Beyond Relational

Data storage for modern applications

Hey there!

I’m Mike Lehan

Software engineer, CTO at StuRents, skydiver, northerner

Follow me @m1ke

joind.in/talk/d4c76

Relational data

Where we, tend to, start

Why do we use relational databases?

- Widely available and understood
- Major frameworks and platforms built around them
- Tend to “do” most things we want them to

Or is it…

- Because it’s “the way we do it”?

> I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.
>
> *Abraham Maslow*

Examples

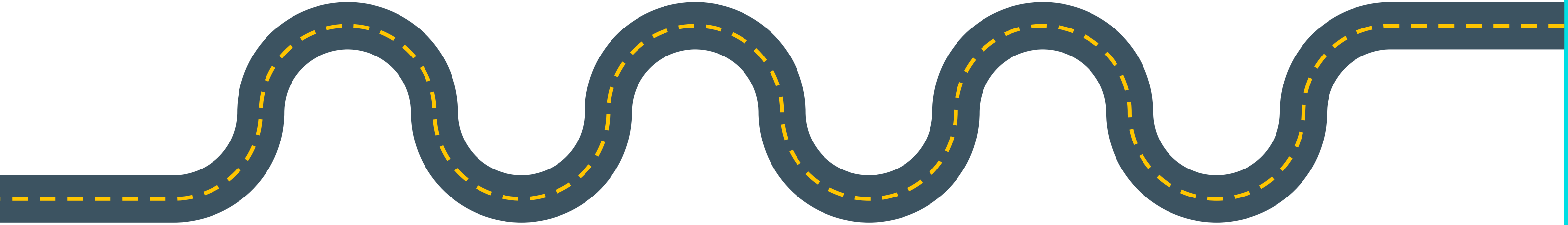
Let’s imagine some components of a real application

MegaSuper: The on-demand taxi app

Connects drivers and passengers via a mobile app

GPS data from every trip is stored for analysis and safety

Some days see magnitude increases in usage



Passengers can pay online via 3rd party processors

2x month-on-month traffic increases

Expansion into new areas of business, like delivery

Rapidly evolving datasets

New business goals, the need to store and analyse more

Unbalanced load

Growth is steady & rapid, hourly usage spikes dramatically

3rd party messaging

Payment confirmations, inbound SMS traffic is essential to running the business

Stop!

A few points to clarify

Don’t get me wrong

- Using a relational database is OK
- There’s no need to rebuild your whole application
- Many use-cases are best served by a relational DB

As engineers it’s important to understand a range of tools … also I’m quite into AWS, sorry if other clouds get less focus

Back to our totally made up taxi company...

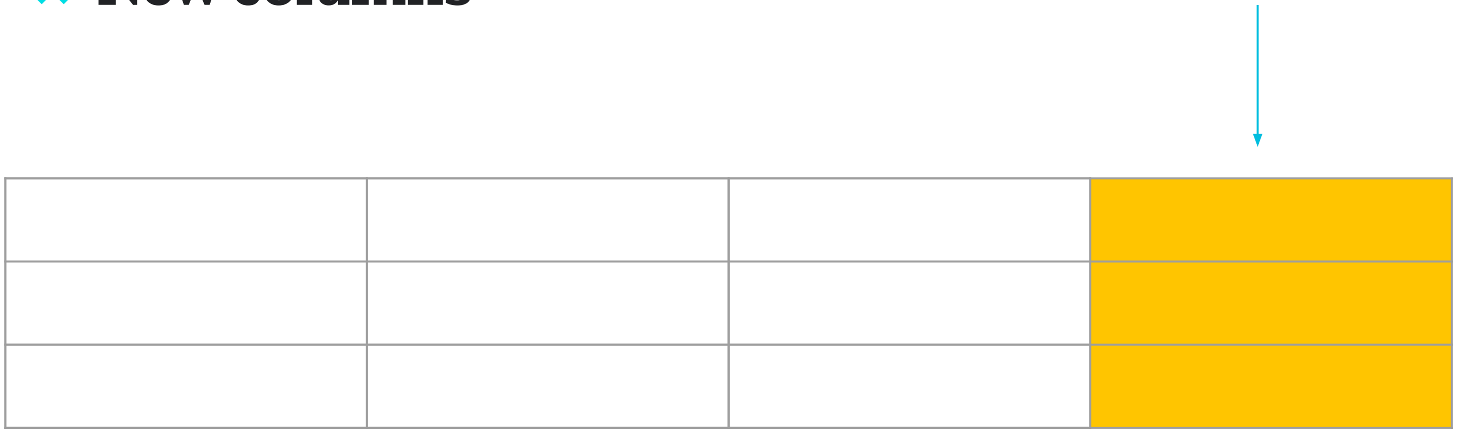
Rapidly evolving datasets

New business goals, the need to store and analyse more information

Changing business requirements

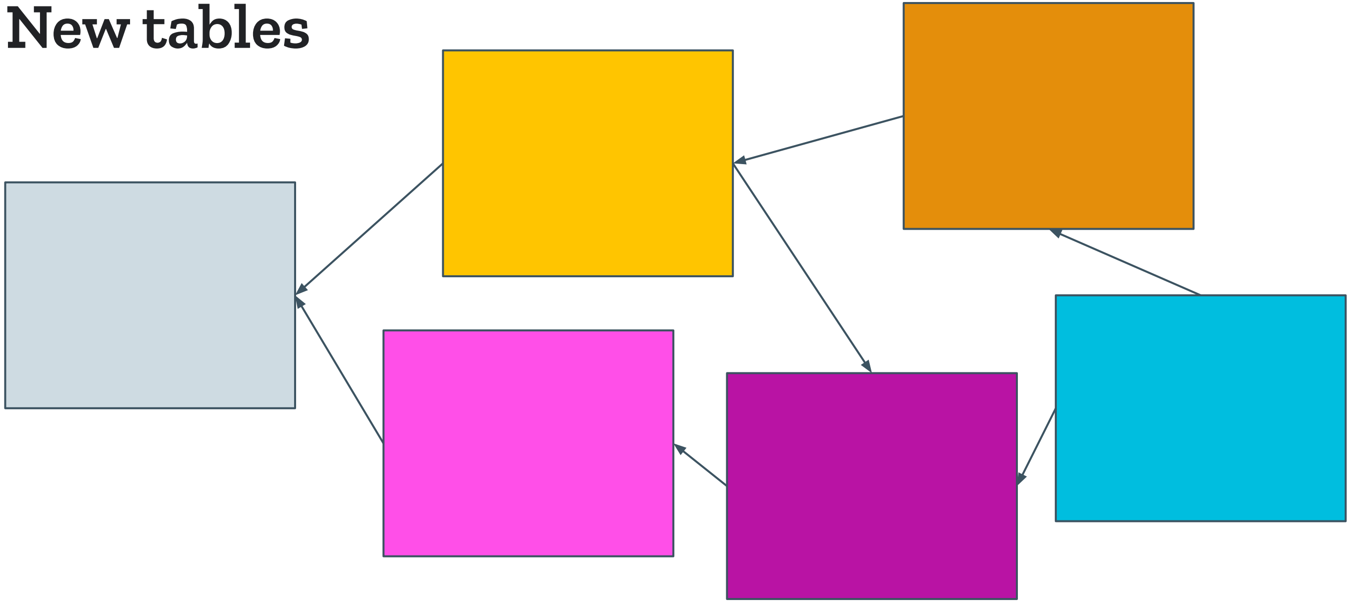
- Scope of entities expands - new fields needed
- We can represent these as new columns in existing tables
- Or we can add new tables and build out relationships

New columns



New tables

New tables



How about if we stored data in object terms

- Applications are based around objects interacting
- Joins can achieve expansive systems at the expense of heavy coupling and complex queries
- Structured data, (generally via JSON) is very popular in front-back web and API interactions

```
// driver
{
  “name”: “Alice”
  “ratings”: [
    {
      “stars”: 1
    },
    {
      “stars”: 2,
      “reason”: “speeding”
    },
    {
      “stars”: {
        “efficiency”: 3,
        “safety”: 4
      }
      “reason”: “”
    }
}
```

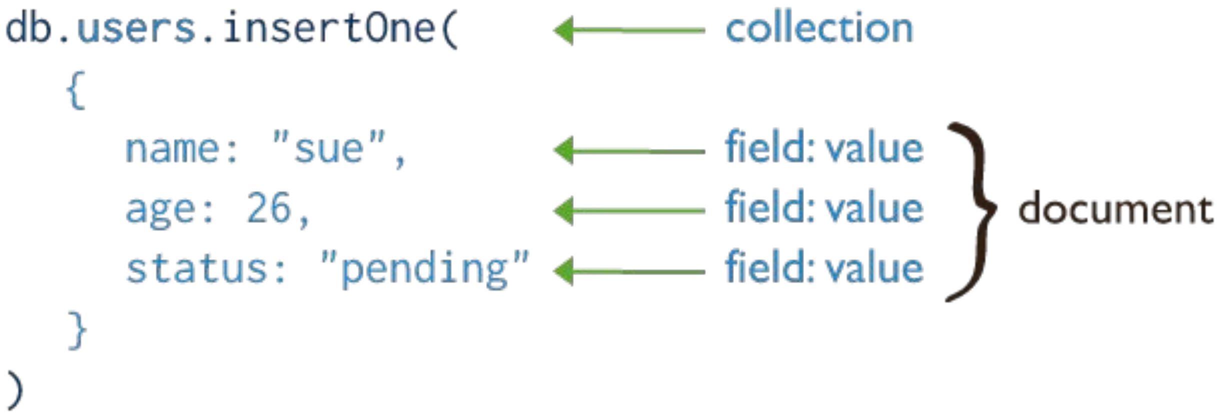
Document stores

Objects all the way down

MongoDB

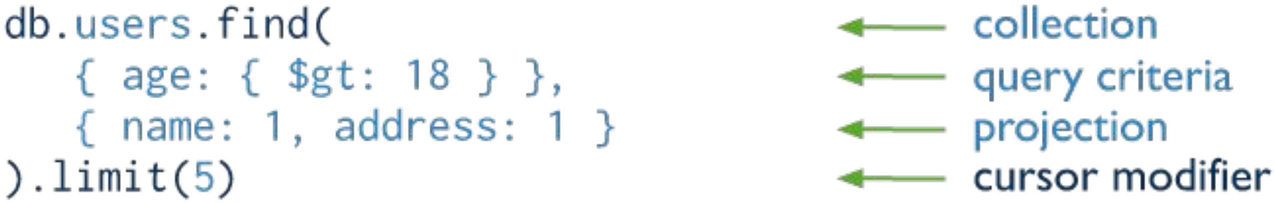
- Stores JSON documents in “collections”
- Supports indexing and transactions
- Can aggregate data or run internal JS functions
- Uses a programmatic API via language extensions for most operations

MongoDB



```
db.users.insertOne(
   {
      name: "sue",
      age: 26,
      status: "pending"
   }
)
```

MongoDB

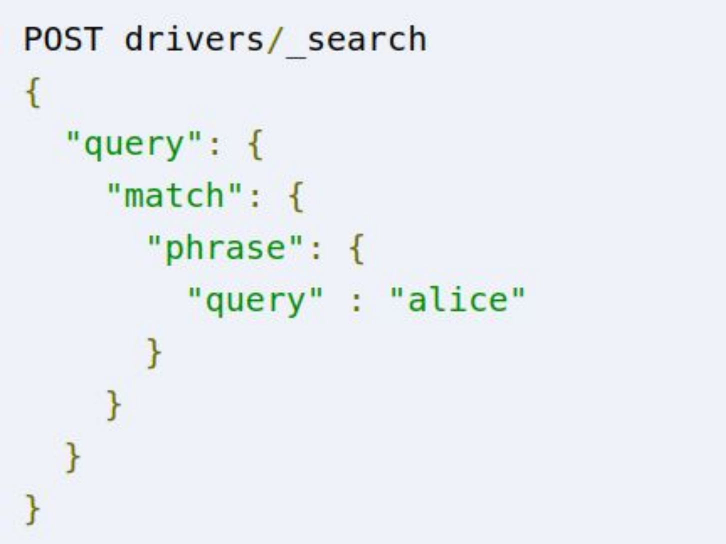


```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

Elasticsearch

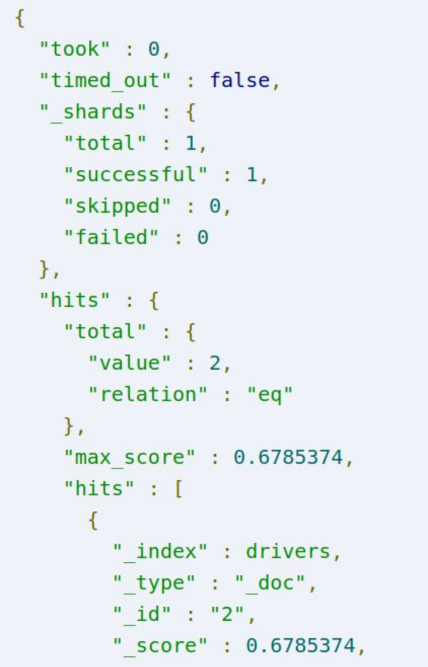
- Document storage adapted for full text search
- Accessed via HTTP API
- Integrates with Logstash and Kibana
- Provides fuzzy search with confidence values
- Works best with denormalized data

Elasticsearch



```
POST drivers/_search
{
  "query": {
    "match": {
      "phrase": {
        "query" : "alice"
      }
    }
  }
}
```

Elasticsearch



```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 0.6785374,
    "hits" : [
      {
        "_index" : drivers,
        "_type" : "_doc",
        "_id" : "2",
        "_score" : 0.6785374,
```

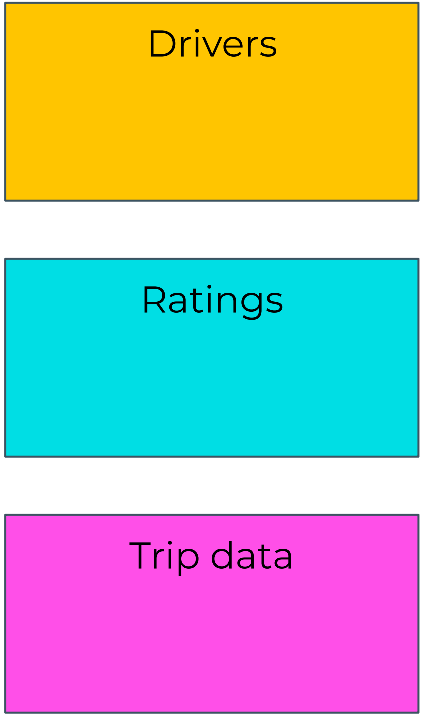
Unbalanced load

Growth is steady & rapid, hourly usage spikes dramatically

What do we mean by unbalanced?

- Unpredictable use of endpoints by users
- Some tables are utilised a magnitude more than others (for both reads & writes)
- Heavily written tables may be lightly read, and vice versa
- Complex queries can also cause extra read load

Writes



Drivers

Only written to when new drivers sign up to the platform

Ratings

Written to for most trips that happen, vulnerable to load spikes

Trip data

Written to constantly, usage increases exponentially with traffic

Reads

Drivers

Queried regularly as users request trips

Ratings

Read back to generate aggregate ratings for drivers to show to users

Trip data

Read rarely by users, large reads by staff for data analysis

Conventional ways to solve these

- Read replicas - increased cost & infrastructure management, only solves for reads
- Application changes to alter load profile - time consuming, may not be possible

Caching!

Key-value stores

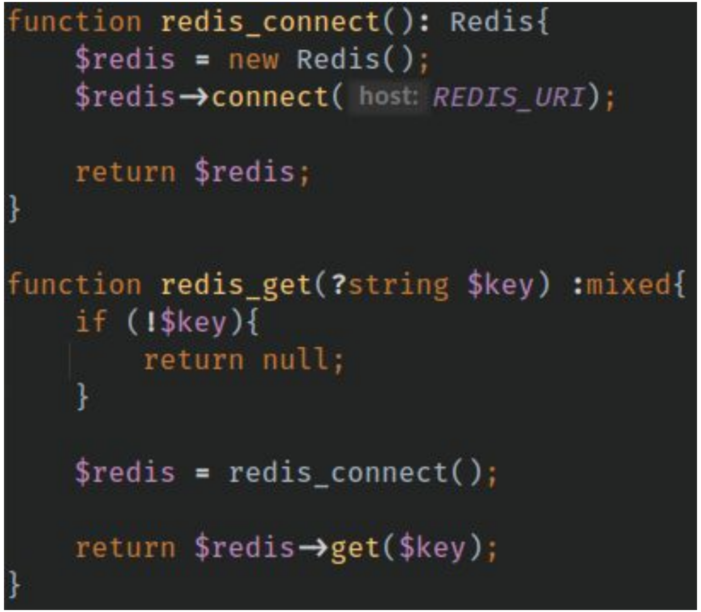
(yes they’re a type of database)

Primary features of a key-value store

- A single key corresponds to a single record - generally very fast lookups
- Can store different types of data under each key - no single schema to consider
- Most document stores are implemented on top of key-value concepts

Redis

- Commonly used as a cache
- May save cache to disk
- May act as a pub/sub message broker
- Cache expiry & access control
- Custom command-based API



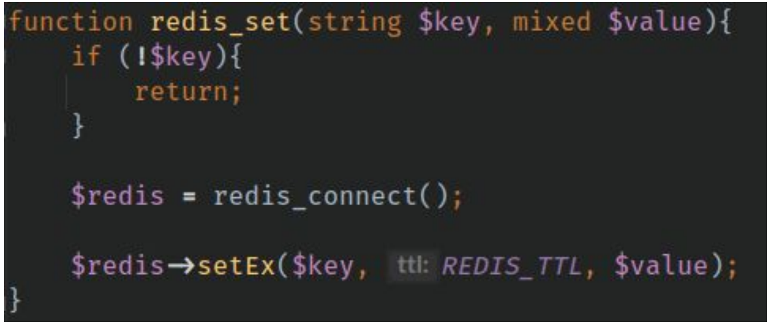
```php
function redis_connect(): Redis{
    $redis = new Redis();
    $redis->connect( host: REDIS_URI);

    return $redis;
}

function redis_get(?string $key) :mixed{
    if (!$key){
        return null;
    }

    $redis = redis_connect();

    return $redis->get($key);
}
```



```php
function redis_set(string $key, mixed $value){
    if (!$key){
        return;
    }

    $redis = redis_connect();

    $redis->setEx($key,  ttl: REDIS_TTL, $value);
}
```

Serverless!

Amazon DynamoDB

- Hybrid key-value & document store with no infrastructure to manage
- Similar to Redis, can expire rows via TTL fields
- Can use a stream to provide secondary processing

Powerful indexing abilities

- **Hash key** - acts like SQL “Group By”
- **Range key** - acts like SQL “Order By” (optional)

User	Datetime	Location
user@email.com	2021-06-17 12:00:00	Manchester
user@email.com	2021-06-18 13:00:00	Amsterdam

Powerful indexing abilities

- Global secondary indexes - create any time
- Same table, choice of any keys

User	Datetime	Location
user@email.com	2021-06-17 12:00:00	Manchester
user@email.com	2021-06-18 13:00:00	Amsterdam

Scalable, with no effort

- In “On-demand” mode, will scale from 0 to 3,000 requests per second with no throttling
- Can scale up to 40,000 read/writes per second given time or “provisioned throughput”
- In on-demand mode, reads (4KB) are priced at $0.3 per million, writes (1KB) at $1.4 per million

3rd party messaging

Payment confirmations, inbound SMS traffic is essential to running the business

Challenges of webhook processing

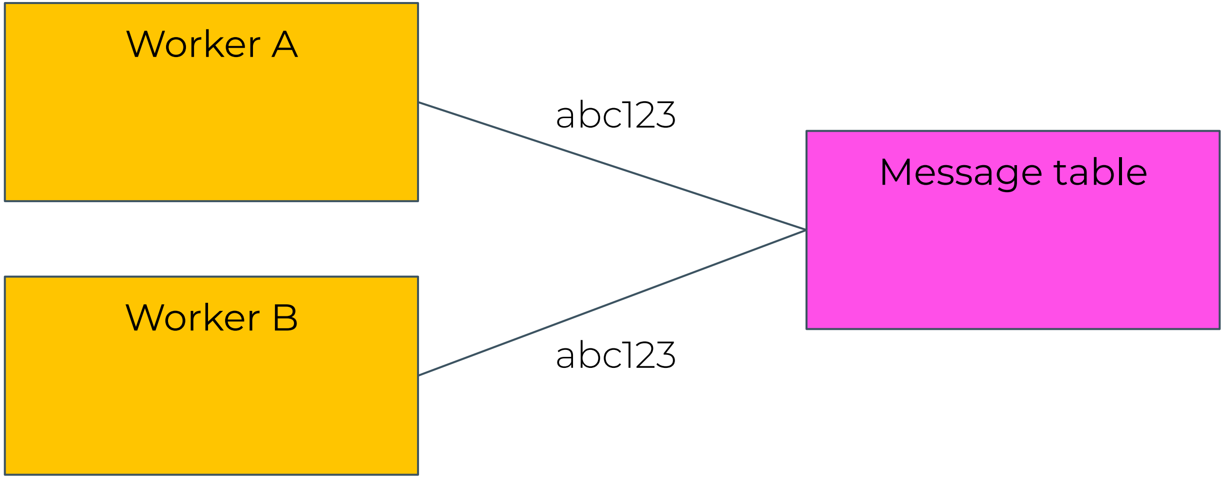
- Webhooks can inform us of important state changes within services we rely on
- Usually webhook sends are “dumb” - they may be retried if they fail
- Most services will not give more information if webhooks are not received properly

Handling this with our database

- HTTP Endpoint - stores incoming records to a database table and returns a 200 response
- Cron - Runs a CLI process at regular intervals, processing items from the table

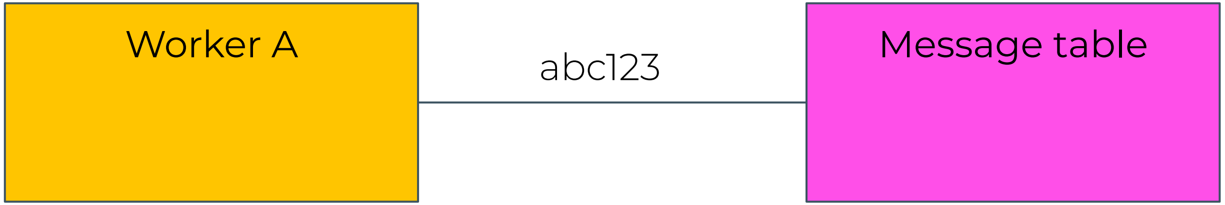
There are still some problems with this approach...

Parallel workers



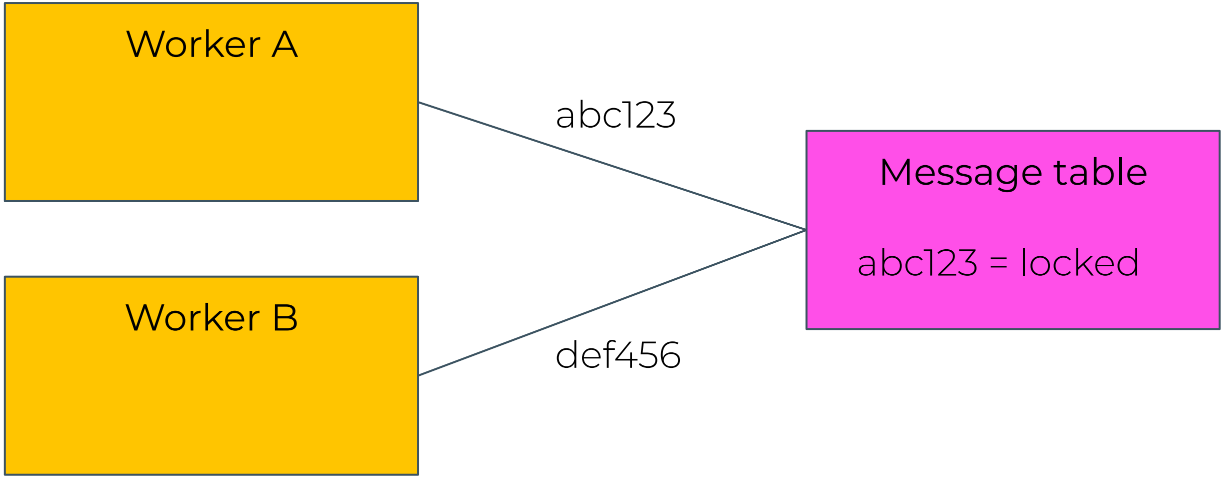
Risk of the same message being read (and therefore processed twice)

Single worker



Now our throughput is limited; if the table gets busier we can't increase speed of processing easily

Parallel workers - locking



Adds application complexity; need fallbacks to handle unreleased locks

Wait, there's more!

- Application processing now depends on database availability - busy database = slower processing
- Message delivery vulnerable to duplicates at network level
- Custom monitoring (via *more* database queries) needed to track inbound processes

Exactly once delivery

Observability

High throughput

Scalability

Queues

Getting your data in order

Primary features of a queue system

- Queues are communications based - rely on senders/receivers or producers/consumers
- Are not used for long term storage
- Simple data storage formats

RabbitMQ

- Hosted message broker
- Can push messages to consumers
- Routing can be set up to direct different messages to specific consumers
- Communicate via AMQP or just HTTP

Amazon Simple Queue Service

- HTTP queue, can integrate with other AWS products, including Lambda
- Has First In First Out mode to guarantee exactly-once, in-order delivery
- Dead-letter queue can handle failed processing

Addressing the challenges of our messaging system

Exactly once delivery

FIFO queues - visibility timeouts

Observability

Built-in metrics

High throughput

Process 3,000 m/s in standard mode

Scalability

Serverless means we tend not to worry

General principles

How do we decide on a data storage solution?

Size is a relevant metric

- In an application with low usage infrastructure choice may be less important
- By usage we can mean frequency of requests, amount of data stored, or both
- Becomes important with growth - if growth is rapid, time to implement may be short

My highly opinionated summary...

- Prefer tools with low infrastructure management or expertise required
- Vendor lock-in is generally an OK price to pay
- Systems are easy to migrate; data is generally not - pick services that offer flexibility
- Match your application to its data storage

The run down

Other data storage solutions, if we have time to talk about them

Graph databases

- Document stores; relations are first class citizens
- Relations are kind of like foreign keys on steroids
- Good for business cases where items may be related in N dimensions
- E.g. Neo4J, ArangoDB 🥑

Time-series databases

- Single purpose DB for storing time-based data
- Often optimised for high throughput, e.g. storing metrics from other systems, sensor readings etc.
- Fast calculations over millions of data-points
- E.g. InfluxDB

Amazon Quantum Ledger

- Document store with table semantics, query language & indexing
- Uses journal to track changes
- Cryptographically verifiable & immutable
- Can stream data to other services
- Proprietary and serverless

Thanks!

Any questions? Ask me on:

Twitter @M1ke

Slack #phpnw & #og-aws

joind.in/talk/d4c76

Presentation template by SlidesCarnival