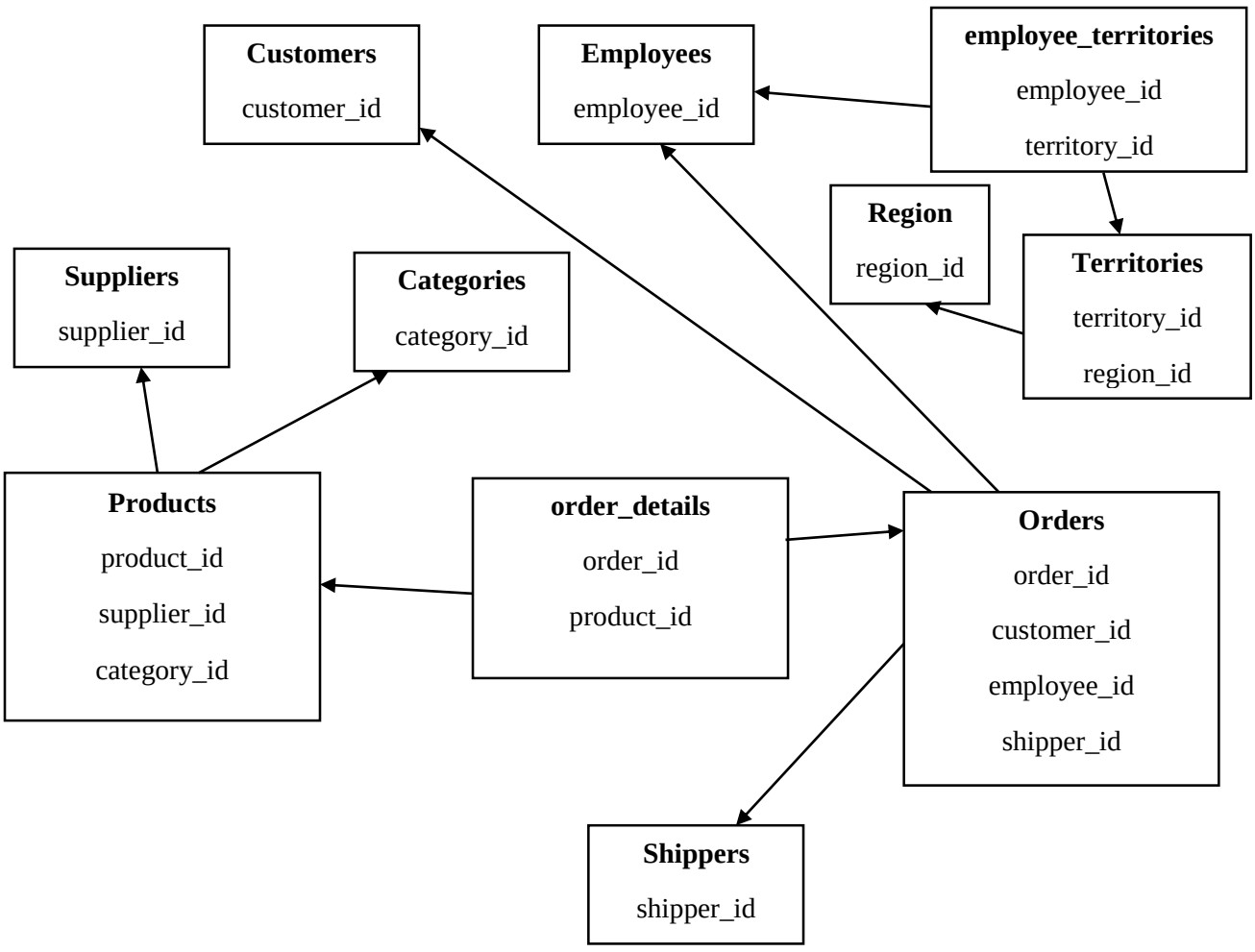
معماری Real-time Data Pipeline – Northwind

تعریف سناریو:

دیتابیسی داریم به نام Northwind که شامل اطلاعات کارمندان، مشتریان، کالا، تامین کنندگان، فروش و ... است.
فهرست جداول این دیتابیس به شرح ذیل می باشد:



دیتابیس northwind تحت Engine، postgresql می‌باشد.

مهمترین و چالشی‌ترین ارتباط بین جداول orders , orderdetails بوده که جداول والد و فرزندی هستند و بنوعی با تمامی جداول در ارتباط می‌باشند.

سناریو به این ترتیب است که می‌خواهیم به ازای انجام هر عملیات crud بصورت لحظه‌ای و realtime تغییرات از دیتابیس به دیتاورهوس انتقال داده شود. در جدول والد اگر سطحی درج شود، قطعا فرزند هم دارد یا ممکن است والدی صاحب یک فرزند باشد و فرزند جدیدی هم به آن اضافه شود.

منظور این ا ست اگر کارمندی- تامین کننده‌ای- م شتری- کالایی و ... ا ضافه/حذف/ویرایش شد تمامی تغییرات در لحظه سمت دیتاورهوس منعکس شود.

❖ نحوه لود دیتا incremental بوده و دو مدل پاکسازی دیتا داریم:

- تغییر data type: نوع داده‌ای بعضی‌ها تغییر کند؛
- new column ستون جدیدی اضافه شود؛
- روی یک سری از relationها لوکاپ داریم.

❖ وظایف و کارهایی که ما باید انجام دهیم به شرح ذیل است:

- ایجاد ETL Pipline روی با airflow؛
- رصد و انتقال تغییرات بصورت لحظه ای با kafka؛
- تغییر اینجین دیتاورهوس به clickhouse؛
- طراحی داشبورد با Grafana.

✓ برای پاکسازی داده می‌توان از spark یا pyspark استفاده کنیم.

✓ پایه همه تکنولوژی‌ها هم docker می‌باشد.

و نکته دیگر مساله این است که بتوانیم عکسی از کارمندان رو در datalake ذخیره کنیم و در طراحی داشبورد (با Grafana) بتوان میزان فروش هر کارمند و عملکرد فروش را به همراه تصویر پرسنلی‌شان نمایش داد.

✓ تصویر پرسنلی بایستی خارج از اسکوپ دیتابیس و در بستر datalake ذخیره گردد.

معماری پیشنهادی (شماتیک)

فهرست تحلیل معماری

۱. تحلیل نیازمندی‌های کسب‌وکار

۲. طراحی معماری کلی

۳. پیاده‌سازی لایه‌به‌لایه

۱. تحلیل نیازمندی‌های کسب‌وکار

۱,۱ نیازمندی‌های عملکردی

Real-time CDC: رصد لحظه‌ای تغییرات در PostgreSQL

Incremental Loading: لود افزایشی به جای Full Load

Data Cleansing: پاکسازی و استانداردسازی داده‌ها

Schema Evolution: مدیریت تغییرات اسکیمایی

Media Handling: ذخیره و بازیابی تصاویر خارج از دیتابیس

Dashboarding: نمایش لحظه‌ای در Grafana

۱,۲ نیازمندی‌های غیرعملکردی

Scalability: مقیاس‌پذیری افقی

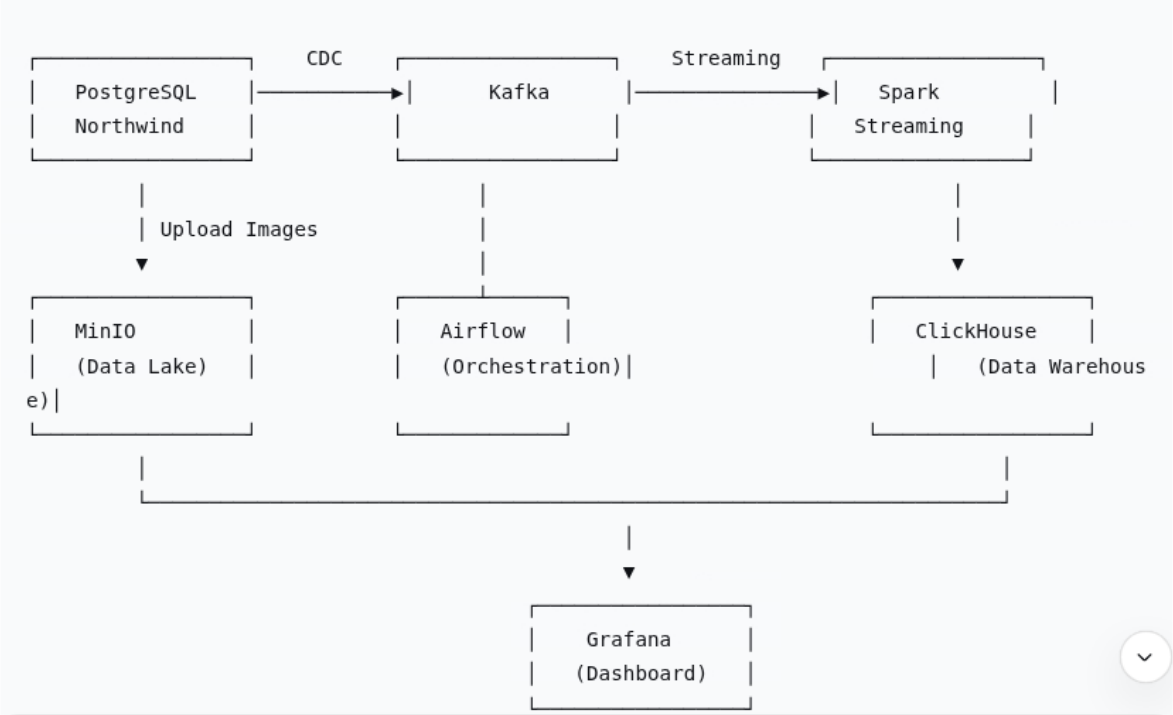
Fault Tolerance: تحمل خطا

Low Latency: تاخیر کم در انتقال داده

Maintainability: قابلیت نگهداری آسان

۲. طراحی معماری کلی

۲,۱ دیاگرام گردش داده



۲,۲ نقش هر کامپوننت

کامپوننت	نقش در معماری	تکنولوژی
Source	داده‌های تراکنشی	PostgreSQL
CDC	ضبط تغییرات	Debezium
Message Bus	انتقال رویدادها	Apache Kafka
Processing	پردازش جریان‌ای	Spark Streaming
Orchestration	هماهنگی	Apache Airflow
Storage	انباره داده	ClickHouse
Data Lake	ذخیره فایل‌ها	MinIO
Visualization	نمایش داده	Grafana

۳. پیاده‌سازی لایه‌به‌لایه

۳,۱ لایه منبع داده (PostgreSQL)

تحلیل ساختار داده‌ها:

sql

-- تحلیل روابط کلیدی

orders (Parent) ← order_details (Child)

employees ← orders(کارمند ثبت‌کننده سفارش)

customers ← orders(مشتری سفارش‌دهنده)

products ← order_details(محصولات سفارش)

استراتژی CDC:

- Logical Replication در PostgreSQL
- Write-Ahead Logs (WAL) برای ضبط تغییرات
- Replication Slots برای اطمینان از عدم از دست رفتن داده

۳,۲ لایه ضبط تغییرات (Debezium + Kafka)

مکانیزم کار Debezium:

تغییر در PG → WAL Log → Debezium Connector → Kafka Topic

پیکربندی Connector:

```
json
{
"  name": "northwind-cdc-connector",
"  config": {
"    connector.class": "io.debezium.connector.postgresql.PostgresConnector",
"    database.hostname": "postgres",
"    database.port": "5432",
"    database.user": "replication_user",
"    database.password": "password",
"    database.dbname": "northwind",
"    database.server.name": "northwind-cluster",
"    table.include.list": "public.*",
"    slot.name": "debezium_northwind",
"    publication.name": "dbz_publication",
"    plugin.name": "pgoutput",
"    tombstones.on.delete": "true",
"    transforms": "unwrap",
"    transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState"
}
}
```

۳,۳ لایه پردازش جریان داده‌ای (Spark Streaming)

الگوی پردازش:

```
python
class NorthwindDataProcessor:
"""
```

```
پردازش‌گر جریان داده‌های Northwind
"""
def process_order_stream(self):
"""
پردازش جریان سفارشات
"""
# ۱. استفاده از Kafka
# ۲. اعتبارسنجی داده‌ها
# ۳. پاکسازی و تبدیل
# ۴. غنی‌سازی داده‌ها
# ۵. نوشتن در ClickHouse
def handle_schema_evolution(self):
"""
مدیریت تغییرات اسکیمایی
"""
# - تشخیص ستون‌های جدید
# - تطبیق انواع داده‌ای
# - بروزرسانی اسکیماهای مقصد
```

۳,۴ لایه هماهنگی (Airflow)

طراحی DAGها:

```
northwind_pipeline/
├── initial_load_dag.py      # لود اولیه
├── cdc_monitoring_dag.py    # نظارت بر CDC
├── data_quality_dag.py      # کیفیت داده
└── maintenance_dag.py       # عملیات نگهداری
```

الگوی DAG لود اولیه:

python

```
with DAG('northwind_initial_load', schedule_interval=None) as dag:
        extract_task = PostgresOperator(
        task_id='extract_historical_data',
        sql='SELECT * FROM orders'
)
     transform_task = PySparkOperator(
        task_id='transform_data',
        application='transformations.py'
)
        load_task = ClickHouseOperator(
        task_id='load_to_clickhouse',
        sql='INSERT INTO orders ...'
)

    extract_task >> transform_task >> load_task
```

۳,۵ لایه انباره داده (ClickHouse)

استراتژی مدلسازی داده:

sql

```
--Wide Table برای عملکرد بهتر در queries
```

```
CREATE TABLE order_wide (
    order_id Int32,
    order_date Date,
    customer_id String,
    customer_name String,
    employee_id Int32,
    employee_name String,
    employee_photo_url String,
    product_id Int32,
    product_name String,
    category_name String,
    supplier_name String,
    unit_price Decimal(10,2),
quantity Int16,
    discount Decimal(10,2),
    total_amount Decimal(15,2),
    -- فیلدهای فنی
    _version UInt64,
    _updated_at DateTime,
    _source_ts_ms Int64
) ENGINE = ReplacingMergeTree(_updated_at, _version)
PARTITION BY toYYYYMM(order_date)
ORDER BY (order_id, product_id);
```

۳٫۶ لایه دریاچه داده (MinIO)

ساختار bucket:

```
employee-photos/
├── employee_1.jpg
├── employee_2.png
└── metadata/
    ├── employee_1_metadata.json
    └── employee_2_metadata.json
```

الگوی دسترسی:

python

class EmployeePhotoManager:

def upload_photo(self, employee_id: int, photo_file) -> str:

"""آپلود تصویر و بازگرداندن URL"""

def get_photo_url(self, employee_id: int) -> str:

"""دریافت URL تصویر"""

def update_photo_metadata(self, employee_id: int, metadata: dict):

"""بروزرسانی متادیتای تصویر"""

۳,۷ لایه نمایش (Grafana)

طراحی داشبوردها:

- Sales Performance: عملکرد فروش کارمندان
- Real-time Orders: سفارشات لحظه‌ای
- Product Analytics: تحلیل محصولات
- Data Quality: کیفیت داده‌ها

۳.۸ استراتژی مانیتورینگ

متریک‌های کلیدی:

- تاخیر انتها به انتها (End-to-End Latency)
- Throughput پیام‌ها در Kafka
- نرخ خطا در پردازش
- کیفیت داده (Data Quality)

هشدارها:

- تاخیر بیش از ۵ ثانیه
- توقف جریان داده
- خطاهای پردازش

۴. چالش‌ها و راه‌حل‌ها

چالش ۱: حفظ ترتیب رویدادها

راه‌حل: استفاده از کلید پیام در Kafka و پارتیشن‌بندی مناسب

چالش ۲: مدیریت تغییرات اسکیمایی

راه‌حل: الگوی Schema Registry و تطبیق پویا

چالش ۳: یکپارچه‌سازی تصاویر

راه‌حل: ذخیره در Data Lake و لینک‌دهی از طریق URL