Modern Banking in 1500-1600 Microservices

Matt Heath & Suhail Patel
$ curl https://api.monzo.com/branches

{"branches": [], "comment": "All our branches are on GitHub."}
Join the 4,000,000 people with a Monzo bank account
Hi everyone 🍀 I'm Monzo's Head of Engineering, and as I promised on Friday I'd like to share some more information about what happened during this outage. Because the nature of the issue was technical, this post is also quite technical. 😎

It's important to note that we had two major incidents last week that many of you will have experienced (sorry again.) The first incident lasted most of the week and affected only our prepaid product – ie. Monzo Alpha and Beta cards. The second outage affected both the prepaid product and our new current account for a period of around 1½ hours on Friday afternoon. This post is about the latter.

You can learn more about our overall backend architecture in this blog post 965 I published last year, but it's important to understand the role of a few components in our stack at a high level to understand this issue:

- **Kubernetes** 124 is a system which deploys and manages all of our infrastructure. Monzo's backend is written as several hundred microservices, packaged into Docker containers. Kubernetes manages these Docker containers and ensures they are running properly across our fleet of AWS nodes.

- **etcd** 152 is a distributed database used by Kubernetes to store information about which services are deployed, where they are running, and what state they're in. Kubernetes requires a stable connection to etcd in order to work properly, although if `etcd` does go down all of our services do continue running – they just can't be upgraded, or scaled up or down.

- **linkerd** 556 is a piece of software that we use to manage the communication between all of the services in our backend. In a system like ours, thousands of network calls are happening every second, and linkerd does the job of routing and load balancing all of these calls. In order to know where to route these calls, it relies on being able to receive updates about where services are located from Kubernetes.
TODAY

- Patisserie Valerie
  - Food
  - Breakfast
  - £26.80

THURSDAY 24 SEPTEMBER

- Waitrose
  - Groceries
  - £1.26

- Patisserie Valerie
  - Food
  - Breakfast
  - £6.45

- Boots
  - Drugs
  - Personal
  - £0.90

WEDNESDAY 23 SEPTEMBER

- Texture Restaurant Ltd
  - Food
  - Dinner
  - £211.00

TUESDAY 22 SEPTEMBER

- You've not used us in a while

⚠️ Please let us know if everything OK! Let us know.
Salary Sorter

£1,285.60
Current account balance

INCOMING PAYMENT
Salary payment £2,120.00

SORT YOUR MONEY
Spending £995
Bills £625
Savings £500

NEXT TIME?
Remember this for next time
We'll remember this sort the next time this payment comes in.

Next
Hey Emma 😊 I'm Stephen from Monzo. How may I help you today?

I was wondering if it's possible to use my card overseas?
https://www.youtube.com/watch?v=HHslPa1AVX8
A microservice
A microservice
Something You Know
Like a password or PIN number

Something You Have
Like a mobile phone

Something You Are
Like a fingerprint or facial recognition
Security Team
Authorization flow
Mobile App Security
Decisioning
Monitoring

Product Team
App Experience
Notifications

Payments Team
Mastercard
Faster Payments
Service
Service

Additional functionality
Split | Combine | Retire / Replace
$ shipper deploy --prod 34720 service.aws-billing

Use wearedev repository
Looking up PR 34720
#34720: Expose billing data as Prometheus metrics
Merged by evnsio
aws-billing-metrics → master

Ready to deploy service.aws-billing b2ea82e557cc → cb4b9a47ef32

→ To rollback, use:
shipper rollback --prod -r wearedev service.aws-billing

✅ Ready to deploy
→ service.aws-billing
  🛒 https://slog/?q=service%3A%20service.aws-billing
  📈 https://grafana/d/services?var-service=service.aws-billing

? deploy to prod?
syntax = "proto3";
package slackproto;

import "tools/protoc-gen-typhon/proto/typhon.proto";
import "tools/protoc-gen-validator/proto/validator.proto";

service slack {
  option (router).name = "service.slack";

  // Post a message to a channel
  rpc POSTMessage(PostRequest) returns (PostResponse) {
    option (handler).path = "/post";
  }

  // Returns message history for a given channel
  rpc ChannelHistory(ChannelHistoryRequest) returns (ChannelHistoryResponse) {
    option (handler).path = "/channel_history";
  }
}
$ ls -lah
lrwxr-xr-x@ 20 Jan 19:40 Makefile -> ../Makefile-service
-rw-r--r--@ 26 Nov 17:59 README.md
drwxr-xr-x@ 26 Nov 17:59 config
drwxr-xr-x@ 26 Nov 17:59 consumer
drwxr-xr-x@ 26 Nov 17:59 dao
drwxr-xr-x@ 26 Nov 17:59 domain
drwxr-xr-x@ 26 Nov 17:59 handler
-rw-r--r--@ 26 Nov 17:59 main.go
drwxr-xr-x@ 10 Feb 15:59 manifests
drwxr-xr-x@ 26 Nov 17:59 proto
$ svcquery
Learn things about services, based on your local git checkout.

Usage:
   svcquery [command]

Available Commands:
   analyze  Extract information from API and IAPI services.
   endpoints List the typhon endpoints of a service
   help      Help about any command
   history   Count the number of services and handlers at different points in time.
   owners    List the owners a service

Use "svcquery [command] --help" for more information about a command.
<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Level</th>
<th>Module</th>
<th>Message</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:31:37</td>
<td>DEBUG</td>
<td>service.user-context</td>
<td>[handler.handleGETList] Listing user contexts for user user_00009dcaydDRvjdsz01ULR</td>
<td>polyfill.go:22</td>
</tr>
<tr>
<td>09:31:37</td>
<td>DEBUG</td>
<td>service.user-context</td>
<td>[handler.polyfillUserContexts] Polyfilling contexts for user user_00009dcaydDRvjdsz01ULR, existing: [uc_00009oceGYGnr1dpUTYQhx]</td>
<td>polyfill.go:22</td>
</tr>
<tr>
<td>09:31:37</td>
<td>ERROR</td>
<td>service.account</td>
<td>[handler.listAccountsByPermissions] Error reading additional accounts: internal_service: gocql: no response received from cassandra within timeout period</td>
<td>list_by_permissions.go:137</td>
</tr>
<tr>
<td>09:31:37</td>
<td>TRACE</td>
<td>service.account</td>
<td>[errortracking.captureInSentry] Dropping event: internal_service: gocql: no response received from cassandra within timeout period</td>
<td>errortracking.go:99</td>
</tr>
<tr>
<td>09:31:37</td>
<td>DEBUG</td>
<td>service.user-context</td>
<td>[handler.handleGETList] Failed to polyfill missing contexts: internal_service: gocql: no response received from cassandra within timeout period</td>
<td>list.go:36</td>
</tr>
<tr>
<td>09:31:37</td>
<td>TRACE</td>
<td>service.user-context</td>
<td>[errortracking.captureInSentry] Dropping event: internal_service: gocql: no response received from cassandra within timeout period</td>
<td>errortracking.go:99</td>
</tr>
<tr>
<td>09:31:37</td>
<td>DEBUG</td>
<td>service.user-context</td>
<td>[handler.handleGETList] Listing user contexts for user user_00009dcaydDRvjdsz01ULR</td>
<td>polyfill.go:22</td>
</tr>
<tr>
<td>09:31:37</td>
<td>DEBUG</td>
<td>service.user-context</td>
<td>[handler.polyfillUserContexts] Polyfilling contexts for user user_00009dcaydDRvjdsz01ULR, existing: [uc_00009oceGYGnr1dpUTYQhx]</td>
<td>polyfill.go:22</td>
</tr>
<tr>
<td>09:31:37</td>
<td>ERROR</td>
<td>service.account</td>
<td>[handler.listAccountsByPermissions] Error reading additional accounts: internal_service: gocql: no response received from cassandra within timeout period</td>
<td>list_by_permissions.go:137</td>
</tr>
</tbody>
</table>
Backend Engineering 101

⚠️ It is important that you first carefully complete all the **backend onboarding** steps. This guide assumes this has been done. You will find yourself looking at obscure error messages if you don’t.

Welcome to Backend Engineering 101 🚀 This tutorial will walk you through creating your first backend services at Monzo. It will teach you how to create a new service, implement RPC handlers, query Cassandra (our database), publish and consume messages from the Firehose (our pub/sub abstraction), write unit tests and deploy your code.

How to use this tutorial 😊

Work through this tutorial at your own pace. As well as teaching you how to build backend services it will give you the opportunity to get comfortable with your development environment and the Go language (lots of us are new to Go when we join Monzo). Revisit the tutorial whenever you need to. The code you write here will serve as a useful example when you start picking up tasks in your team.

At the end of each chapter you’ll find links to more in-depth resources. You can jump into these immediately if you’d like some more context, or you can revisit them later. You don’t need to follow the links to complete the tutorial.

Don’t forget to take regular breaks. Grab a coffee, go for a walk and chat to your fellow Monzonauts 😊

Coffee!

Getting help 🌟

If you’ve got any questions you can post them in the #eng-onboarding channel on Slack. To make it easy for others to help, start your message with one of the following emoji:

!? I’ve got a question and it’s blocking me from moving on in the tutorial
!? I’ve got a question but it’s not blocking me
!? I’ve got a suggestion for improving the tutorial

You can also get help from your engineering buddy. They’ll be happy to pair up with you at any point 😊
INFO:   🕒 Waiting for config before serving...
INFO:   🔴 Admin server listening on 127.0.0.1:49545
INFO:   🔴 Config loaded (took 8.961748ms)
INFO:   🌟 Enabling prometheus registry
INFO:   👏 service.api.teapot listening on 127.0.0.1:49548
Improved Organisational Flexibility

Microservices which are granular and well understood

Consistent code structure and tooling
Focus on the problem

Standardising enables engineers to focus on the business problem

Continuously improving tools and abstractions
Increase velocity whilst reducing risk

Make and deploy a series of small and iterative changes
Break down complexity and reduce risks
Thanks!

Matt Heath (@mattheath)
Suhail Patel (@suhailpatel)