Design for Retry

Microservices, REST, message busses and why idempotency is the only way to scale

Aria Stewart

@aredridel

I'm here thanks to PayPal

I'm going to talk about errors.

It's going to be okay.

We all know HTTP

2xx OK

3xx Go elsewhere

4xx Tell user what they did wrong

5xx Bail out and log an error

I'd call this

Error avoidance

You can't avoid errors

Here's the secret

Handle errors instead

4xx Tell the user what they did wrong

5xx Save that request and do something with it later.

Retry it

5xx are errors the *requestor* can handle

Causes!

- database down
- bug in a service
- Deploy in progress
- power failure
- kicked a cable
- Network congestion
- Capacity exceeded

- Tree fell on the data center
- earthquake
- tornado
- birds, snakes and aeroplanes
- Black Friday
- Slashdot effect
- Interns
- QA tests
- DoS attack

You need a queue

Lots of ways to do it

Database on the nodes

Log file

Queue server

gearman

Queues built in

There are many alternatives, but `gearmand` is very simple.

The memcache of job queues.

Three statuses:

- OK (Like 200)
- FAIL (Like 400)
- ERROR (Like 500)

design so ERROR can be retried.

`gearmand` automatically tries a job ERROR again.

And again.

And again.

If it isn't sure it worked?

Tries it again.

You *cannot* know if an error is a failure.

Error handling gets simpler

- Exception? ERROR.
- Database down? ERROR.
- Downstream service timeout? ERROR.

Maybe you retry right away.

How many of you have used a job queue?

You have used a job queue

Let me tell you about one

TRILLIONS of messages

MILLIONS of nodes

100% availability (at least partial) for years.

32 years.

Resilient to MILLIONS of bad actors.

It is attached to the most malicious network.

EMAIL.

250 OK

4xx RETRY

5xx Fail

Responsibility for messages

250 - accept responsibility

4xx - reject responsibility

5xx - return responsibility

reject responsibility.

If there's an error? Fail fast.

The requester can retry.

Fail fast.

Queue work you can't reject. Reject everything you can if there is an error.

You need a smart client.

Keeps outstanding requests.

Resubmit.

We're Nodevember, We're Special

Bonus content time!

Smart Clients on the device

Toto, we're not in AWS anymore.

Ever lose an email because you've been logged out?

Ever try to write email on the web while not on the Internet?

Some ideas

- Queue things in `localStorage`
- Use third-party storage
- Integrate third-party services with this approach

This is really good for offline-first design!

Being offline is the ultimate retriable error.