Building & trusting a cloud bank

@gmorpHEME #qconlondon

S T A R L I N G  B A N K
18th June 2012
we can rebuild the bank in an hour
we can rebuild the bank in an hour

job done
we can rebuild the bank in an hour

job done(*)

* terms and conditions apply
we could rebuild in an hour but...

• ...the problem might not be us
• ...there might be some phone calls
• ...only in AWS
• ...from recent backups
• ...only if we make the decision to do it
Starling Bank today

- Full UK Current Account
- iOS & Android Apps
- Debit Card
- Faster Payments & DDs
- Overdrafts

In-app Support
- Open API
- Closed Beta (100s Accs)
- Core Ledger
- Credit, AML, KYC, Fraud Processes
security > resilience > scale
velocity > economy
architect for change
embrace cloud
three key categories of failure

• errors that correlate by infrastructure
• errors that correlate by function
• errors that emerge in complex systems under load

• we’ll talk about the first two
immutable infrastructure
instance termination is safe

• single stateless service per instance
• if ever a server is in doubtful state, kill it
  • pen testing?
  • chaos experiments?
  • suspicious activity?
• chat-ops slack bot
  • starbot kill
• rolling deployments by termination (not quick but safe)
  • starbot recycle
  • starbot reboot
...everywhere

• everything in our core infrastructure is either
  • immutable service in EC2
  • data in a managed service

• no large infrastructure pet
  • no “clusters”
  • no state in EC2
  • no EBS volumes to manage
  • no shared caches
  • no external queues
  • no orchestration engines
  • (yet!)
a Starling service

- simple AWS approach
- ELB / ASG / RDS across 3 AZs in eu-west-1
- "service discovery" is just DNS
- service is docker as systemd unit on CoreOS
- all specified in CloudFormation (!)
- with config and versions in S3
impact of instance outage

- 2x (5s interval + 2s timeout) = max 14s to drop out of ELB
- Some 504s then 5m of reduced capacity
- 14s when 1 of our ~10 services is partially degraded
at least one bank has an iOS app that takes ~14s to start
production chaos

• we know we're resilient because we kill servers all the time
importance of noise-free steady state

errors

errors

(No customer impact)

chaos

rolling release
artificial load in production

- monitoring and control are difficult without volume
- we deploy a “simulator” service in production
- generates synthetic transactions
- e.g. 160,000 card authorisations a day
- continual assurance on available headroom
- interruptions are obvious
- all servers are naturally warmed up
- synthetic transactions are difficult
impact of AZ / region outage

• AZ loss => ASGs and ELBs rebalance
• region loss (EC2/RDS) => rebuild

• S3 outage
  • lose some message archiving
  • new instances fail retrieving config (easy fix)
self-contained systems

http://scs-architecture.org
Starling as self-contained systems

- all services have their own RDS instance
- inter-service comms is generally async
- mobile layer integrates data from different services
- no start-up order dependencies
not pure SCS

• we're mobile-first (and API-first!) – web is secondary
• services not owned by single team
• our services have REST APIs but no internal web UI
  • internal (inter-service)
  • external (mobile)
  • management (web console)
  • operations (health check etc.)
• one key area with sync interaction (balance allocation)
each service exposes distinct APIs
testing service loss

- starbot chat-ops exposes
  - starbot kill
  - starbot kill all
- available to all developers

greghawkins 8:27 PM 🌟
starbot kill all calendar in demo

starbot 🧸 8:27 PM
Forcefully terminating every single calendar instance in demo.
Underway... keep an eye on https://dashboard.status-dashboard.json and https://dashboard/versions-dashboard.json

(LOTS OF AUTONOMOUS SERVICES CONTINUALLY TRYING TO DO IDEMPOTENT THINGS TO EACH OTHER)
DITTO architecture

(Do idempotent things to others)
DITTO architecture

• async + idempotence + retry
  • async: 202 Accepted (once written to store)
  • idempotence: create with PUT
  • retry: accept and store (or 400) then work from database

• each service constantly working towards correctness
• often achieve idempotence by immutability
  • subsequent requests match previous or fail
  • reflects append-only approach to data

• no distributed transactions
POST 201 Created {uuid}

Make a payment

PUT {uuid}

202 Accepted

PUT {uuid}

202 Accepted
POST 201 Created {uuid}

PUT {uuid}

PUT {uuid}

202 Accepted

PUT {uuid}

202 Accepted

Retry provides "at least once" Idempotence = "at most once"
234,000
CATCH-UP PROCESSORS WITHOUT BACK-OFF

Exception logs/m

- card
- customer

Graph showing exception logs per minute for card and customer.
cherish your bad data
impact of service outage

• UI degrades gracefully
• back-end work delayed
  • payments
  • card creation
  • ledger postings
  • interest accrual
• but real disruption: card auth & ATM usage
continuous delivery
you can do anything you can undo
continuous delivery of back-end

• continual deployment to non-prod, sign-off into prod
• auto build, dockerise, test, scan, deploy < 30m
• in first 221 days of production environment
  • 134 releases of software (~1 per weekday)
  • 70 releases of infrastructure (~1 per 2 weekdays)
summary

• SCS + immutable infra + CD
• infrastructure failure absorbed
• failure of function isolated and tolerated
  • UI degrades gracefully
  • items “buffered” and retried
  • fixed safely and swiftly
• this year
  • ++services, scale!, k8s, ML/data
Careers: starlingbank.com/careers
Hackathon: starlingbank.com/hackathon

Sushi image credit: www.vecteezy.com/vector-art/92795-sushi-platter-vectors