Лабораторная работа №2

Нечаев Игорь Сергеевич 334772

Оценочные расчеты

MAU = 50 000 000
DAU 50% от MAU = 25 000 000
Подписок в среднем 300 для fanout

Суточные и пиковые RPS

1) Средний пользователь вызывает в день:
публикация постов 0.1 раз пик 1 запрос

ленты 10 раз пик 50

написание комментариев 1 пик 5

чтение комментариев 5 пик 30

Количество в сутки
публикация постов 25 000 000 * 0.1 = 2 500 000 постов
2 500 000 / 86 400 = 30 RPS пик 300 RPS

запрос ленты 25 000 000 * 10 = 250 000 000
250 000 000 / 86 400 = 3000 пик 15 000 RPS

написание комментариев 25 000 000 * 5 = 125 000 000
1500 RPS пик по НФТ 10000 RPS

чтение комментариев 1500 пик 10000 RPS

Подписок в среднем 300
2 500 000 постов/день * 300 = 9000 RPS вставок в ленту
пик x10 -> 100 000 RPS

Объем хранения на 1 и 3 года

Посты
+-1000 символов плюс мета итого 1 кб

В день постов 2 500 000 -> в год постов 900 000 000 -> 1Tb
С учётом роста в 3 раза согласно заданию 3Tb

Комментарии
0.5 кб
В день 25 000 000 -> 10 000 000 000 в год -> 5 Tb
С учётом роста в 3 раза согласно заданию 15Tb
Индекс ленты
0.1 кб данных одна запись
Для 300 подписок в среднем
25 000 000 * 300 * 365 = 2737500 и 6 нулей -> 30 Tb
С учётом роста в 3 раза согласно заданию 90 Tb

Кеш
Лента активных 20% DAU 5 000 000
0.1 кб * +-100 постов -> 10 кб -> всего 50 Gb

Популярные посты
1% из всех 2 500 000, пост 1 кб -> не более 1Gb

Популярные пользователи
0.1% из всех 2 500 0000, профиль 10 кб -> не более 10 Gb

Всего на 3 года нужно 200 Tb с округлениями для данных

Для реплик, индексов, служебных данных умножаю на 4 и округляю до 1000 Tb

Бюджет на рост в 3 раза без архитектурных изменений

Горизонтально масштабирую Cassandra, Redis, Kafka
Добавляю партиции по времени

Рост линейный, сумма сильно зависит от деталей реализации

Модель данных (логическая)

Реляционный контур

Пользователи:

```
users (
   id           BIGSERIAL PK,
   username     VARCHAR(32)  NOT NULL UNIQUE,
   email        VARCHAR(255) NOT NULL UNIQUE,
   password_hash TEXT        NOT NULL,
   created_at   TIMESTAMPTZ  NOT NULL,
```

```
    is_celebrity BOOLEAN NOT NULL,
)
```

Посты

```
posts (
    id            BIGSERIAL PK,
    author_id     BIGINT      NOT NULL REFERENCES users(id),
    text          TEXT        NOT NULL,
    created_at    TIMESTAMPTZ NOT NULL,
    like_count        INT     NOT NULL DEFAULT 0,
    comment_count     INT     NOT NULL DEFAULT 0,
    share_count       INT     NOT NULL DEFAULT 0
)
INDEX idx_posts_author_created (author_id, created_at DESC)
```

Подписки

```
subscriptions (
    PRIMARY KEY (follower_id, followee_id),
    id  BIGINT NOT NULL REFERENCES users(id),
    fan_id  BIGINT NOT NULL REFERENCES users(id),
)
INDEX  idx_subscriptions_fan_id  (id, fan_id)
```

События для парадигмы event sourcing

```
outbox_events (
    event_id             BIGSERIAL PK,
    aggregate_type VARCHAR(32) NOT NULL,    -- 'post', 'comment', 'subscription'
    aggregate_id   BIGINT      NOT NULL,    -- id поста, комментария и т.п.
    aggregate_event_type     VARCHAR(64) NOT NULL,
    created_at     TIMESTAMPTZ NOT NULL,
    processed_at   TIMESTAMPTZ NULL
)
INDEX idx_outbox_unprocessed (processed_at, created_at)
```

Транзакционные границы:
Запись сущности + новый ивент в outbox_events

Хранилище ленты

Персонифицированный индекс, для хранения Cassandra

```
CREATE TABLE user_feed (
    user_id      BIGINT,
    created_at   TIMESTAMPTZ,
```

```
    post_id      BIGINT,
    PRIMARY KEY (user_id, created_at, post_id)
) WITH CLUSTERING ORDER BY (created_at DESC, post_id DESC);
```

partition key: user_id, clustering key: (created_at, post_id)

Когда автор создает новый пост: записываем пост в posts + событие -> Outbox Dispatcher отправляет событие в Kafka -> один из Feed Worker каждому подписчику добавляет запись в user_feed. Если пост от звезды он идет из только в user_feed автора, которая будет сливаться с user_feed подписчика fan out on read

Комментарии: схема под запрос «N последних по посту, сортировка по времени»

Дублирующий SQL контур NoSQL:
SQL comments -> SQL events -> Kafka -> Comments Worker

```
comments_by_post (
    post_id BIGINT,
    created_at TIMESTAMPTZ,
    comment_id BIGINT,
    user_id BIGINT,
    text TEXT,
    PRIMARY KEY (post_id, created_at, comment_id)
) WITH CLUSTERING ORDER BY (created_at DESC, comment_id DESC);
```

GET /posts/1/comments?limit=20&cursor=2025-05-12T12:00:15Z|9876543

SELECT * FROM comments_by_post WHERE post_id = 1 AND (created_at, comment_id) < (2025-05-12T12:00:15Z, 9876543) ORDER BY created_at DESC, comment_id DESC LIMIT 20;

Опишите, что и где кэшируется для гидратации постов по ID.

Redis, собираю пост + автор + 5 первых комментов
Кеш ленты первые 10 -> Кеш поста -> Кеш профиля

Кеш ленты хранит ответ для 10 постов
feed:{userId}:{[post]}

Кеш поста
post:{userId}:{

author_id, text, created_at, like_count, comment_count, share_count, comments_json }
Кеш профиля
post:{userId}:{
username, is_celebrity
}

Технические решения (сравнение и выбор)

Стратегия формирования ленты

1. Fan-out on write

   При записи поста отправка post_id во все user feed подписчиков

   Чтение очень быстрое
   Запись долгая, при большом кол-ве подписчиков очень долгая

2. Fan-out on read
   При запросе ленты читаем всех авторов, их посты -> мерджим -> ответ

   Чтение очень долгое
   Запись быстрая

3. Hybrid
   Fan-out on write для тех у кого не более X подписчиков,
   Fan-out on read для авторов у которых более X подписчиков, X - 95 перцентиль количества подписчиков

   Компромисс, лучшее из Fan-out on write и Fan-out on read

Пагинация для infinite scroll

Offset vs cursor

Offset проблемы при вставках, фильтрации
Cursor лучший выбор, но требует быстро находить элемент по id

Использую Cursor пагинацию по created_at, при равенстве по post_id из user_feed

Outbox/Transactional messaging

Transactional нет гарантии, риск потери обработки события
Outbox гарантированно запишет event который воркер будет публиковать в Kafka.
Допустима поломка обработчиков, их масштабирование.

Идемпотентность и повторная доставка

At-least-once + идемпотентность -> нет повторной доставки
Event sourcing model + таблица processed_events

Гидратация

Cache aside
Локальный кеш Guava на 10 секунд (взрывные запросы)
Redis Cluster TLL постов 30 минут, профилей 12 часов
Service

Батч по 20 элементов из НФТ

Таймауты:
Redis 10ms
Database 60ms
Feed 30ms
Гидратация 60ms

p95 < 150 ms

Деградация:
Redis недоступен: backoff и rate limit к БД
Service недоступен: отдаем только кэш

Бэкап/восстановление и репликация для разных хранилищ

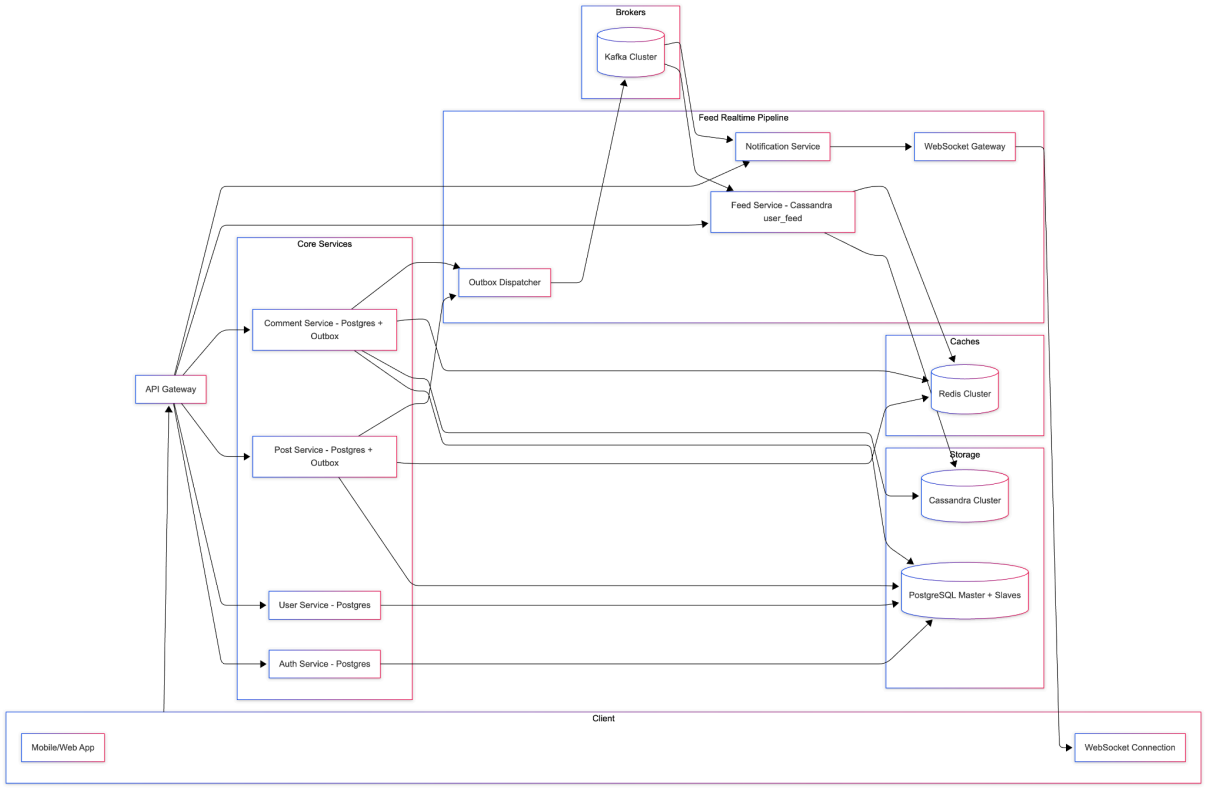
PostgreSQL Master + N slaves on read, бекапы в разных датацентрах на разных континентах
Cassandra/Kafka/Redis replication factor 3
При восстановлении warm up

Диаграммы

Component / Container

```
---
config:
 theme: mc
---
```

```
flowchart LR
subgraph Client["Client"]
       App["Mobile/Web App"]
       WSConn["WebSocket Connection"]
 end
subgraph CoreServices["Core Services"]
       AuthService["Auth Service - Postgres"]
       UserService["User Service - Postgres"]
       PostService["Post Service - Postgres + Outbox"]
       CommentService["Comment Service - Postgres + Outbox"]
 end
subgraph FeedPipeline["Feed Realtime Pipeline"]
       FeedService["Feed Service - Cassandra user_feed"]
       NotificationService["Notification Service"]
       OutboxDispatcher["Outbox Dispatcher"]
       WS["WebSocket Gateway"]
 end
subgraph Brokers["Brokers"]
       Kafka[("Kafka Cluster")]
 end
subgraph Caches["Caches"]
       Redis[("Redis Cluster")]
 end
subgraph Storage["Storage"]
       PostgresDB[("PostgreSQL Master + Slaves")]
       CassandraDB[("Cassandra  Cluster")]
 end
   Client --> APIGW["API Gateway"]
   APIGW --> AuthService & UserService & PostService & CommentService & FeedService
& NotificationService
   PostService --> PostgresDB & Redis & OutboxDispatcher
   CommentService --> PostgresDB & CassandraDB & Redis & OutboxDispatcher
   UserService --> PostgresDB
   AuthService --> PostgresDB
   FeedService --> CassandraDB & Redis
   OutboxDispatcher --> Kafka
   Kafka --> FeedService & NotificationService
   NotificationService --> WS
   WS --> WSConn
```

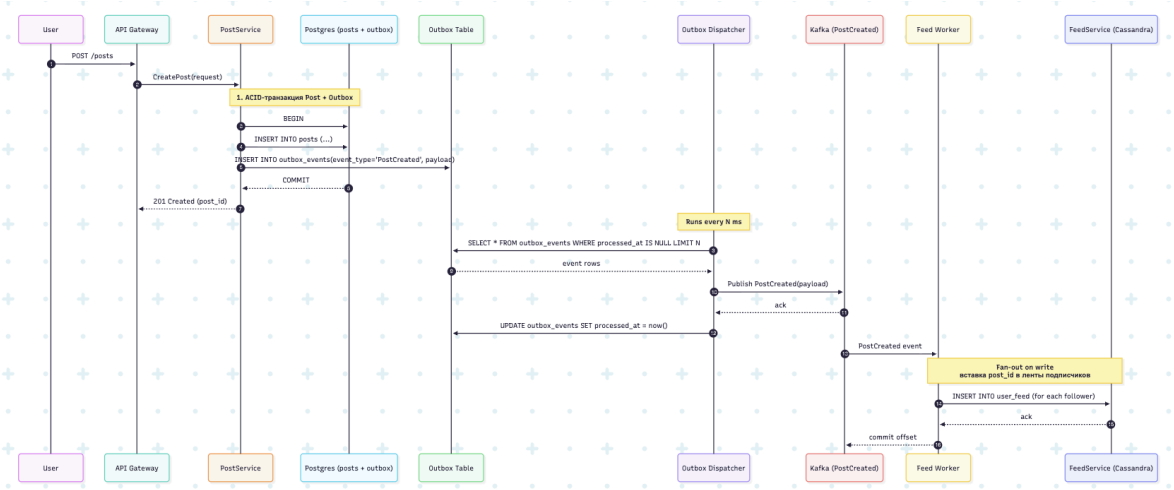


Sequence

публикация поста с Outbox и доставкой события

```
sequenceDiagram
    autonumber
    participant U as User
    participant API as API Gateway
    participant PS as PostService
    participant DB as Postgres (posts + outbox)
    participant OB as Outbox Table
    participant DIS as Outbox Dispatcher
    participant K as Kafka (PostCreated)
    participant FW as Feed Worker
    participant FS as FeedService (Cassandra)

    U->>API: POST /posts
    API->>PS: CreatePost(request)

    Note over PS,DB: 1. ACID-транзакция Post + Outbox
    PS->>DB: BEGIN
    PS->>DB: INSERT INTO posts (...)
    PS->>OB: INSERT INTO outbox_events(event_type='PostCreated', payload)
    DB-->>PS: COMMIT
    PS-->>API: 201 Created (post_id)

    Note over DIS: Runs every N ms
    DIS->>OB: SELECT * FROM outbox_events WHERE processed_at IS NULL LIMIT N
    OB-->>DIS: event rows

    DIS->>K: Publish PostCreated(payload)
    K-->>DIS: ack
```

```
    DIS->>OB: UPDATE outbox_events SET processed_at = now()


    K->>FW: PostCreated event

    Note over FW,FS: Fan-out on write<br/>вставка post_id в ленты подписчиков
    FW->>FS: INSERT INTO user_feed (for each follower)
    FS-->>FW: ack

    FW-->>K: commit offset
```



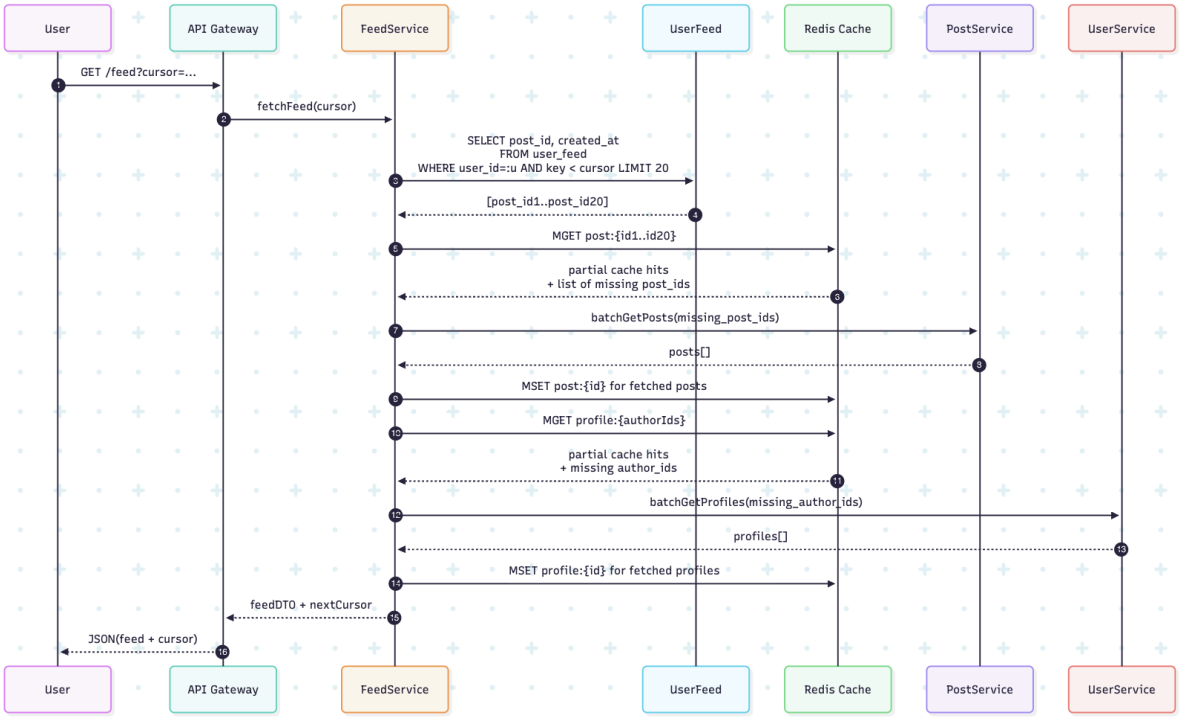
Чтение ленты с курсором и гидратацией

```
sequenceDiagram
    autonumber
    participant U as User
    participant API as API Gateway
    participant FS as FeedService
    participant USR as UserFeed
    participant RED as Redis Cache
    participant PS as PostService
    participant UPS as UserService

    U->>API: GET /feed?cursor=...
    API->>FS: fetchFeed(cursor)

    FS->>USR: SELECT post_id, created_at<br/>FROM user_feed<br/>WHERE user_id=:u AND key < cursor LIMIT 20
    USR-->>FS: [post_id1..post_id20]

    FS->>RED: MGET post:{id1..id20}
    RED-->>FS: partial cache hits<br/>+ list of missing post_ids

    FS->>PS: batchGetPosts(missing_post_ids)
    PS-->>FS: posts[]
    FS->>RED: MSET post:{id} for fetched posts
```

FS->>RED: MGET profile:{authorIds}
RED-->>FS: partial cache hits
+ missing author_ids

FS->>UPS: batchGetProfiles(missing_author_ids)
UPS-->>FS: profiles[]
FS->>RED: MSET profile:{id} for fetched profiles

FS-->>API: feedDTO + nextCursor
API-->>U: JSON(feed + cursor)



комментарий → событие → WebSocket-уведомление
sequenceDiagram
autonumber
participant U as User (Client)
participant API as API Gateway
participant CS as CommentService
participant DB as Postgres (comments + outbox)
participant OB as Outbox Events
participant DIS as Outbox Dispatcher
participant K as Kafka (CommentCreated)
participant NS as NotificationService
participant WS as WebSocket Gateway
participant Device as Телефон автора поста

U->>API: POST /posts/{id}/comments
API->>CS: addComment(request)

Note over CS,DB: 1. запись комментария + создание Outbox события
CS->>DB: BEGIN
CS->>DB: INSERT INTO comments(post_id, user_id, text, created_at)
CS->>OB: INSERT INTO outbox_events(type="CommentCreated", payload)
DB-->>CS: COMMIT

CS-->>API: 201 Created
API-->>U: OK (comment_id)

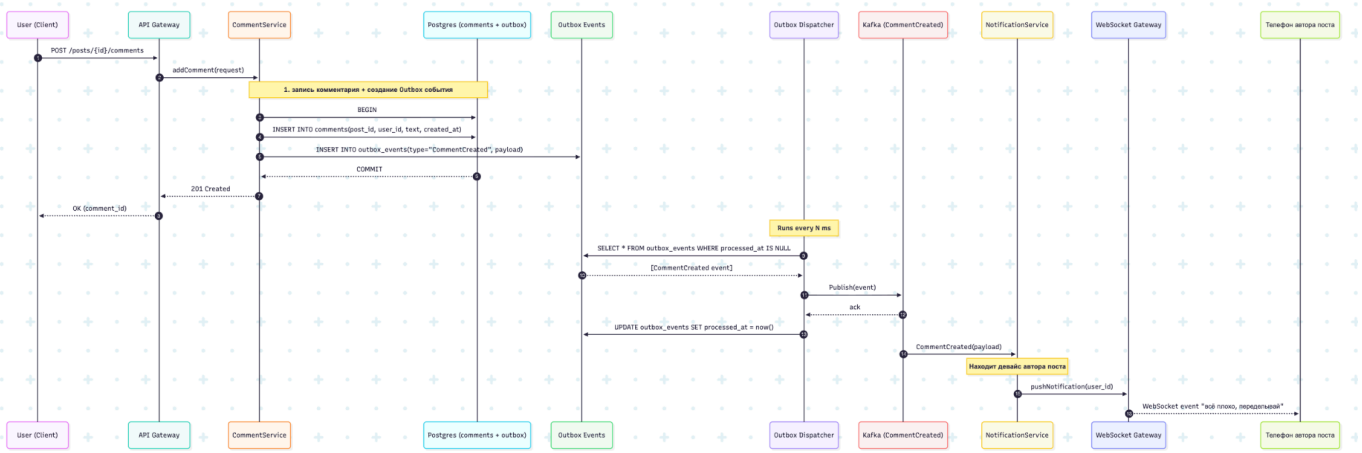
Note over DIS: Runs every N ms
DIS->>OB: SELECT * FROM outbox_events WHERE processed_at IS NULL
OB-->>DIS: [CommentCreated event]

DIS->>K: Publish(event)
K-->>DIS: ack
DIS->>OB: UPDATE outbox_events SET processed_at = now()

K->>NS: CommentCreated(payload)

Note over NS: Находит девайс автора поста
NS->>WS: pushNotification(user_id)

WS-->>Device: WebSocket event "всё плохо, переделывай"



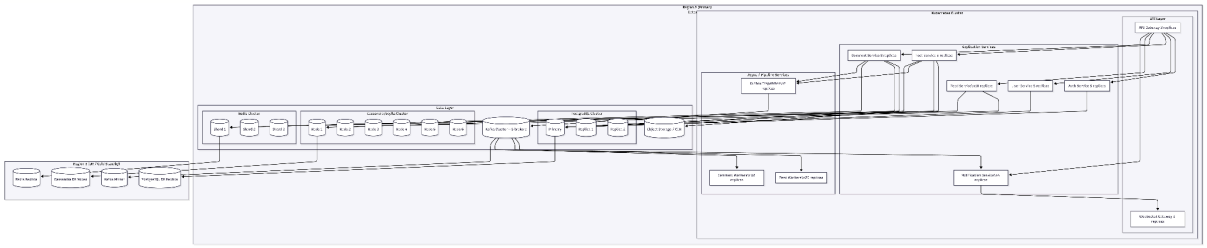
Deployment (узлы/контейнеры, масштабирование, внешние зависимости)

```
flowchart TB

    subgraph RegionA["Region A (Primary Datacenter)"]

        subgraph K8s["Kubernetes Cluster"]
```

```
    subgraph API_Tier["API Layer"]
        APIGW[API Gateway 3 replicas]
        WSGW[WebSocket Gateway 3 replicas]
    end

    subgraph App_Services["Application Services"]
        AuthSvc[Auth Service 5 replicas]
        UserSvc[User Service 5 replicas]
        PostSvc[Post Service 8 replicas]
        CommentSvc[Comment Service 8 replicas]
        FeedSvc[Feed Service\n10 replicas]
        NotifySvc[Notification Service\n4 replicas]
    end

    subgraph Infra_Services["Async / Pipeline Services"]
        OutboxDisp[Outbox Dispatcher\n2 replicas]
        FeedWorker[Feed Workers\n20 replicas]
        CommentWorker[Comment Workers\n12 replicas]
    end

end

subgraph Data["Data Layer"]

    subgraph PG["PostgreSQL Cluster"]
        PG_Primary[(Primary)]
        PG_Replica1[(Replica 1)]
        PG_Replica2[(Replica 2)]
    end

    subgraph CASS["Cassandra/Scylla Cluster"]
        Cass1[(Node 1)]
        Cass2[(Node 2)]
        Cass3[(Node 3)]
        Cass4[(Node 4)]
        Cass5[(Node 5)]
        Cass6[(Node 6)]
    end

    subgraph RedisCluster["Redis Cluster"]
        RedisSh1[(Shard 1)]
        RedisSh2[(Shard 2)]
        RedisSh3[(Shard 3)]
    end
```

```
        KafkaCluster[(Kafka Cluster — 5 brokers)]
        CDN[(Object Storage / CDN)]
    end

end


subgraph RegionB["Region B (DR / Cold Standby)"]
    PG_DR[(PostgreSQL DR Replica)]
    Cass_DR[(Cassandra DR Nodes)]
    Kafka_DR[(Kafka Mirror)]
    Redis_DR[(Redis Replica)]
end


APIGW --> AuthSvc
APIGW --> UserSvc
APIGW --> PostSvc
APIGW --> CommentSvc
APIGW --> FeedSvc
APIGW --> NotifySvc

NotifySvc --> WSGW

AuthSvc --> PG_Primary
UserSvc --> PG_Primary
PostSvc --> PG_Primary
CommentSvc --> PG_Primary

FeedSvc --> Cass1
CommentSvc --> Cass1

PostSvc --> RedisSh1
FeedSvc --> RedisSh1
CommentSvc --> RedisSh1

PostSvc --> OutboxDisp
CommentSvc --> OutboxDisp
OutboxDisp --> KafkaCluster

KafkaCluster --> FeedWorker
KafkaCluster --> CommentWorker
KafkaCluster --> NotifySvc

PostSvc --> CDN
```

```
PG_Primary --> PG_DR
Cass1 --> Cass_DR
KafkaCluster --> Kafka_DR
RedisSh1 --> Redis_DR
```



Class/Data (логическая модель для реляционной части + стереотипы/примечания для нереляционной).

```
classDiagram

    class User {
        +bigint id PK
        +string username UNIQUE
        +string display_name
        +string avatar_url
        +string email UNIQUE
        +timestamp created_at
        +timestamp updated_at
    }

    class Subscription {
        +bigint follower_id PK, FK -> User.id
        +bigint followee_id PK, FK -> User.id
        +timestamp created_at
        %% follower_id, followee_id = composite PK
    }

    class Post {
        +bigint id PK
        +bigint author_id FK -> User.id
        +string text
        +jsonb media
        +timestamp created_at
        +timestamp updated_at
        +int like_count
        +int comment_count
    }
```

```
class Comment {
    +bigint id PK
    +bigint post_id FK -> Post.id
    +bigint user_id FK -> User.id
    +string text
    +timestamp created_at
    +timestamp updated_at
    +bigint reply_to_comment_id
}

class OutboxEvent {
    +uuid event_id PK
    +string event_type
    +json payload
    +timestamp created_at
    +timestamp processed_at NULLABLE
}

class UserFeed {
    <<Cassandra>>
    +bigint user_id (PARTITION KEY)
    +timestamp created_at (CLUSTER KEY DESC)
    +bigint post_id (CLUSTER KEY DESC)
    +double rank_score
    +string source_type
    %% Sort order: newest first
}

class CommentsByPost {
    <<Cassandra>>
    +bigint post_id (PARTITION KEY)
    +timestamp created_at (CLUSTER KEY DESC)
    +bigint comment_id (CLUSTER KEY DESC)
    +bigint user_id
    +string text
    +bool is_deleted
    %% Query optimized: last N comments for post
}

User "1" --> "many" Post : author
User "1" --> "many" Comment : writes
User "many" --> "many" Subscription : follows

Post "1" --> "many" Comment : has
```

```
Post --> OutboxEvent : "post events"
Comment --> OutboxEvent : "comment events"

Post --> UserFeed : "fan-out events"
Comment --> CommentsByPost : "materialized view"
```