Taming a beast

Improving the Reliability of a Monolithic Web Service

Syed Humza Shah
Senior Software Engineer, Deliveroo
@shumzash, https://humza.sh/about
Scenario 1: Web clients reach the same service

Customers
Partners
Internal Applications

Web servers of the same kind
Scenario 1: Problems

- All web servers have the same resources
- CDN downtime affects all web clients
- Internal API services face needless latency
- Difficult root cause identification
- Specific faulty workloads can affect all clients
Recipe 1: Separate external/internal web clients

Customers → Reverse proxy

Partners → Reverse proxy

Reverse proxy → Internal Applications

Internal Applications → Web server(s) Type A

Internal Applications → Web server(s) Type B

Private Network
Recipe 1: Improvements

- Better resource utilisation
- Separate public/internal traffic
- Faulty workloads are scoped
- Reverse proxy gives us faster levers
Scenario 2: Job worker processes multiple queues

- High priority job queue
- Medium priority job queue
- Low priority job queue

Polling for jobs

Background job processor
Scenario 2: Problems

- Must have one polling frequency
- Uniform resource allocation
- Difficult root cause identification
Recipe 2: Separate process per job queue

Background job processor 1

High priority job queue → Polling for jobs

Background job processor 2

Medium priority job queue → Polling for jobs
Recipe 2: Improvements

- Better resource allocation
- Queue-specific polling frequency
- Easier root cause identification
- Easier to have separate datastore per queue
Scenario 3: One datastore and one credential set

Web worker 1

Background job processor 1

Data extractor

Datastore e.g. Relational database, NoSQL datastore, or Key-Value cache
Scenario 3: Problems

- Read spikes can affect write performance and vice versa
- Upgrades are limited
- Datastore logs with login user are difficult to interpret
- Faulty deployments can exhaust connections
Recipe 3: Poolers, replicas, and multiple users
Recipe 3: Improvements

- Read/write affect each other less
- Connection exhaustion less likely
- Easier tracing query source through datastore logs with username
- Worker-specific data access level
Recipe 4:  
Improve development workflows
Recipe 5: Improve "Mean Time to Detect" (MTTD)
Recipe 6: Actively know your vendor’s limits
Conclusion: You CAN tame the beast!
Questions?