats) over HTTP or REST, thus easy to consume in the browser.

Model View Controller

- **Model** Manages the data and the behavior of the applications, processes events sent by the controller. Model has no information about the views and controllers. Model responds to the requests for information about its state (from the view) and responds to the instructions to change state (from controller).
- **View** prepares the interface which is shown to the user. Users interact with the application through views. Views present the information that the model or controller tell the view to present to the user and also handle user requests and sends them to the controller.
- **Controller** glues the model to the view, processes user requests and updates the model when the user manipulates the view. Controller also updates the view when the model changes.



RESTful Web Services

- Representational State Transfer (REST) is a set of architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred.
- The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia system.
- A RESTful web service is a web API implemented using HTTP and REST principles.
- The REST architectural constraints are as follows:
  - Client-Server
  - Stateless
  - Cacheable
  - Layered System
  - Uniform Interface
  - Code on demand

Relational Databases

- A relational database is database that conforms to the relational model that was popularized by Edgar Codd in 1970.
- The 12 rules that Codd introduced for relational databases include:
  - Information rule
  - Guaranteed access rule
  - Systematic treatment of null values
  - Dynamic online catalog based on relational model
  - Comprehensive sub-language rule
  - View updating rule
  - High level insert, update, delete
  - Physical data independence
  - Logical data independence
  - Integrity independence
  - Distribution independence
  - Non-subversion rule

Relational Databases

- Relations
  - A relational database has a collection of relations (or tables). A relation is a set of tuples (or rows).
- Schema
  - Each relation has a fixed schema that defines the set of attributes (or columns in a table) and the constraints on the attributes.
- Tuples
  - Each tuple in a relation has the same attributes (columns). The tuples in a relation can have any order and the relation is not sensitive to the ordering of the tuples.
- Attributes
  - Each attribute has a domain, which is the set of possible values for the attribute.
- Insert/Update/Delete
  - Relations can be modified using insert, update and delete operations. Every relation has a primary key that uniquely identifies each tuple in the relation.
- Primary Key
  - An attribute can be made a primary key if it does not have repeated values in different tuples.

ACID Guarantees

- Relational databases provide ACID guarantees.
- Atomicity
  - Atomicity property ensures that each transaction is either “all or nothing”. An atomic transaction ensures that all parts of the transaction complete or the database state is left unchanged.
- Consistency
  - Consistency property ensures that each transaction brings the database from one valid state to another. In other words, the data in a database always conforms to the defined schema and constraints.
- Isolation
  - Isolation property ensures that the database state obtained after a set of concurrent transactions is the same as would have been if the transactions were executed serially. This provides concurrency control, i.e. the results of incomplete transactions are not visible to other transactions. The transactions are isolated from each other until they finish.
- Durability
  - Durability property ensures that once a transaction is committed, the data remains as it is, i.e. it is not affected by system outages such as power loss. Durability guarantees that the database can keep track of changes and can recover from abnormal terminations.

Non-Relational Databases

- Have better horizontal scaling capability and improved performance for big data at the cost of less rigorous consistency models.
- Unlike relational databases, non-relational databases do not provide ACID guarantees.
- Most non-relational databases offer “eventual” consistency, which means that given a sufficiently long period of time over which no updates are made, all updates can be expected to propagate eventually through the system and the replicas will be consistent.
- The driving force behind the non-relational databases is the need for databases that can achieve high scalability, fault tolerance and availability.
- These databases can be distributed on a large cluster of machines. Fault tolerance is provided by storing multiple replicas of data on different machines.

Non-Relational Databases - Types

- Key-value store
  - Key-value store databases are suited for applications that require storing unstructured data without a fixed schema. Most key-value stores have support for native programming language data types.
- Document store
  - Document store databases store semi-structured data in the form of documents which are encoded in different standards such as JSON, XML, BSON, YAML, etc.
- Graph store
  - Graph stores are designed for storing data that has graph structure (nodes and edges). These solutions are suitable for applications that involve graph data such as social networks, transportation systems, etc.
- Object store
  - Object store solutions are designed for storing data in the form of objects de?ned in an object-oriented programming language.