Архитектура облачного приложения

План

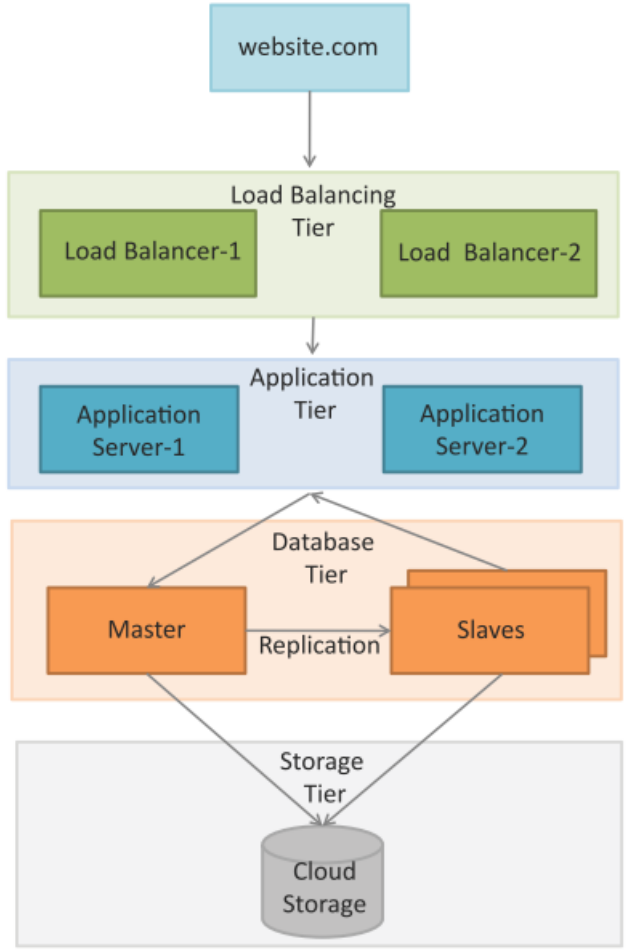
- Cloud Application Design Considerations
- Cloud Application Reference Architectures
- Design Methodologies
- Data Storage
- Data Analytics
- Развертывание и управление

Вопросы проектирования облачных приложений

- Масштабируемость
  - Масштабируемость является важным фактором, который побуждает разработчиков приложений переходить к облачным вычислениям. Создание приложений, которые могут обслуживать миллионы пользователей без ущерба для их производительности, всегда было сложной задачей. С ростом разработчиков облачных вычислений разработчики могут предоставить достаточные ресурсы для удовлетворения своих уровней рабочей нагрузки.
- Надежность и Доступность
  - Надежность системы определяется как вероятность того, что система будет выполнять предполагаемые функции в указанных условиях в течение определенного периода времени. Доступность - это вероятность того, что система будет выполнять заданную функцию в заданных условиях в заданное время.
- Защита
  - Безопасность является важным аспектом проектирования для облачных приложений, учитывая аутсорсинговый характер среды облачных вычислений.
- Техническое обслуживание и модернизация
  - Чтобы достичь быстрого выхода на рынок, компании обычно запускают свои приложения с готовым набором основных функций, а затем поэтапно добавляют новые функции по мере их завершения. В таких сценариях важно проектировать приложения с низкими расходами на техническое обслуживание и модернизацию.

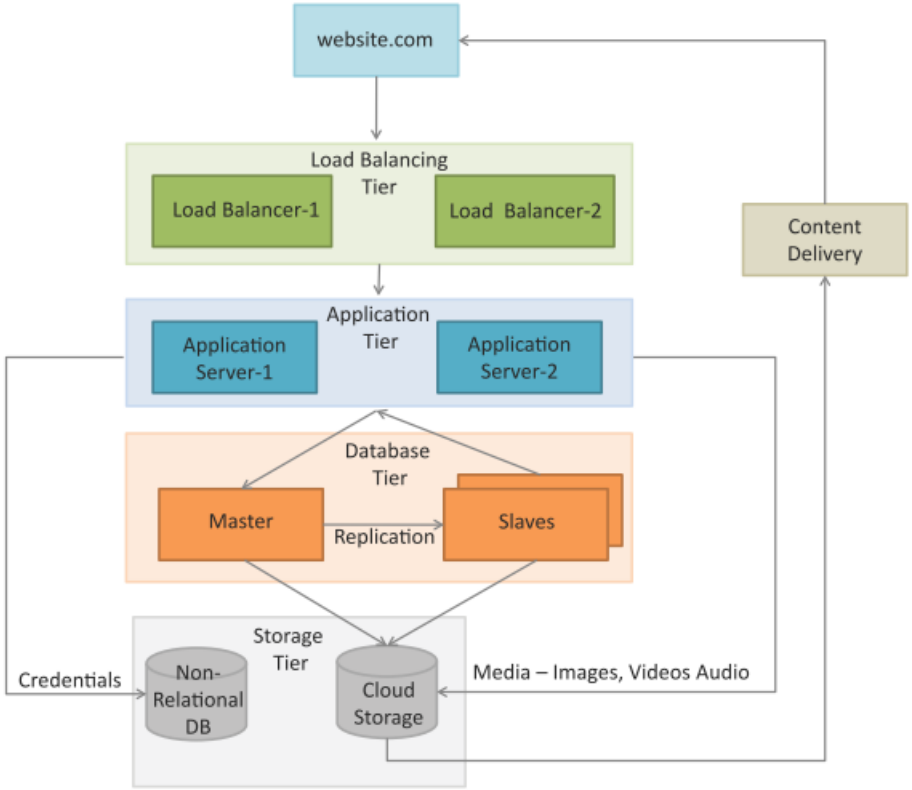
Примеры архитектур –
Электронная коммерция, бизнес-приложения для бизнеса, банковские и финансовые приложения

- Уровень балансировки
  - Уровень балансировки нагрузки состоит из одного или нескольких балансировщиков нагрузки.
- Уровень приложения
  - Для этого уровня рекомендуется настроить автоматическое масштабирование.
  - Автомасштабирование может быть запущено, если записанные значения для любого из указанных показателей, таких как загрузка процессора, использование памяти и т.д., Превышают определенные пороговые значения



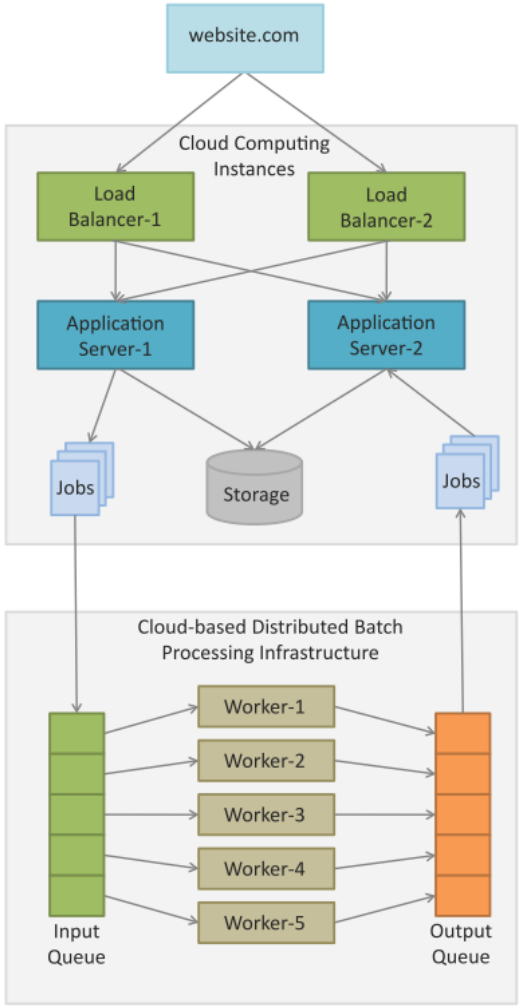
Примеры архитектур –
Приложения для доставки контента

- На рисунке показана типичная архитектура развертывания для приложений доставки контента, таких как фотоальбомы онлайн, веб-трансляция видео и т. д.
- В этом развертывании показаны как реляционные, так и нереляционные хранилища данных.
- Для доставки носителей используется сеть доставки контента (CDN), которая состоит из глобальной сети пограничных расположений.
- CDN используется для ускорения доставки статического контента, такого как изображения и видео.



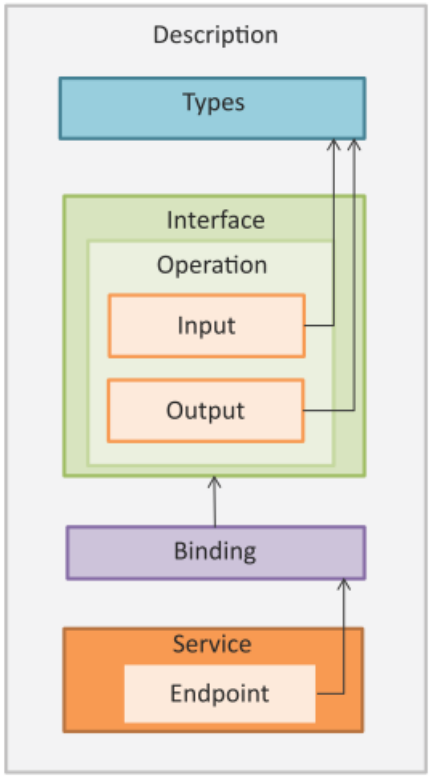
Примеры архитектур –
Приложения для аналитики данных

- На рисунке показана типичная архитектура развертывания для приложений с интенсивным вычислением, таких как Data Analytics, Media Transcoding и т. Д.
- Состоит из веб-приложений, хранилищ, вычислений / аналитики и уровней базы данных.
- Уровень аналитики состоит из облачных распределенных пакетных платформ обработки данных, таких как Hadoop, которые подходят для анализа больших данных.
- Задания анализа данных (например, MapReduce) отправляются на уровень аналитики с серверов приложений.
- Задания ставятся в очередь для выполнения, и по завершении анализируемые данные представляются с серверов приложений.



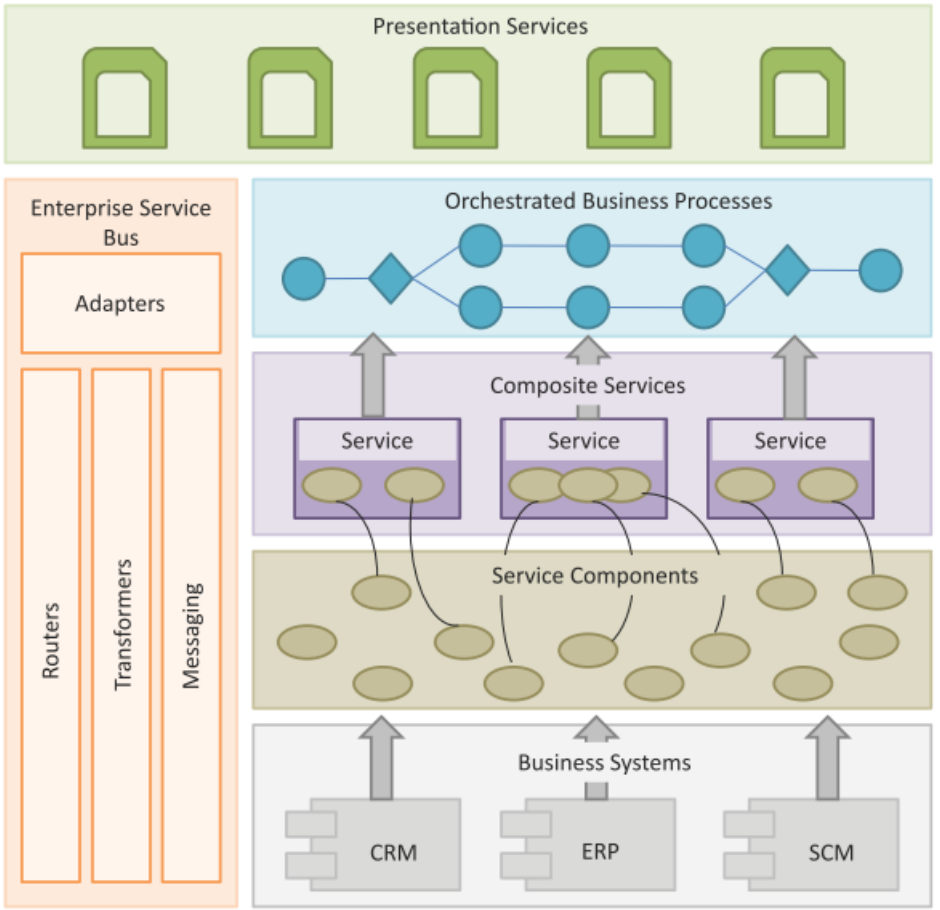
Service Oriented Architecture

- Сервис-ориентированная архитектура (SOA) - это хорошо зарекомендовавший себя архитектурный подход для проектирования и разработки приложений в виде сервисов, которые могут совместно использоваться и повторно использоваться.
- SOA представляет собой набор дискретных программных модулей или сервисов, которые составляют часть приложения и в совокупности обеспечивают функциональность приложения.
- Службы SOA разрабатываются в виде слабосвязанных модулей без встроенных в службы аппаратных вызовов.
- Эти службы обмениваются сообщениями друг с другом.
- Службы описываются с помощью языка описания веб-служб (WSDL).
- WSDL - это язык описания веб-сервисов на основе XML, который используется для создания описаний служб, содержащих информацию о функциях, выполняемых службой, и входах и выходах службы.



SOA Layers

- Бизнес-системы
  - Этот уровень состоит из заказных встроенных приложений и унаследованных систем, таких как Enterprise Resource Planning (ERP), CRM (Управление взаимоотношениями с клиентами), Supply Chain Management (SCM) и т. д.
- Компоненты службы
  - Компоненты службы позволяют вышеперечисленным уровням взаимодействовать с бизнес-системами. Компоненты сервиса несут ответственность за реализацию функциональности предоставляемых услуг.
- Композитные услуги
  - Это крупномасштабные службы, которые состоят из двух или более компонентов обслуживания. Композитные службы могут использоваться для создания компонентов масштаба предприятия или отдельных компонентов бизнес-единицы.
- Управляемые бизнес-процессы
  - Композитные службы могут быть организованы для создания бизнес-процессов более высокого уровня. В этих слоях определяются композиции и оркестровки составных сервисов для создания бизнес-процессов.
- Презентационные услуги
  - Это самый верхний уровень, который включает пользовательские интерфейсы, предоставляющие пользователям сервисы и организованные бизнес-процессы.
- Корпоративная служебная шина
  - Этот уровень объединяет службы с помощью адаптеров, маршрутизации, преобразования и обмена сообщениями.

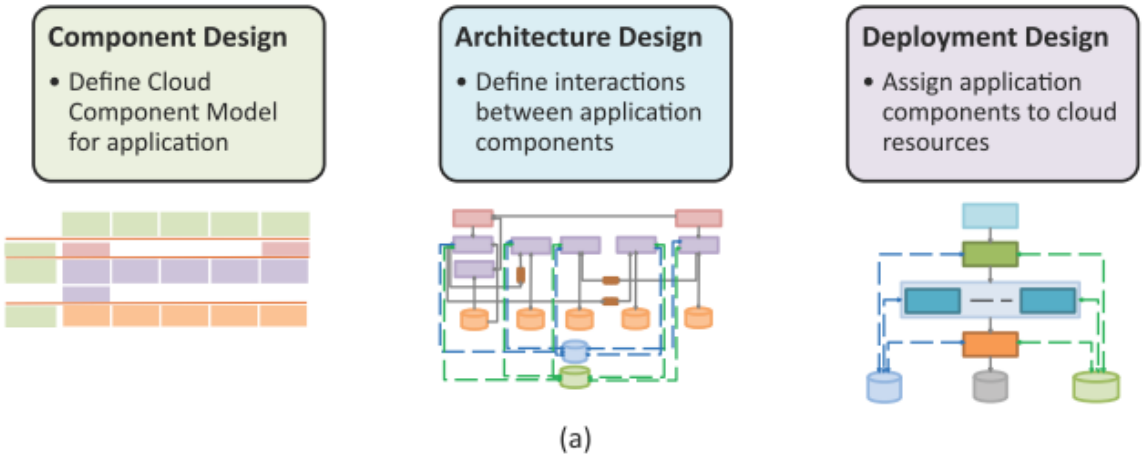


Cloud Component Model

- Cloud Component Model - это методология проектирования приложений, которая обеспечивает гибкий способ создания облачных приложений в быстром, удобном и независимом от платформы режиме.
- CCM - это архитектурный подход для облачных приложений, который не привязан к какому-либо определенному языку программирования или облачной платформе.
- Облачные приложения, разработанные с использованием подхода CCM, могут иметь развернуты в гибридных облаках, в которых различные компоненты приложения могут быть развернуты в облачной инфраструктуре и платформах различных поставщиков облачных решений.
- Приложения, разработанные с использованием CCM, обладают большей мобильностью и функциональной совместимостью.
- Приложения на основе CCM обладают лучшей масштабируемостью благодаря развязке компонентов приложения и обеспечению асинхронных механизмов связи.

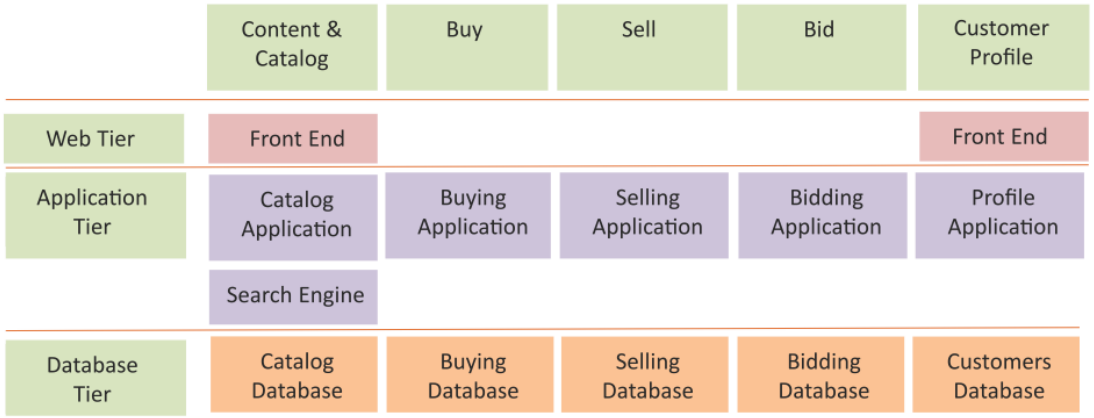
CCM Application Design Methodology

- Подход CCM к разработке приложений включает:
  - Проектирование компонентов
  - Архитектурный дизайн
  - Дизайн развертывания



CCM Component Design

- Модель облачного компонента создается для приложения на основе всестороннего анализа функций и строительных блоков приложения.
- Модель облачных компонентов позволяет идентифицировать строительные блоки облачного приложения, которые классифицируются на основании выполняемых функций и типа требуемых облачных ресурсов.
- Каждый строительный блок выполняет набор действий для получения желаемых результатов для других компонентов.
- Каждый компонент берет определенные входы, выполняет заранее определенный набор действий и выдает желаемые результаты.
- Компоненты предлагают свои функции в качестве сервисов через функциональный интерфейс, который может использоваться другими компонентами.
- Компоненты сообщают о своей производительности базе данных производительности через интерфейс производительности.

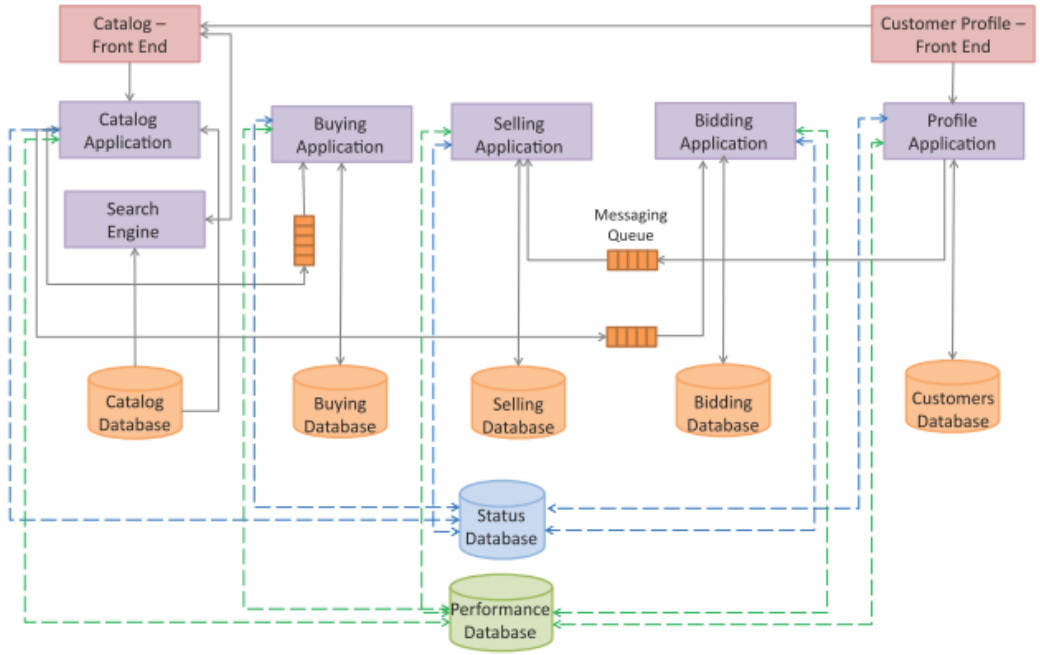


CCM map for an e-Commerce application

CCM Architecture Design

- In Architecture Design step, interactions between the application components are defined.

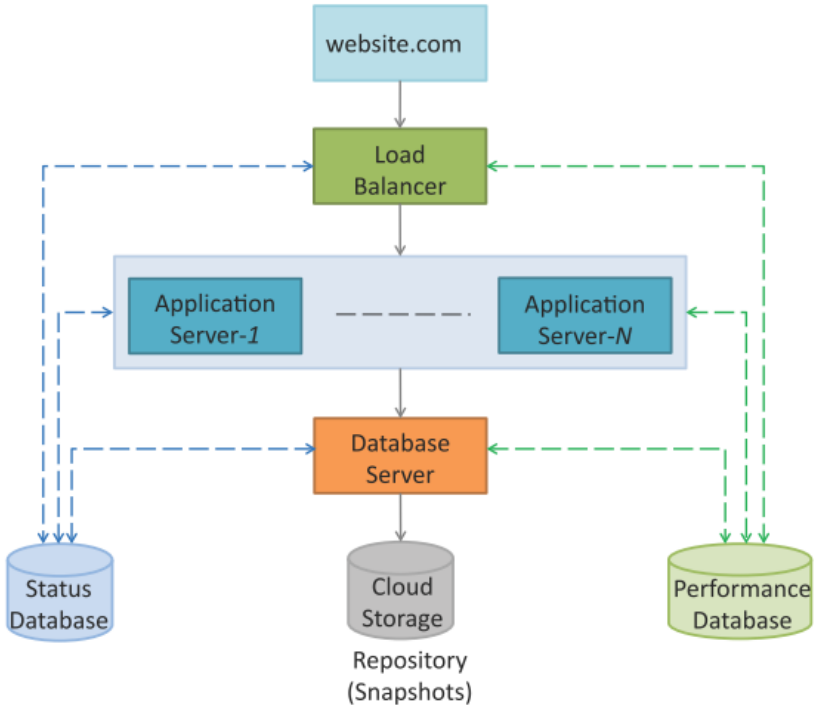
- CCM components have the following characteristics:
  - Loose Coupling
    - Components in the Cloud Component Model are loosely coupled.
  - Асинхронная связь
    - Разрешая асинхронную связь между компонентами, можно добавить емкость, добавив дополнительные серверы, когда нагрузка приложения возрастает. Асинхронная связь становится возможной благодаря использованию очередей сообщений.



Architecture design of an e-Commerce application.

CCM Deployment Design

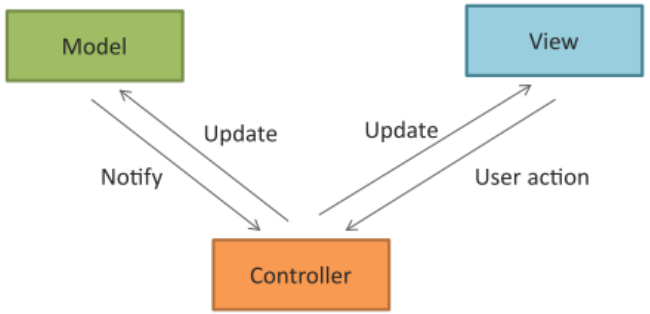
- На этапе разработки развертывания компоненты приложения сопоставляются с конкретными облачными ресурсами, такими как веб-серверы, серверы приложений, серверы баз данных и т. Д.
- Поскольку компоненты приложения разработаны с учетом их асинхронной связи и не зависят от состояния, компоненты могут быть развернуты независимо друг от друга.
- Такой подход упрощает миграцию компонентов приложения из одного облака в другое.
- Благодаря такой гибкости в разработке и развертывании приложений разработчики приложений могут обеспечить соответствие приложений требованиям производительности и затрат при изменении контекстов.



Deployment design of an e-Commerce application

Model View Controller

- Model View Controller (MVC) is a popular software design pattern for web applications.
- Model
  - Модель управляет данными и поведением приложений. Модель обрабатывает события, отправленные контроллером. Модель не имеет информации о представлениях и контроллерах. Модель реагирует на запросы информации о своем состоянии (из представления) и отвечает на инструкции по изменению состояния (от контроллера).
- View
  - Представление готовит интерфейс, который отображается пользователю. Пользователи взаимодействуют с приложением через представления. Представления представляют информацию о том, что модель или контроллер сообщают представление пользователю, а также обрабатывают запросы пользователей и отправляют их контроллеру.
- Controller
  - Контроллер обрабатывает запросы пользователей и обновляет модель, когда пользователь манипулирует представлением. Контроллер также обновляет представление при изменении модели.



Topology and Orchestration Specification for Cloud Applications (TOSCA)

16 января 2014 года технический комитет OASIS TOSCA утвердил TOSCA 1.0 в качестве стандарта.

Цель проекта TOSCA — создать стандарт для взаимодействия облачных платформ. Идея стандарта состоит в том, чтобы избежать зависимости от конкретного поставщика (vendor lock-in), которая возникает из-за использования множества «облачных» API и слишком разрозненных технологий безопасности, управления и совместимости

OASIS TOSCA

Похожие проекты

- HOT
  - Heat Orchestration Template
  - Declarative
  - YAML
- TOSCA
  - Topology & Orchestration Standard for Cloud Application
  - Declarative & Imperative
  - XML and now YAML

Сходства

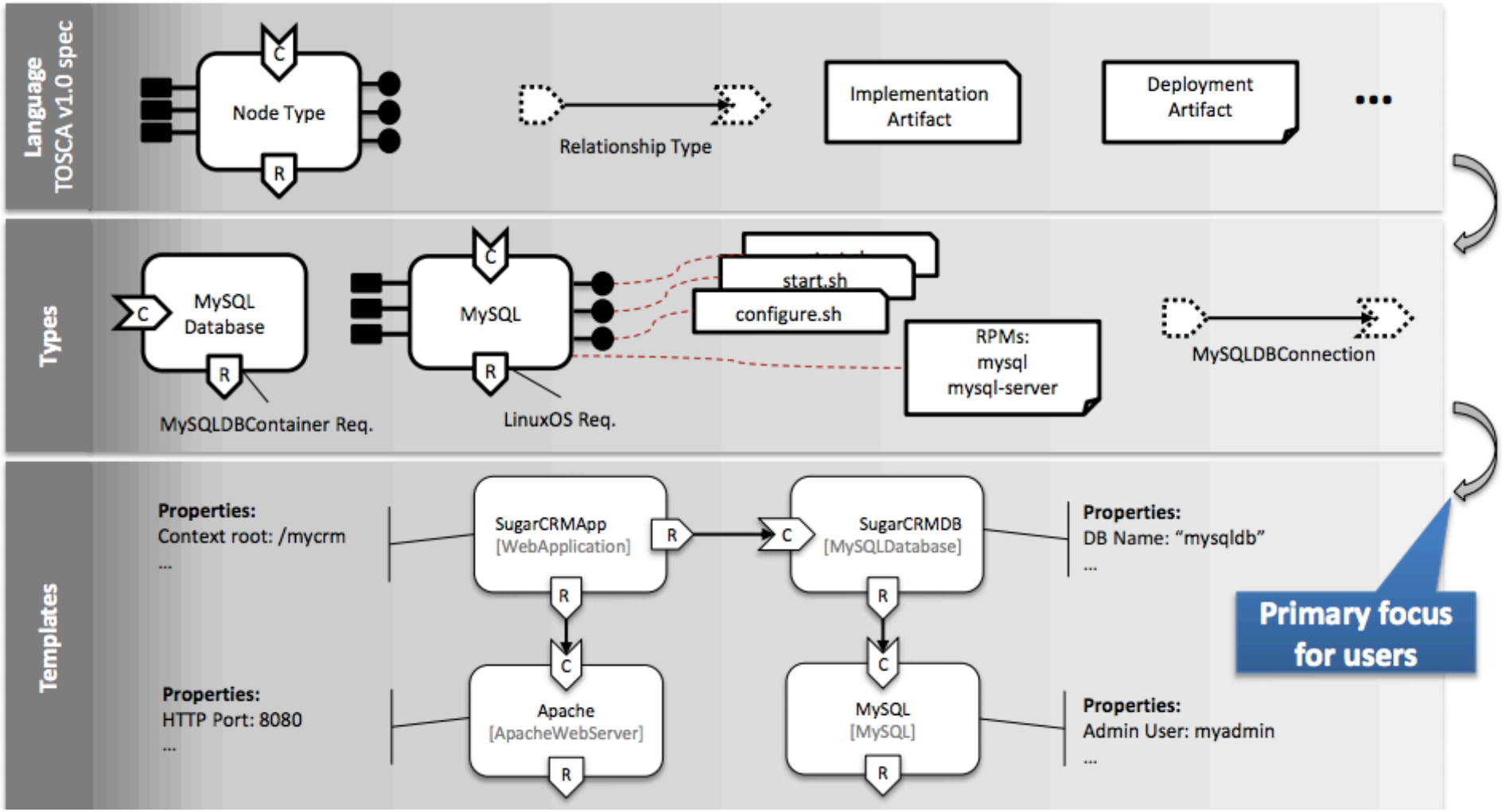
- Структурные сходства:
  - Описание
  - Входные и выходные параметры
  -
- Проектирование компонентов
  - компонент является объектом типа
  - зависит от других компонентов
  - имеет конфигурацию / начальное состояние в формате свойств

Различие

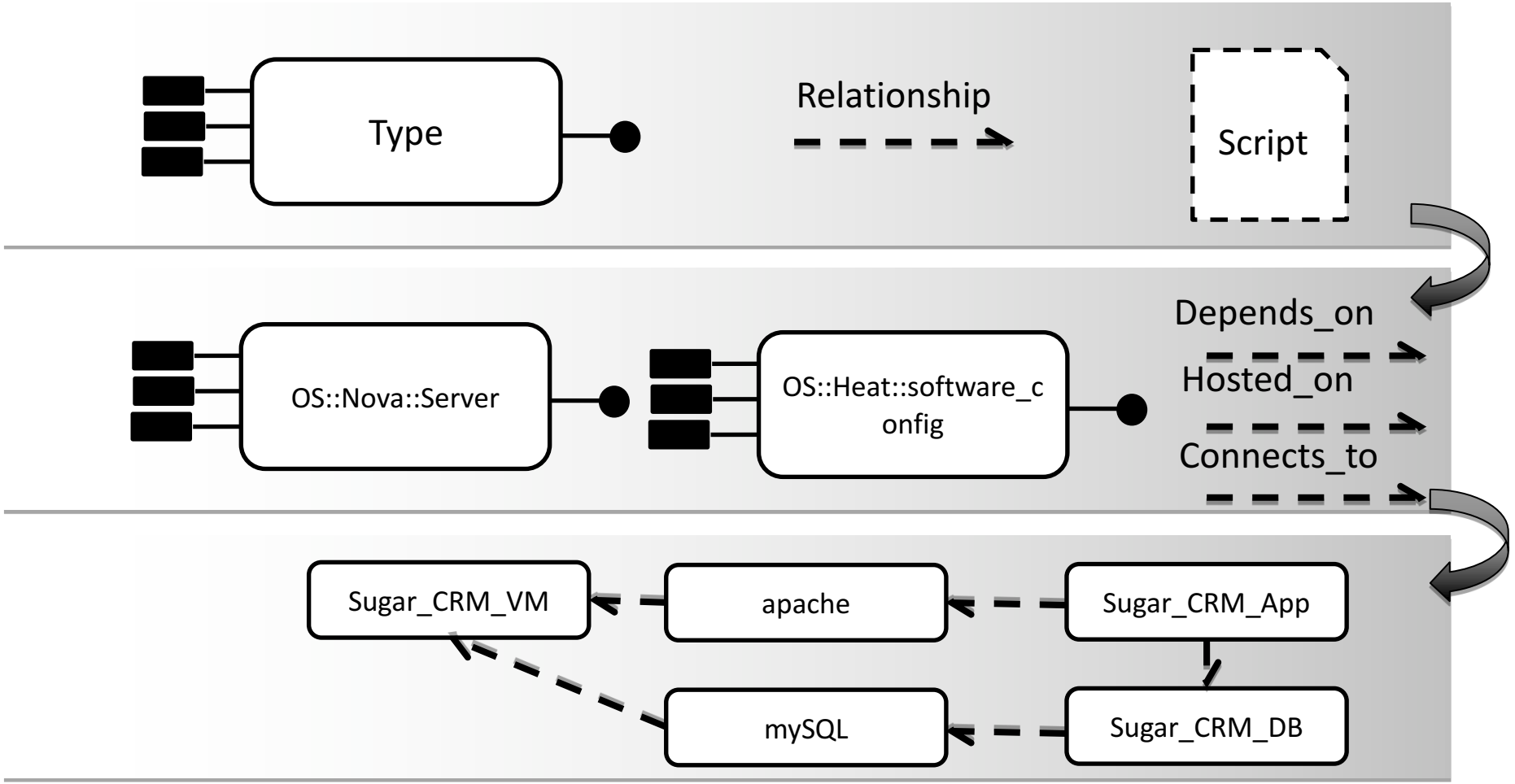
- Жизненный цикл
  - HOT имеет ограниченный жизненный
  - TOSCA использует интерфейсы для

- Поддержка работы приложения
  - HOT 100% декларативным.
  - TOSCA использует любые интерфейсы для любого описания любого процесса.

Sugar CRM Example



HOT


Type
Relationship
Script
OS::Nova::Server
OS::Heat::software_config
Depends_on
Hosted_on
Connects_to
Sugar_CRM_VM
apache
Sugar_CRM_App
mySQL
Sugar_CRM_DB

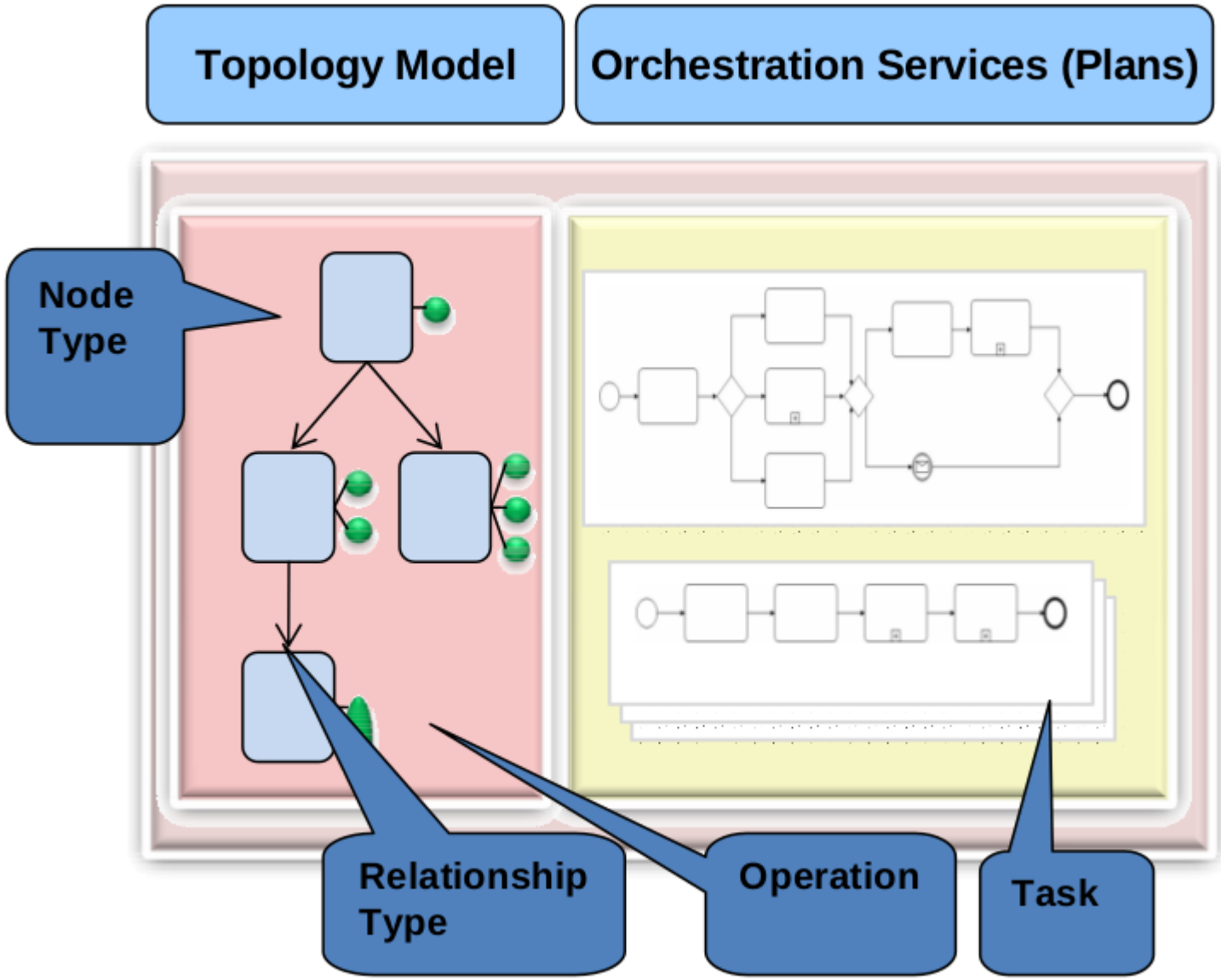
Описание инфраструктуры и самой функции

```
inputs:
  add_or_del:
    type: string
    constraints:
      - valid_values: [add, del]
  ip_or_url:
    type: string


node_templates:
  firewall:
    type: tosca.nodes.ImprovedSoftwareComponent
    requirements:
      - host: server
    interfaces:
      Standard:
        create:
          implementation: scripts/vnf_firewall.sh
          inputs:
            param1: create
    properties:
      actions:
        configure_firewall:
          program:
            implementation: scripts/vnf_firewall.sh
            inputs:
              param1: { get_input: add_or_del }
              param2: { get_input: ip_or_url }
        get_configuration:
          program:
            implementation: scripts/vnf_firewall.sh
            inputs:
              param1: show
            outputs:
              configuration:
                type: string
```

```
server:
  type: tosca.nodes.ImprovedCompute
  attributes:
    ports:
      in: port1
      out: port1
  capabilities:
    host:
      properties:
        disk_size: 10 GB
        mem_size: 2048 MB
        num_cpus: 1
  properties:
    os_image: tosca_zab

port1:
  type: tosca.nodes.network.Port
```


Topology Model
Orchestration Services (Plans)
Node Type
Relationship Type
Operation
Task