Лекция 6
Базы данных

План занятия

- Виртуализация
- Работа с БД
  - Транзакции
  - Оптимизация запросов
- Чистая архитектура

Виртуализация

Гипервизорная виртуализация

•**Гипервизор** — это программное обеспечение, которое управляет виртуальными машинами (VM), создавая виртуализированные ресурсы и распределяя их между виртуальными машинами.

Виртуализация уровня ядра

•**Виртуализация уровня ядра** происходит на уровне операционной системы. Она использует возможности ядра ОС для создания изолированных сред (контейнеров), которые делят общие ресурсы с хост-системой, но изолированы друг от друга.

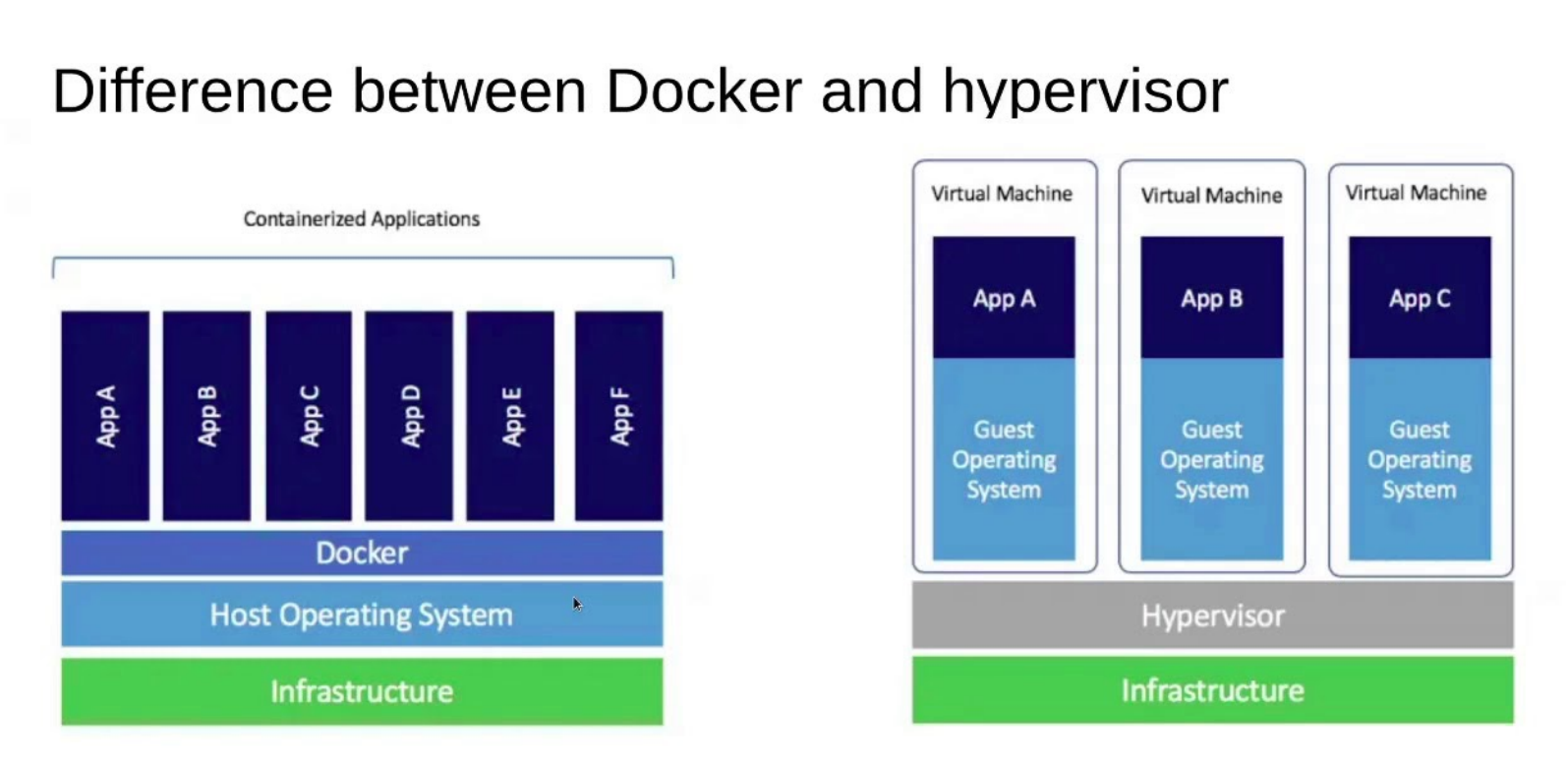
Виртуализация - Docker

Docker — это платформа для создания, распространения и запуска приложений в изолированных средах, называемых **контейнерами**.

Контейнеры в Docker используют возможности **виртуализации уровня ядра.**

Docker позволяет упаковывать приложения и их зависимости в единый контейнер, который может быть запущен на любой машине, где установлен Docker.

Виртуализация



Docker

Hypervisor

Работа с БД

Введение в SQL

SQL (Structured Query Language) — это язык программирования, используемый для управления данными в реляционных базах данных.

Основные команды SQL

- **SELECT** — используется для извлечения данных из базы данных.
- **INSERT** — добавляет новые записи в таблицу.
- **UPDATE** — изменяет существующие записи в таблице.
- **DELETE** — удаляет записи из таблицы.
- **CREATE** — создает таблицы, базы данных, индексы.
- **ALTER** — изменяет структуру таблицы (добавление столбцов, изменение типов данных и т. д.).
- **DROP** — удаляет таблицы или базы данных.

Введение в SQL

Структура SQL-запроса:

1.SELECT: указывает, какие столбцы данных необходимо извлечь.

2.FROM: указывает таблицу или несколько таблиц, из которых будут извлечены данные.

3.WHERE: задает условие для фильтрации данных.

4.ORDER BY: сортирует результат по указанным столбцам.

5.GROUP BY: группирует строки по указанным столбцам.

6.HAVING: фильтрует сгруппированные данные (аналог WHERE для групп).

7.LIMIT/OFFSET: ограничивает количество возвращаемых строк.

Пример SQL-запроса:

```
SELECT *
FROM candles_1m
WHERE instrument='AAPL’
       AND open<close
ORDER BY ts
LIMIT 10;
```

Пакеты SQL: database/sql

https://pkg.go.dev/database/sql

Пакет для работы с реляционными БД. Содержит все
необходимые функции и методы: Open, Close, Exec, Query,
QueryRow, Begin, Commit, Rollback, Prepare.

- Open – открывает соединение с БД
- Close – закрывает соединение. Если коннект не закрыть, он останется открытым на стороне БД
- Exec – выполняет запрос, возвращает интерфейс sql.Result (используется для DELETE, UPDATE, TRUNCATE, ...)
- Query – выполняет запрос и возвращает курсор *sql.Rows
- QueryRow – выполняет запрос, умеет вычитывать один результат, возвращает *sql.Row
- Begin – открывает транзакцию, которую обязательно нужно либо завершить (Commit), либо откатить/отменить (Rollback)

database/sql: query

```
1  func main() {
2    ...
3
4    const selectCandlesQuery = `SELECT instrument, period, ts, open, high, low,
5      close FROM candles_1m`
6    rows, err := db.Query(selectCandlesQuery)
7    if err != nil {
8      log.Fatal(err)
9    }
10
11   defer rows.Close()
12
13   var candles []Candle
14   for rows.Next() {
15     var c Candle
16     err = rows.Scan(&c.Ticker, &c.Period, &c.TS, &c.Open, &c.High, &c.Low, &c.Close)
17     if err != nil {
18       log.Fatal(err)
19     }
20     candles = append(candles, c)
21   }
22   if err = rows.Err(); err != nil {
23     log.Fatal(err)
24   }
25 }
```

database/sql: custom types

```go
type CandlePeriod int // in database string

const (
  CandlePeriod1m CandlePeriod = iota
  CandlePeriod2m
  CandlePeriod10m
  CandlePeriodUnknown
)

func (p *CandlePeriod) Scan(value interface{}) error {
  asBytes, ok := value.([]byte)
  if !ok {
     return errors.New("scan source is not string")
  }

  asString := string(asBytes)

  switch asString {
  case "1m":
     *p = CandlePeriod1m
  case "2m":
     *p = CandlePeriod2m
  case "10m":
     *p = CandlePeriod10m
  default:
     *p = CandlePeriodUnknown
  }

  return nil
}
```

```go
func (p CandlePeriod) Value() (driver.Value, error) {
  switch p {
  case CandlePeriod1m:
    return "1m", nil
  case CandlePeriod2m:
    return "2m", nil
  case CandlePeriod10m:
    return "10m", nil
  }
}
```

Пакеты SQL: jackc/pgx

- Support for approximately 70 different PostgreSQL types
- Automatic statement preparation and caching
- Batch queries
- Single-round trip query mode
- Full TLS connection control
- Binary format support for custom types (allows for much quicker encoding/decoding)
- COPY protocol support for faster bulk data loads
- Tracing and logging support
- Connection pool with after-connect hook for arbitrary connection setup
- LISTEN / NOTIFY
- Conversion of PostgreSQL arrays to Go slice mappings for integers, floats, and strings
- hstore support
- json and jsonb support
- Maps inet and cidr PostgreSQL types to netip.Addr and netip.Prefix
- Large object support
- NULL mapping to pointer to pointer
- Supports database/sql.Scanner and database/sql/driver.Valuer interfaces for custom types
- Notice response handling
- Simulated nested transactions with savepoints

jackc/pgx: query

```
1  func main() {
2    ...
3    const selectCandlesQuery = `SELECT instrument, period, ts, open, high, low,
4      close FROM candles_1m`
5    rows, err := conn.Query(context.Background(), selectCandlesQuery)
6    if err != nil {
7      log.Fatal(err)
8    }
9
10   defer rows.Close()
11
12   var candles []Candle
13   for rows.Next() {
14     var c Candle
15     err = rows.Scan(&c.Ticker, &c.Period, &c.TS, &c.Open, &c.High, &c.Low, &c.Close)
16     if err != nil {
17       log.Fatal(err)
18     }
19     candles = append(candles, c)
20   }
21   if err = rows.Err(); err != nil {
22     log.Fatal(err)
23   }
24 }
```

Транзакции

Транзакция — это логически завершенная последовательность операций, которая выполняется как единое целое.

Все операции в рамках транзакции должны либо успешно завершиться, либо не быть выполнены вообще (если транзакция отменяется).

Это гарантирует консистентность базы данных и предотвращает частичное выполнение операций, что может привести к ошибкам или несоответствиям.

Транзакции - ACID

Транзакции обеспечивают четыре основные характеристики, известные как ACID:

Atomicity (Атомарность):
- Все операции в транзакции либо выполняются полностью, либо не выполняются вовсе.

Consistency (Согласованность):
- Транзакция переводит базу данных из одного согласованного состояния в другое.

Isolation (Изолированность):
- Операции одной транзакции не видны другим транзакциям до завершения текущей транзакции.

Durability (Долговечность):
- После того как транзакция завершена, все изменения в базе данных сохраняются, даже в случае сбоя системы.

https://ru.wikipedia.org/wiki/ACID

Транзакции - ACID

Транзакции обеспечивают четыре основные характеристики, известные как ACID:

Atomicity (Атомарность):
- Все операции в транзакции либо выполняются полностью, либо не выполняются вовсе.

Consistency (Согласованность):
- Транзакция переводит базу данных из одного согласованного состояния в другое.

Isolation (Изолированность):
- Операции одной транзакции не видны другим транзакциям до завершения текущей транзакции.

Durability (Долговечность):
- После того как транзакция завершена, все изменения в базе данных сохраняются, даже в случае сбоя системы.

https://ru.wikipedia.org/wiki/ACID

Транзакции – Уровни изоляций

Уровни изоляции определяют поведение бд при параллельном выполнении транзакций.

SQL стандарт определяет **четыре уровня изоляции** транзакций:

1. READ UNCOMMITTED (Чтение не закоммиченых данных)
2. READ COMMITTED (Чтение закоммиченных данных)
3. REPEATABLE READ (Повторяемое чтение)
4. SERIALIZABLE (Сериализуемый)

Поведение при различных уровнях изолированности [править | править код]

«+» — предотвращает, «-» — не предотвращает.

Уровень изоляции	Фантомное чтение	Неповторяющееся чтение	«Грязное» чтение	«Грязная» запись[3] (аномалия не указана в стандарте SQL-99)
SERIALIZABLE	+	+	+	+
REPEATABLE READ	-	+	+	+
READ COMMITTED	-	-	+	+
READ UNCOMMITTED	-	-	-	+

ru.wikipedia.org/wiki/Уровень_изолированности_транзкций

Оптимизация запросов – EXPLAIN ANALYSE

```
EXPLAIN ANALYSE SELECT * FROM candles_1m
       WHERE instrument = 'AAPL’
       AND ts > '2024-09-06 13:50:00’
       AND ts < '2024-09-06 14:00:00';
```

```
Seq Scan on candles_1m (cost=0.00..66.40
rows=688 width=49)
(actual time=0.020..6.365 rows=688 loops=1)
Filter: (instrument = 'AAPL'::text)
Rows Removed by Filter: 2064
Planning Time: 0.132 ms
Execution Time: 11.722 ms
```

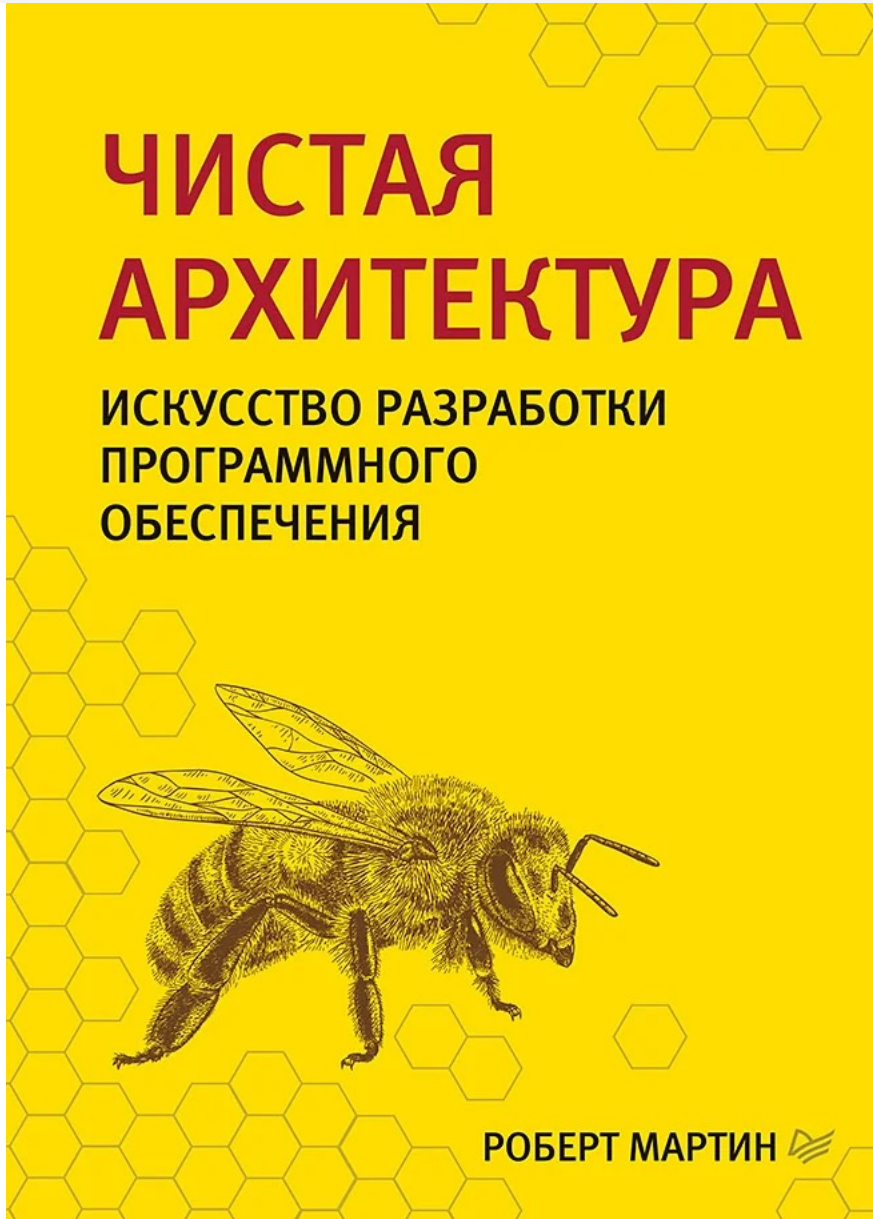
Оптимизация запросов – индексы

```
CREATE INDEX candles_1m_instrument_ts ON
candles_1m (instrument, ts);
```

```
EXPLAIN ANALYSE SELECT * FROM candles_1m
      WHERE instrument = 'AAPL’
      AND ts > '2024-09-06 13:50:00’
      AND ts < '2024-09-06 14:00:00';
```

```
Bitmap Heap Scan on candles_1m (cost=4.39..24.55 rows=9 width=49) (actual time=1.422..1.531 rows=9
loops=1)
Recheck Cond: ((instrument = 'AAPL'::text) AND (ts > '2021-09-06 13:50:00'::timestamp without time
zone) AND (ts < '2021-09-06 14:00:00'::timestamp without time zone))
Heap Blocks: exact=2
-> Bitmap Index Scan on candles_1m_instrument_ts (cost=0.00..4.39 rows=9 width=0) (actual
time=1.397..1.405 rows=9 loops=1)
Index Cond: ((instrument = 'AAPL'::text) AND (ts > '2021-09-06 13:50:00'::timestamp without
time zone) AND (ts < '2021-09-06 14:00:00'::timestamp without time zone))
Planning Time: 3.883 ms
Execution Time: 1.665 ms
```

Чистая архитектура



Чистая архитектура

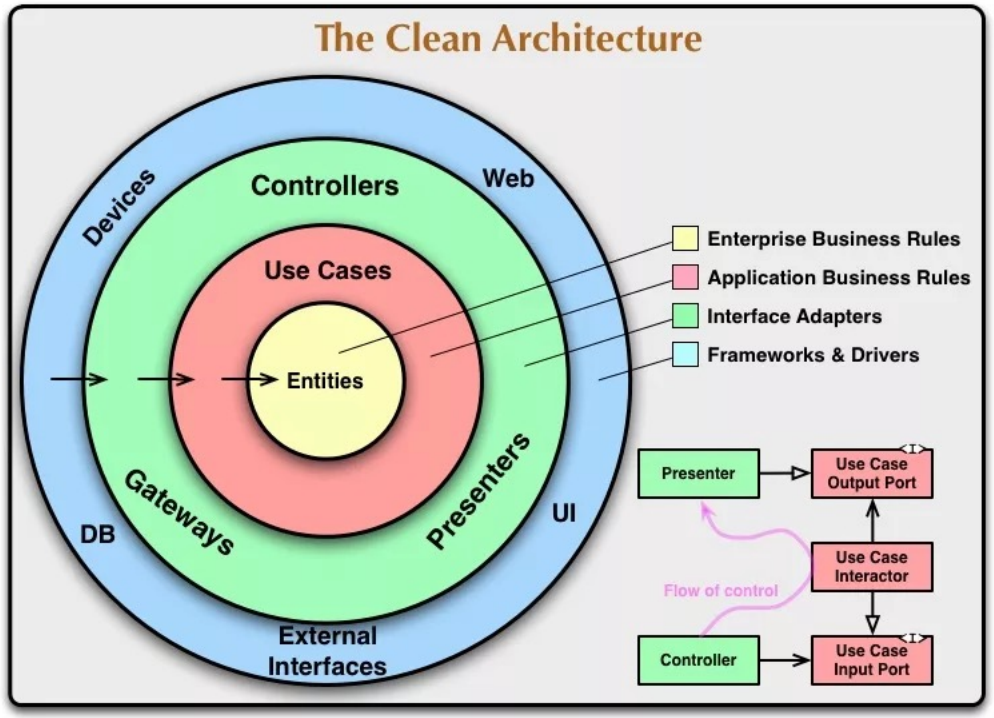
Архитектура – это подход к организации/структуризации кода, обеспечивающий масштабируемость приложения, надежность и удобство при доработках и тестировании.

Чистая архитектура – это подход к проектированию приложений, предложенный Робертом Мартином.

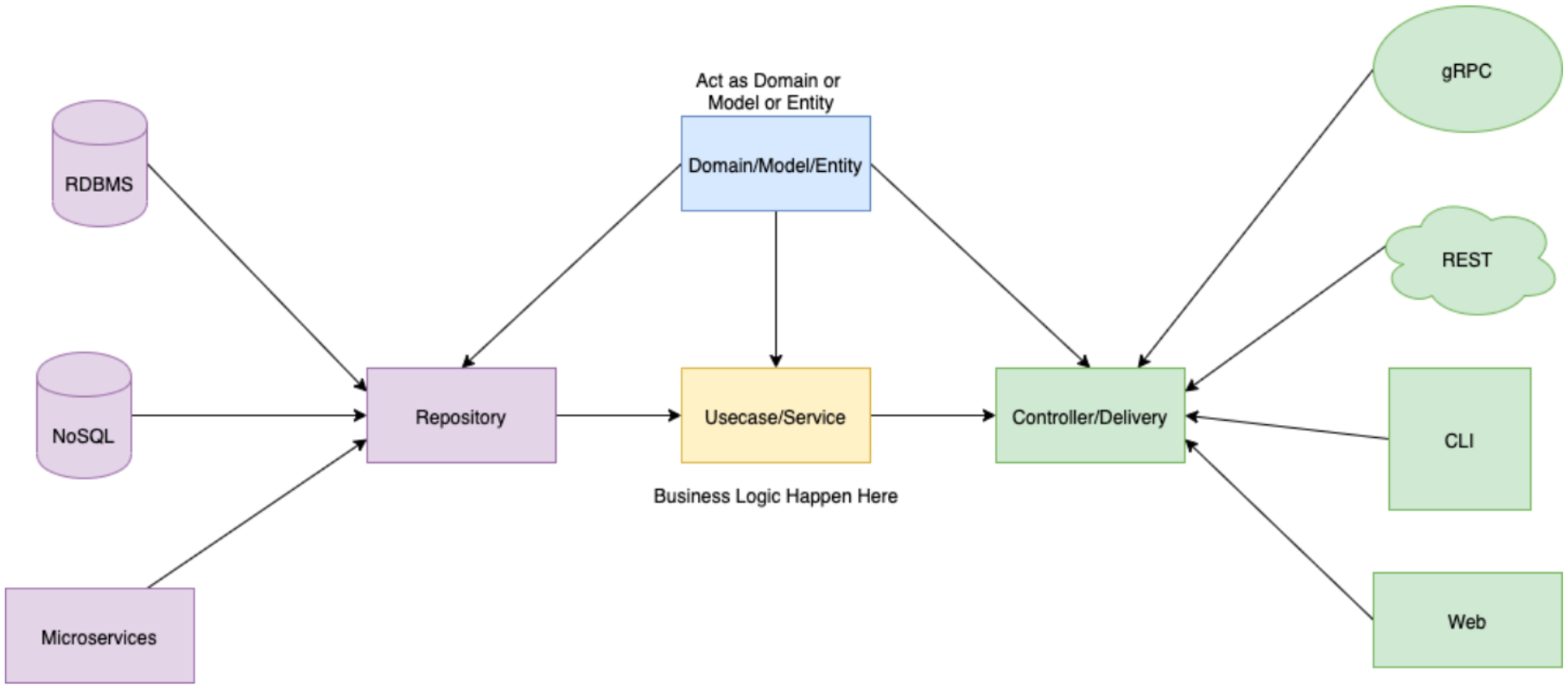
Чистая архитектура

Основные концепции:

- Разделение приложений на слои
- Внутренние слои системы (бизнес-логика, сущности) не должны зависеть от внешних слоев
- Зависимости между слоями должны быть упорядочены так, чтобы внешние слои зависели от более внутренних, а не наоборот.
- Взаимодействие с внешними компонентами(например с бд), должны просходить через абстракции(интерфейсы).



Чистая архитектура на Го



Вопросы