

Návrh systémů IoT

5. Ukládání dat, databáze.

Stanislav Vítek

Katedra radioelektroniky

České vysoké učení technické v Praze

Stav v bezstavovém světě

- Často stojíme před úkolem, jak uchovat stav systému
 - HTTP i REST jsou koncipovány jako bezstavové
- Řešením může být databáze
- Někdy stačí uchovat stav v rámci tzv. relace - **server side session**

Session ve Flasku

- Session pracuje na jednoduchém principu - každý uživatel, který vstoupí na stránku obdrží jednoznačné číslo
- SessionID se typicky ukládá do Cookies, nebo se posílá jako součást URL

```
from flask_session import Session

app = Flask(__name__)
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)
```

- Session jsou ukládány na disku serveru (/tmp) nebo je možné uložit je do databáze

Typická aplikace - přihlašování

```
@app.route("/")
def index():
    # check if the users exist or not
    if not session.get("name"):
        # if not there in the session then redirect to the login page
        return redirect("/login")
    return render_template('index.html')

@app.route("/login", methods=["POST", "GET"])
def login():
    # if form is submitted
    if request.method == "POST":
        # record the user name
        session["name"] = request.form.get("name")
        # redirect to the main page
        return redirect("/")
    return render_template("login.html")
```

Databáze

- Databáze je místo, kam lze ukládat data a definovat operace, které s nimi lze provádět.
- Zdroje dat vhodné k ukládání
 - Logy webových/emailových serverů, routerů, ...
 - Sítě senzorů
 - Internet věcí
 - Stroje ovládané počítači

Např. jeden zámořský let Boeingu 777 vygeneruje odhadem 640TB dat

Data = hodnoty + časové značky

- Data jsou neměnná.
- Nová data = nový záznam.
- Čtení souvislé posloupnosti vzorků dat, např. pro generování grafu.
- Vysoce komprimovatelná data.
- Mazání obvykle ve velkém časovém úseku.
- Vysoká přesnost pro krátké časové období.
- Jednotlivé hodnoty nejsou tak důležité.

Typy databázových systémů

- Relační databázové systémy
 - Založeny na relačním modelu dat a relační algebře
 - Pro manipulaci s daty se nejčastěji využívá jazyk **SQL**
 - Structured Query Language
 - behaviorální popis problému
 - Zaručená konzistence dat
- NoSQL
 - Zaměření na výkon a rychlost zpracování
 - Vysoká škálovatelnost, schéma spravovaných dat se může měnit
 - Připouští dočasnou nekonzistenci

Relační databáze

- Základem relačních databází jsou databázové **tabulky**
- Tabulky jsou na sobě určitým způsobem závislé
 - existuje mezi nimi jistá logická vazba **relace**.
- Tabulky jsou též nazývány entitami
 - jsou chápány jako prvek reálného světa.
- Každá tabulka je tvořena **sloupci** a **řádky**
 - sloupce reprezentují vlastnosti této entity.

Relační databáze poprvé definoval v roce 1970 Edgar Codd (IBM Research Lab)

Záznamy v relační databázi

- Každý sloupec musí mít jedinečný název a určený datový typ
 - typicky **INTEGER, FLOAT, VARCHAR, TEXT, BLOB, DATETIME**
- Řádky tabulky reprezentují samotné záznamy (objekty) v databázové tabulce
- Každý řádek by měl mít svůj jedinečný identifikátor, podle kterého bude možné určit příslušný záznam - **klíč**.
 - Primární klíče slouží jako jednoznačný (unikátní) identifikátor záznamu (řádku)
 - Cizí klíče slouží k vyjádření vztahů (relací) mezi jednotlivými tabulkami

DML – příkazy pro manipulaci s daty

- **SELECT** – vybírá data z databáze.
- **INSERT** – vkládá data do databáze.
- **UPDATE** – edituje data v databázi.
- **DELETE** – odstraňuje data odpovídající podmínce.
- **CALL** – volá uložené procedury.
- **LOCK TABLE** – zamyká celé tabulky:
 - zámky čtení (READ)
 - zápisu (WRITE)

DDL - příkazy pro definici databázových struktur

- **CREATE** - vytváří objekty databáze (i databázi samotnou).
- **ALTER** - mění strukturu databázových objektů.
- **DROP** - odstraňuje objekty databáze (tabulky, indexy, databáze).
- **TRUNCATE** - odstraní všechny záznamy z tabulky, přičemž zachová strukturu tabulky (objekt tabulky).
- **COMMENT** - umožní přidávat komentáře k objektům databáze.
- **RENAME** - změna názvu objektu databáze

DCL - příkazy pro řízení uživatelských práv

- **GRANT** – nastavuje uživatelská práva k tabulkám databáze.
- **REVOKE** – odebírá přístupová práva přidělená příkazem GRANT.

TCL - příkazy pro řízení transakcí

- **COMMIT** - potvrzení provedení transakce.
- **SAVEPOINT** - definuje záchytný bod transakce, ke kterému se lze během provádění transakce vrátit.
- **ROLLBACK** - ruší transakci a vrací se zpět ke stavu původnímu.
- **START TRANSACTION** - zahajuje blok transakce.
- **SET TRANSACTION** - umožňuje změnit level izolace transakce globálně nebo pro aktuální session.

Tabulky

Tabulka 1: postavy

id	name	home
1	John	Winterfell
2	Tyrion	Casterly Rock
3	Daenerys	Dragon Stone

```
CREATE TABLE characters
(
    id INT PRIMARY KEY,
    name VARCHAR (255),
    home VARCHAR (255)
);
```

Tabulky

Tabulka 2: epizody

episode	char_id	appeared
1	1	10
1	2	5
1	3	7
2	1	5
2	2	6
3	3	15

```
CREATE TABLE characters
(  
    episode INT,  
    char_id INT,  
    appeared INT  
);
```

SQL - vkládání, mazání a změna řádků

```
INSERT INTO characters  
VALUES (1, 'John', 'Winterfall');
```

```
INSERT INTO characters  
VALUES (2, 'Tyrion', 'Casterly Rock');
```

```
INSERT INTO characters  
VALUES (3, 'Daenerys', 'Dragon Stone');
```

```
DELETE FROM characters  
WHERE name = 'John' OR id = 3;
```

```
UPDATE characters SET  
home = 'Winterfell' WHERE id = 1
```


SQL - dotaz v jedné tabulce

```
SELECT id, name FROM characters WHERE id = 1;
```

id	name
1	John

```
SELECT * FROM characters WHERE id < 3;
```

id	name	home
1	John	Winterfell
2	Tyrion	Casterly Rock

SQL - dotaz ve více tabulkách, agregace

```
SELECT name, sum(appeared) as sum
FROM characters, appearance
WHERE id = character_id
GROUP BY id
```

name	sum
John	20
Tyrion	5
Daenerys	16

Transakce

Konzistence dat v databázi je zajištěna pomocí transakcí. U relačních databází se jedná o princip **ACID** - **Atomicity** (atomicita), **Consistency** (konzistence), **Isolation** (nezávislost), **Durability** (trvanlivost).

- **Atomicita** Transakce musí proběhnout vcelku. Vyskytne-li se nějaká chyba v průběhu transakce, pak je celá transakce zrušena.
- **Konzistence** Transakce převede databázi z jednoho konzistentního stavu do jiného. Dojde k zápisu pouze platných dat, které dodržují integritní omezení
- **Nezávislost** Je-li spuštěno víc transakcí v jednom okamžiku, tyto transakce se navzájem neovlivňují. To znamená, že provedením částečné změny v databázi v průběhu transakce se projeví až po dokončení transakce.
- **Trvanlivost** Dokončení úspěšné transakce jsou data uložena v databázi natrvalo.

Škálování

- Škálování je schopnost systému, sítě nebo procesu zvládnout narůstající množství operací nebo objemu dat.
- Při škálování databází existují dva přístupy:
 - **vertikální**
 - Přidání dalších zdrojů v rámci stejné logické jednotky (CPU, RAM, HDD)
 - **horizontální**
 - Přidání dalších logických jednotek zdrojů - distribuované systémy
 - Tradiční model vyvažování zátěže, základ cloudu
 - Schopnost distribuovat jak data, tak zatížení jednoduchých operací na mnoha serverech bez sdílení RAM nebo diskového prostoru.
 - Horizontální škálování je považováno za modernější přístup.

Index

- Umožňují rychlejší vyhledávání v tabulce a nalezení odpovídajícího objektu (řádku)
- Tabulka může obsahovat více klíčů
- Nevýhodou je zpomalení aktualizace a místo potřebné pro strukturu indexů.
- Existují čtyři základní typy indexů:
 - i. **Index** (sekundární), vytvářen ve sloupcích, kde se nenachází primární klíč. Slouží k efektivnímu hledání záznamů, které jsou jiné než primární klíč.
 - ii. **Primární** Tento index se vytváří ve sloupci, ve kterém se nachází primární klíč. V tomto sloupci je každá hodnota unikátní a žádná hodnota není neznámá.
 - iii. **Unikátní** Jedinečný index zajišťuje dostupnost pouze neopakujících se hodnot, a proto je každý řádek jedinečný.
 - iv. **Fulltextový** efektivní vyhledávání slov v řetězcových datech.

Architektura databázových systémů

- **Aplikační vrstva** nebo vrstva UI představuje rozhraní pro všechny uživatele
- **Logická vrstva** obsahuje základní funkce řídicího systému relační databáze (RDBMS). Tato část systému obsahuje celou řadu implementací specifických pro dodavatele. Vztahuje se ke konkrétnímu databázovému modelu a používá jeho konstrukční dotazovací a manipulační prostředek.
- **Datové soubory**, které uchovávají uživatelská data.
 - **Indexy** pro rychlý přístup k datovým položkám, které obsahují určité hodnoty.
 - **Statistické údaje** s informacemi o datech v databázi. Jsou používány dotazovacím procesorem k výběru efektivních způsobů provedení dotazu.
 - **Logy**, které se používají ke sledování provedených dotazů. Tyto data mohou být využita třeba při obnovení databáze v případě selhání systému.

Python DBAPI

- <https://peps.python.org/pep-0249/>
- Definuje nízkoúrovňový přístup k DB z Pythonu
- Podpora hlavních SQL databází: MySQL, PostgreSQL, Oracle, ..., SQLite
- Klíčový koncept - [Cursor](#)
 - poskytuje metody pro spouštění SQL dotazů a získávání dat
 - pokud nemá databáze přímou podporu kurzorů, je kurzor emulován

Příklad relačních databáze - SQLite

- Jednoduchá [databáze](#), která ukládá data v binární formě na lokální paměťové médium
- Podmnožina SQL jazyka
- Řádkový klient, podpora v řadě programovacích jazyků včetně C, C++, Python, ...
- Použití: uživatelský profil v prohlížeči Mozilla, lokální storage v Androidu, ...
- Celá řada GUI nástrojů: [SQLiteStudio](#), [SQLite Database Browser](#), [SQLite Viewer](#)

SQLite - vytvoření databáze

```
import sqlite3
from sqlite3 import Error

def create_connection(db_file):
    """ create a database connection to a SQLite database """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        print("SQLite version: ", sqlite3.version)
    except Error as e:
        print("Error: ", e)
    finally:
        if conn:
            conn.close()

if __name__ == '__main__':
    create_connection("iot.db")
```

SQLite - vytvoření tabulky

```
def db_exec(conn, sql):  
    """ execute a SQL command  
    :param conn: Connection object  
    :param sql: SQL statement  
    """  
    try:  
        c = conn.cursor()  
        c.execute(sql)  
    except Error as e:  
        print(e)  
  
sql_create_projects_table = """ CREATE TABLE IF NOT EXISTS values (  
    id INTEGER PRIMARY KEY,  
    timestamp DATETIME NOT NULL,  
    temperature REAL NOT NULL,  
    humidity REAL NOT NULL); """
```

SQL - dotazy nad daty

Vložení dat

```
insert into values (timestamp, temperature, humidity)
values (datetime(), 12.2, 65.5);
```

Získání dat

```
select * from values
```

Formátování dat

```
select
    strftime('%Y-%m-%d %H:%M:%S', datetime()),
    temperature,
    humidity
from values
```

SQL - vložení dat v Pythonu

```
def db_insert(conn, data):  
    """ create a new record  
    :param conn: the Connection object  
    :param data: tuple of data  
    :return: data from table  
    """  
  
    sql = """  
    insert into values (timestamp, temperature, humidity)  
    values (datetime(), ?, ?); """  
  
    cur = conn.cursor()  
    cur.execute(sql, data)  
    conn.commit()  
  
    return cur.lastrowid
```

SQL - získání dat v Pythonu

```
def db_fetch (conn):  
    """ fetch data from table  
    :param conn: the Connection object  
    :return:  
    """  
  
    cur = conn.cursor()  
    cur.execute("SELECT * FROM values")  
  
    rows = cur.fetchall()  
  
    for row in rows:  
        print(row)
```

Object-Relational Mapping (ORM)

- Manipulace s daty z databáze pomocí objektově orientovaného paradigmatu.
- Je to běžná OOP knihovna, která zapouzdřuje kód potřebný k manipulaci s daty, takže není nutné používat SQL; místo toho probíhá komunikace přímo s objektem ve stejném jazyce, který používáte.
- Kód využívající ORM může vypadat třeba takto:

```
book_list = BookTable.query(author="Linus")
```

- Příklady: [Tortoise ORM](#), [SQLAlchemy](#)

SQLAlchemy

- Výkonný a flexibilní SQL toolkit a ORM pro Python

SQLAlchemy Core

- Toolkit pro základní komunikaci s databází:
 - připojení k databázi,
 - sestavení SQL dotazů,
 - vykonání dotazů a další

SQLAlchemy ORM

- deklarativní mapování, definice datových typů, filtry a další

Připojení k databázi

```
from sqlalchemy import create_engine

db = create_engine("sqlite:///database.db", echo=True)

# Varianty pro jiné systémy, Oracle, MS SQL atd...
db = create_engine('postgresql://uživatel:heslo@localhost:5432/mojedata')
db = create_engine('mysql://uživatel:heslo@localhost/databáze')
```

- Nad instancí **Engine** je možné volat metody jako `connect()` nebo `execute()`
- Parametr `echo` umožňuje zobrazení všech výstupů včetně prováděných dotazů
- Připojení k databázi v je tzv. líné
 - k fyzickému spojení dojde až v okamžiku nutnosti

Deklarativní mapování

- Pro deklaraci dat slouží třída `declarative_base`

```
from sqlalchemy.orm import declarative_base
Base = declarative_base(db)
```

- Od třídy `Base` jsou pak odvozeny datové struktury

```
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(128))
```

- Vytvoření záznamu pro nového uživatele je pak vytvoření instance

```
new_user = User(name='Stanislav Vitek')
```

Základní práce s daty

- Vytvoření databáze

```
Base.metadata.create_all(db)
```

- Abychom mohli pracovat s daty na úrovni databáze, musíme vytvořit session

```
from sqlalchemy.orm import sessionmaker  
Session = sessionmaker(bind=db)
```

- Přidání dat

```
session = Session()  
session.add(new_user)  
session.commit()  
session.close()
```

NoSQL databáze

- Vyrůstající objem a propojitelnost dat
 - Ztráta předvídatelné architektury, nebo neznáme dopředu přesnou strukturu dat
 - ACID zbytečně restriktivní
 - Výpadek je normální a umíme na něj reagovat
-
- Kdy nepoužívat NoSQL
 - Potřebuji-li transakce
 - Chci mít pevnou strukturu (schéma)
 - Potřebuji spojovat tabulky

Rozdělení NoSql databází

- Dokumentové databáze (Document stored)
 - místo s řádkem pracují s dokumentem, [CouchDB](#), [MongoDB](#)
- Databáze klíč – hodnota (Key-value)
 - důraz na rychlost, [Memcached](#), [Riak](#), [Redis](#)
- Time Series / Streaming Database Management Systems
 - [InfluxDB](#)
 - <https://github.com/xephonhq/awesome-time-series-database>

<https://hostingdata.co.uk/nosql-database/>

Key-value databáze

- Znamé již od roku 1979 ([dbm](#), Ken Thompson)
- Asociativní pole nebo hashovací tabulka s ukládáním hodnot podle unikátního klíče
- Jmenné prostory pro oddělení dat
- Serializovaná data (protokoly Apache Thrift, Protocol Buffers, ...)
- Rychlé vytvoření, rychlost práce s daty
- Základní API - operace **GET, PUT, DELETE**

Kde použít key-value databázi

- Ukládání informace o session
 - Každá webová session má svoje vlastní unikátní **session_id**
 - Uživatelské profily, preference
 - Každý uživatel má unikátní **user_id** + preference (jazyk, časové pásmo, ...)
 - Nákupní košík
-

Kde spíš ne:

- Tam, kde jsou relace mezi daty
- Dotazy nad daty, transakce

Riak

- Struktura: Instance -> Bucket -> Objects
 - **Bucket** - kolekce objektů (logické uspořádání, jmenný prostor)
 - **Object** - pár klíč-hodnota
 - klíč - UNICODE řetězec, unikátní v rámci bucketu
 - hodnota - cokoliv, asociace s content-type (Header)
- Komunikace prostřednictvím HTTP API (cURL)

```
curl -i -X PUT
  -H 'Content-Type: text/plain'
  -d 'Ivan Trojan, 1964'
  http://localhost:8098/buckets/actors/keys/trojan

curl -i -X GET
  http://localhost:8098/buckets/actors/keys/trojan
```

Redis

1. Single-threaded - požadavky jsou v databázi zpracovávány v pořadí příchodu
2. Praktické datové typy - **List, Hash, Set, Sorted Set** a **HyperLogLog**
 - Redis je přímo navržena pro použití jako fronta, zásobník, sestavování TOP něčeho, průniky množin, tvorba komplexních struktur aj.
3. Message Broker a Keyspace Notifications
 - Publish/Subscribe, v kombinaci s funkcí Redis Keyspace Notifications lze sledovat změny v datech (synchronizací do pomalejších úložišť).
4. Pipelining - požadavky lze odeslat bez čekání na odpověď na každý samostatný požadavek.
5. Cache-like chování - lze nastavit dobu životnosti objektu

Dokumentové databáze

- Datový model dokument - datová struktura se samopopisným charakterem
- Dokumenty používají pro ukládání dat i pro komunikaci s klienty
- Relační i objektové mapování
- Vhodné pro úlohy, kde je vazba na reálné měnící se dokumenty
 - Logování událostí v systému
 - Management obsahu, blogovací platformy
 - Webová analytika

MongoDB

- Pro komunikaci se používá JSON
- Interně jsou data uložena v BSON
 - maximální velikost 16MB
 - větší soubory fragmentovány
- Dokumenty mají flexibilní schéma
 - Lze měnit za běhu, ve skutečnosti jsou dokumenty podobné
- Obtížný balancing
- Záznamu lze přiřadit **_id**, pokud není specifikováno, MongoDB přiřadí unikátní

MongoDB a Python

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydict = {"name": "John", "address": "Highway 37" }
x = mycol.insert_one(mydict)
print(x.inserted_id)

myquery = { "address": "Park Lane 38" }
mydoc = mycol.find(myquery)

myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }
mycol.update_one(myquery, newvalues)
```

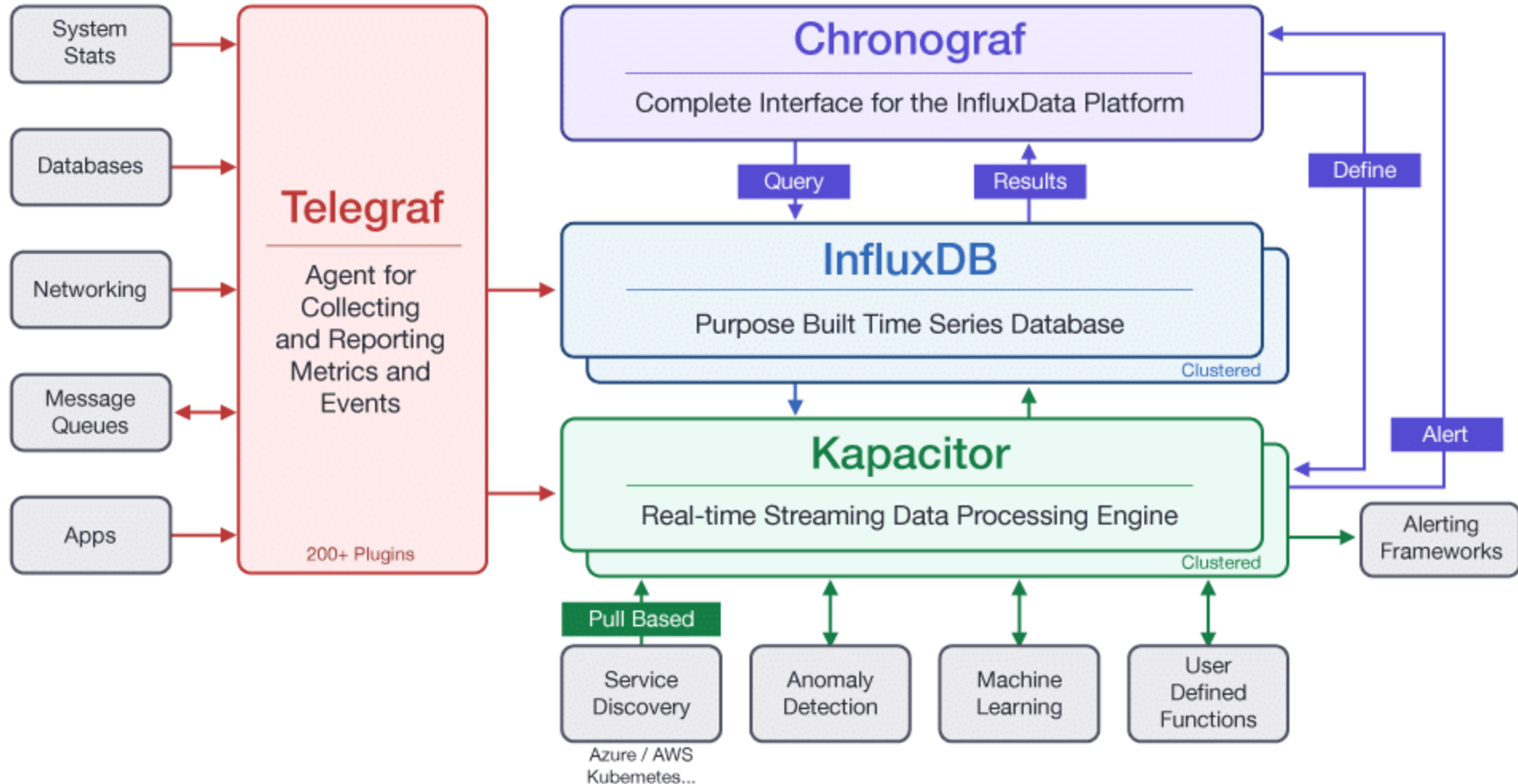
Databáze pro práci s časovými řadami

- Specializovaný DBMS optimalizovaný pro správu velkých objemů dat časových řad
- Optimalizované ukládání dat a sharding
- Podpora provozních funkcí (např. dotazy založené na rozsahu)
- Správa časové granularity
- Analýza a vytěžování časových řad

InfluxDB

- Open-source databáze časových řad (InfluxData)
- Napsáno v jazyce GO
- Dotazovací jazyk podobný SQL (InfluxQL) (<1.8)
- Dotazovací jazyk podobný JS (Flux) (>=1.8)
- Grafické rozhraní, rozhraní příkazového řádku (CLI) a rozhraní API HTTP
- Podpora distribuovaných nasazení
- Integrace s nástroji pro sběr dat v časových řadách, analytika a vizualizaci (např. Telegraf, Grafana).

InfluxDB - architektura



Telegraf

- Serverový agent řízený zásuvnými moduly pro shromažďování a vykazování metrik.
 - Pokud znáte cloudovou platformu, jako je AWS nebo Azure, jedná se v podstatě o Kinesis Data Streams nebo Stream Analytics.
- Telegraf má zásuvné moduly, které umožňují získávat různé metriky přímo ze systému, na kterém běží, stahovat metriky z rozhraní API třetích stran nebo dokonce používat statsd a Kafka.
- Telegraf má také výstupní pluginy pro odesílání metrik do různých dalších datových úložišť, služeb a front zpráv, jako jsou InfluxDB, Kafka, Graphite, OpenTSDB, Datadog a mnoho dalších.

Chronograf

- Chronograf je administrativní UI a vizualizační engine.
- Umožňuje
 - nastavení a údržbu monitorování
 - nastavení a správu alertů
 - správu databází
 - správu uživatelů a podporu organizací
- Je poměrně jednoduchý na používání a obsahuje šablony a knihovny, které umožňují rychlé vytvoření řídicího panelu s vizualizací dat v reálném čase a snadné vytváření pravidel pro upozorňování a automatizaci.

Kapacitor

- Kapacitor je nativní nástroj pro zpracování proudových dat.
- Dokáže zpracovávat proudová i dávková data z InfluxDB.
- Kapacitor umožňuje
 - zapojit vlastní logiku nebo uživatelem definovanou funkci pro zpracování výstrah s dynamickými prahovými hodnotami,
 - porovnávat metriky na základě vzorů,
 - vypočítávat statistické anomálie a provádět konkrétní akce na základě těchto výstrah, jako je dynamické vyrovňování zátěže.

Data management a typy dat

- Časově strukturovaný strom sloučení (TSM)
 - datová struktura, která se používá k uložení setříděných, komprimovaných datových řad.
- Index časových řad (TSI)
 - adresuje miliony jedinečných časových řad bez ohledu na množství paměti na hardwaru serveru.
- Postupy automatického snižování vzorků (downsampling) a pronájmu dat (retention).

Uložení dat - bod časové řady

- Timestamp - časové razítko ve formátu RFC3339 UTC (rrrr-mm-ddThh:mm:ssZ)
- Field keys - řetězcová metadata, podobně jako název sloupce
- Field values - aktuální změřená data
- Tag sets - dodatečné informace o měřeních
 - Tag keys - řetězcová metadata, podobná klíčům polí
 - Tag values řetězcové hodnoty
- Measurement - Kontejner měření pro uložení časových značek, polí a tagy (podobně jako tabulka v RDBMS)

Organizace dat

Buckets - kolekce datových bodů, která obsahuje:

- Measurement
- Tag sets
- Retention policy
 - DURATION - pravidla pro časové období, po které jsou datové body uchovávány v InfluxDB
 - REPLICATION - počet verzí, které by měly být uchovávány na clusteru

InfluxQL - základní příkazy

- CREATE DATABASE name
- USE DATABASE name
- SHOW DATABASES
- CLEAR DATABASE name
- CREATE RETENTION POLICY name ON measurement
- DURATION 1d REPLICATION 1
- INSERT treasures,captain_id=pirate_king value=2

FLUX - jazyk pro zkoumání dat

- Funkcionální přístup ke zkoumání a zpracování dat
- Inspirovaný jazykem Javascript
- Operátor pipe-forward (`|>`) umožňuje řetězit více dat. operací
- Každá datová operace manipuluje s tabulkou a vytváří novou tabulku jako výstup

```
from(bucket:"example-bucket")  
|> range(start:-1h)
```

FLUX - příklady

```
from(bucket:"example-bucket")
|> range(start:-1h)
|> r._measurement == "temperature" and r.room == "kitchen"
```

```
from(bucket:"example-bucket")
|> range(start:-1h)
|> r._measurement == "temperature" and r.room == "kitchen"
|> aggregateWindow(every: 1m, fn: mean)
```