**Шаблоны бессерверной архитектуры (****Serverless Architecture)**

**1. Введение**

**1.1 Определение бессерверной архитектуры:**

* Базовые принципы и понятия.
* Отличие от традиционных серверных моделей.

**1.2 Основные элементы бессерверной архитектуры:**

* Функции (Functions) как основные строительные блоки.
* Событийные источники (Event Sources).
* Примеры облачных провайдеров (AWS Lambda, Azure Functions, Google Cloud Functions).

**2. Графическое представление шаблона**

**2.1 Диаграмма бессерверной архитектуры:**

* Иллюстрация основных компонентов и их взаимосвязей.
* Примеры типовых сценариев взаимодействия.

**3. Назначение и Цели Применения**

**3.1 Назначение:**

* Эффективное использование ресурсов.
* Автомасштабирование и гибкая настройка.

**3.2 Цели применения:**

* Уменьшение затрат и увеличение производительности.
* Ускорение развертывания и поставки продукта.

**4. Возможные Вариации и Модификации**

**4.1 Адаптация под проект:**

* Варианты интеграции с существующими приложениями.
* Особенности выбора облачного провайдера.

**4.2 Возможности расширения:**

* Использование дополнительных сервисов облачных провайдеров.
* Интеграция с микросервисной архитектурой.

**5. Особенности Применения: Примеры и Практические Кейсы**

**5.1 Примеры применения:**

* Обработка событий от IoT устройств.
* Автоматизация процессов обработки данных.

**5.2 Практические кейсы:**

* Разработка приложения с использованием бессерверной архитектуры.
* Опыт внедрения в реальных проектах.

**6. Заключение**

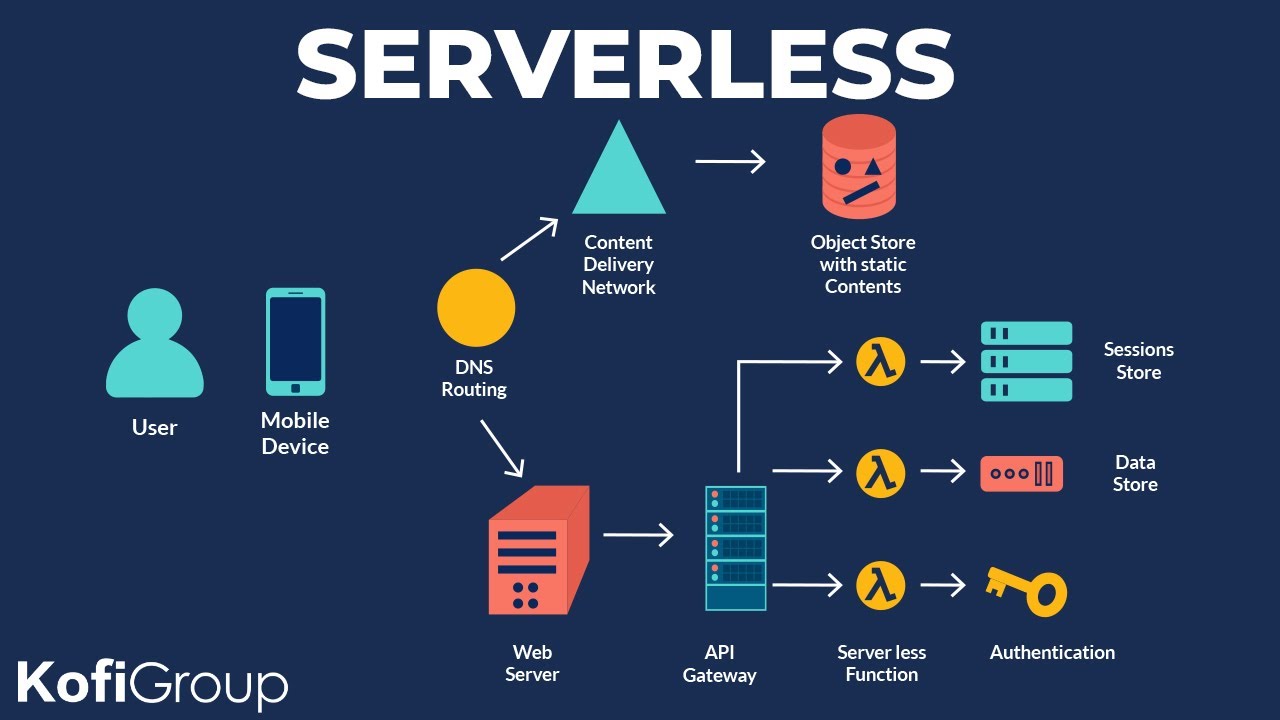
**6.1 Перспективы развития бессерверной архитектуры:**

* Тенденции в области технологий серверногоless.
* Роль бессерверных решений в будущем развитии информационных систем.

# **Вступление**

## **1.1 Определение бессерверной архитектуры:**

Бессерверная архитектура — это следующий этап эволюции облачных платформ в направлении полностью облачного машинного кода. С бессерверной архитектурой разработчики оказываются ближе к бизнес-логике и при этом могут не беспокоиться об инфраструктуре. Это шаблон, который не подразумевает "нет сервера", а "меньше сервера". Бессерверный код управляется событиями. Код может запускаться чем угодно — от традиционных веб-запросов HTTP до таймера или результата передачи файла. Инфраструктура без сервера позволяет мгновенно масштабироваться в соответствии с эластичными требованиями и предлагает микро-выставление счетов, чтобы действительно "платить за то, что вы используете".



В отличие от традиционных серверных моделей, где инфраструктура постоянно поддерживается, бессерверная архитектура предоставляет вычислительные ресурсы только при необходимости. Это позволяет оптимизировать использование ресурсов и снижает затраты.

*Свойства Serverless Architecture:*

1. Абстракция. Вы не управляете сервером, на котором запускается ваша программа. Вы вообще ничего про него не знаете, все нюансы операционной системы, обновлений, сетевых настроек и прочего спрятаны от вас. Это сделано для того, чтобы вы могли сосредоточиться на разработке полезной функциональности, а не на администрировании серверов.
2. Эластичность. Провайдер Serverless услуги автоматически предоставляет вам больше или меньше вычислительных ресурсов, в зависимости от того, насколько большая нагрузка приходится на ваше приложение.
3. Эффективная стоимость. Если ваше приложение простаивает — вы ничего не платите, т.к. оно в этот момент не использует вычислительных ресурсов. Оплата же происходит только за время, которое ваше приложение реально работает.
4. Ограниченный жизненный цикл. Ваше приложение запускается в контейнере, и, спустя короткое время, от десятка минут до нескольких часов, сервис автоматически его останавливает. Конечно же, если приложение снова должно быть вызвано — новый контейнер будет запущен.

## **1.2 Основные элементы бессерверной архитектуры**

Функции (Functions) как основные строительные блоки:

Функции представляют из себя небольшие, независимые куски кода, выполняющие конкретные задачи. Они являются основными строительными блоками бессерверных приложений и запускаются в ответ на события, такие как HTTP-запросы или изменения в хранилище данных.

Событийные источники (Event Sources):

Событийные источники – это источники данных, которые могут вызывать выполнение функций. Примерами могут служить изменения в базе данных, поступление новых данных из потока данных или HTTP-запросы. Эти события активируют выполнение функций.