ARC309

From Monolithic to Microservices

Evolving Architecture Patterns in the Cloud

Adrian Trenaman
SVP Engineering, gilt.com
@adrian_trenaman

Derek Chiles
Sr. Mgr, Solutions Architecture, AWS
@derekchiles

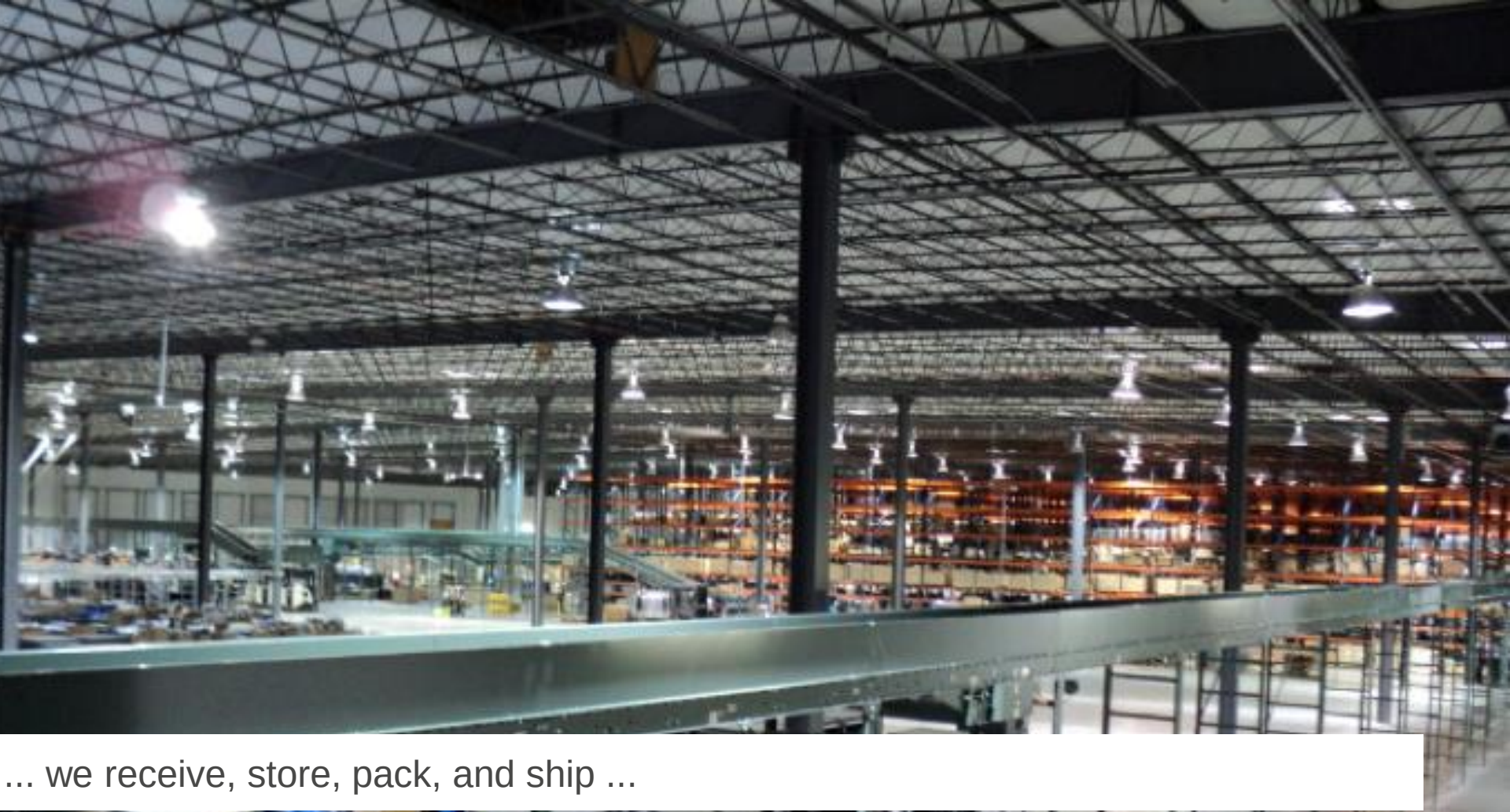
October 2015

amazon web services

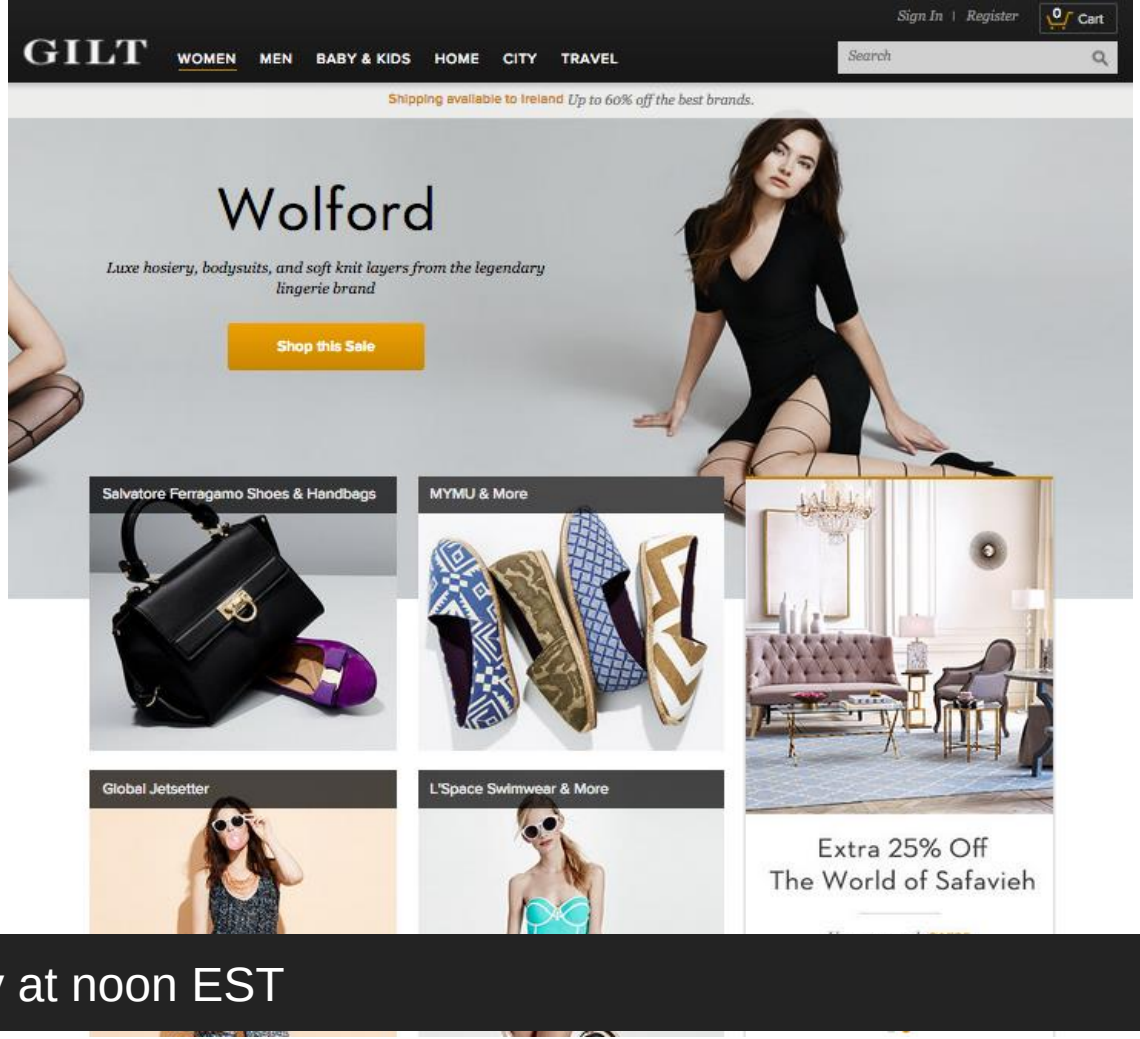
Gilt’s Journey

Gilt: Luxury designer brands at members-only prices

... we shoot the product in our studios

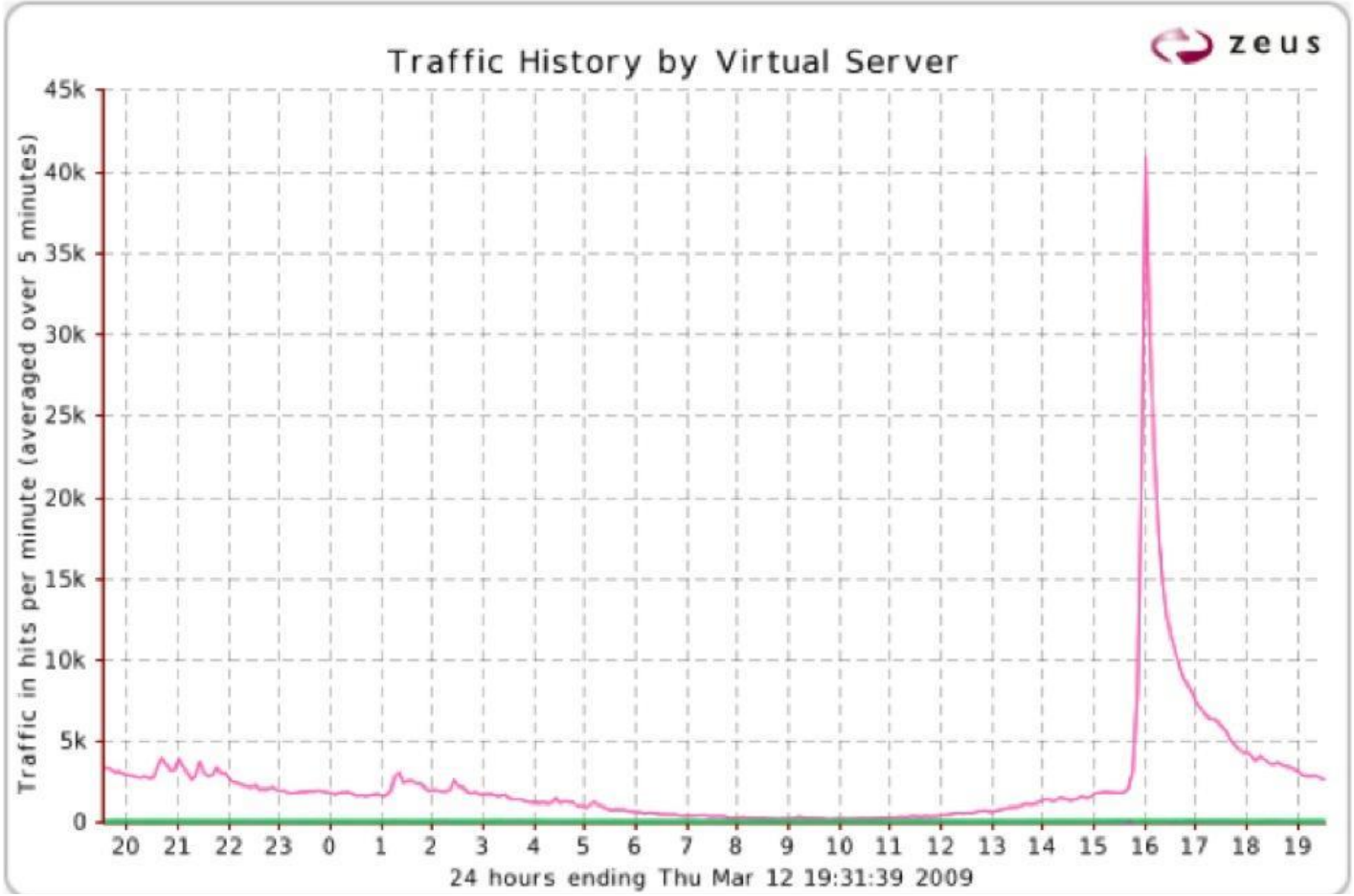


... we receive, store, pack, and ship ...



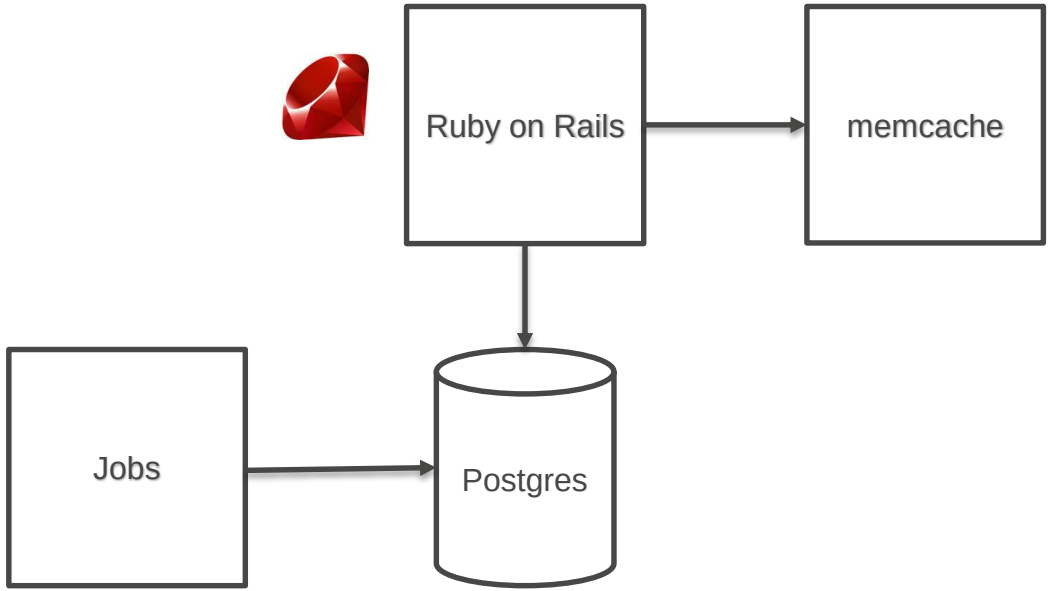
... we sell every day at noon EST

… stampede!



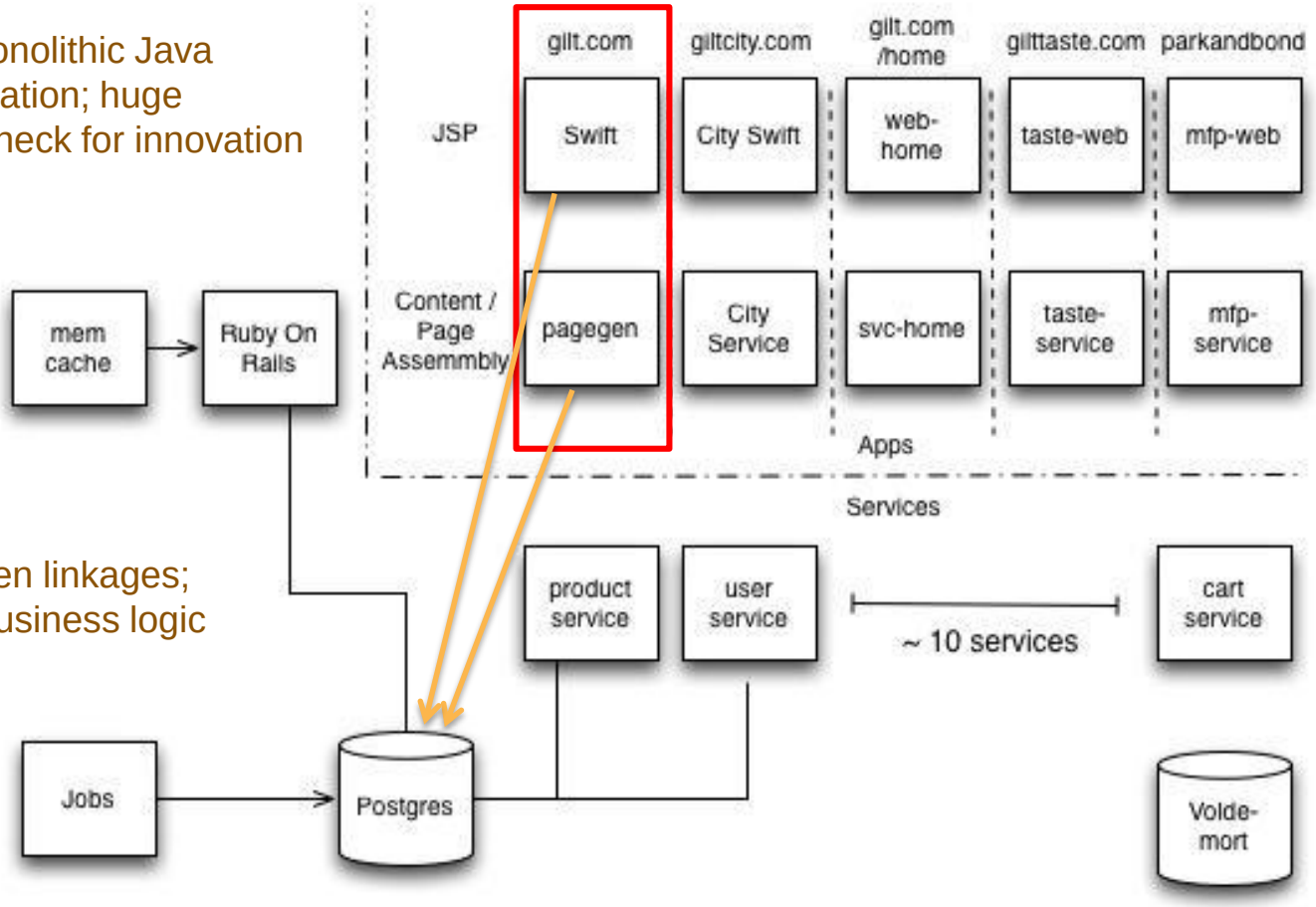
... this is what noon *really* looks like.

How It All Started…



From Rails to Riches — 2007: A Ruby on Rails monolith

(3) Monolithic Java application; huge bottleneck for innovation



(2) Teams focused on business lines

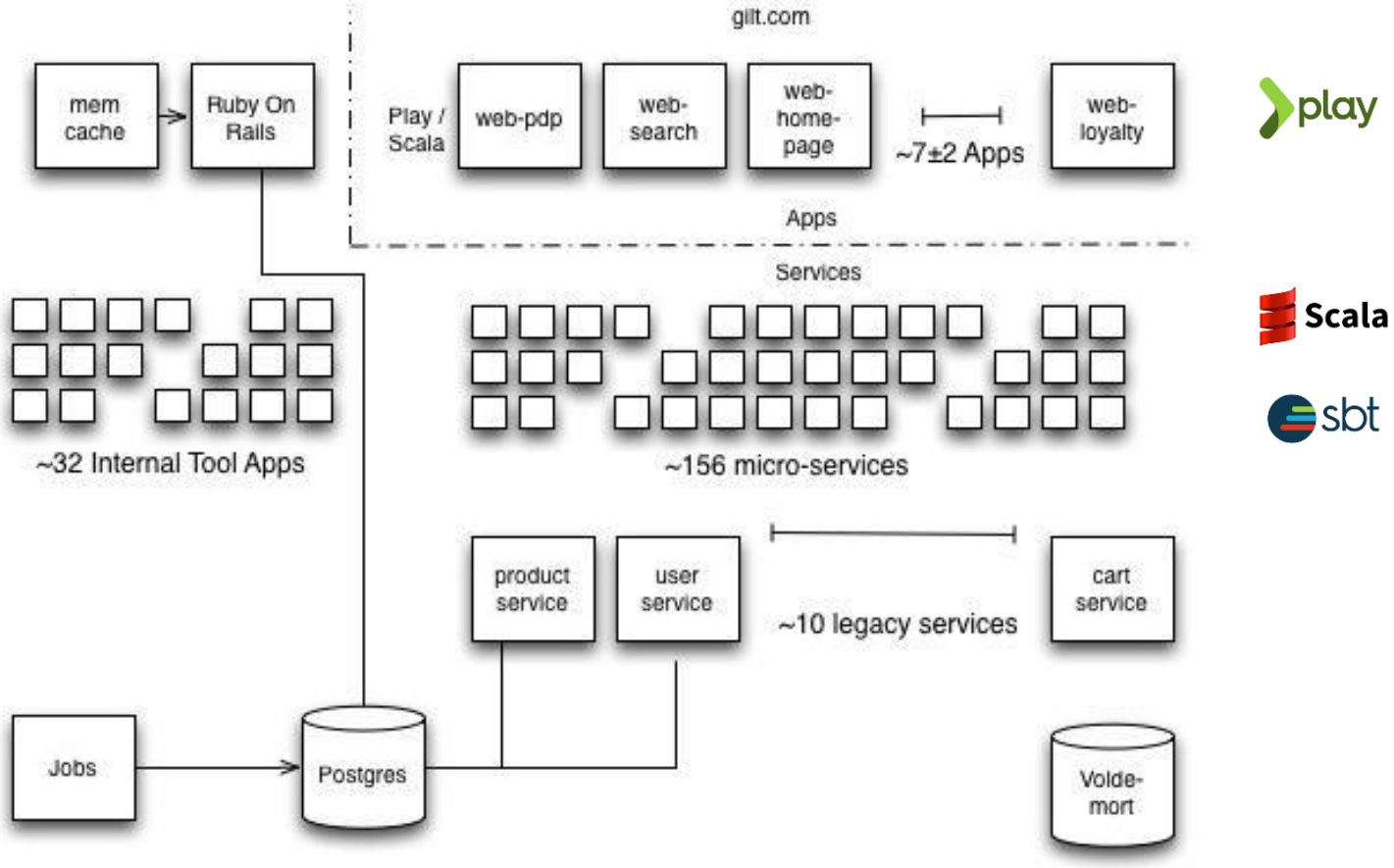
(4) Hidden linkages; buried business logic

(1) Large, loosely typed services

2011: Java, loosely-typed, monolithic services

“How can we arrange our teams around strategic initiatives? How can we make it fast and easy to get to change to production?”

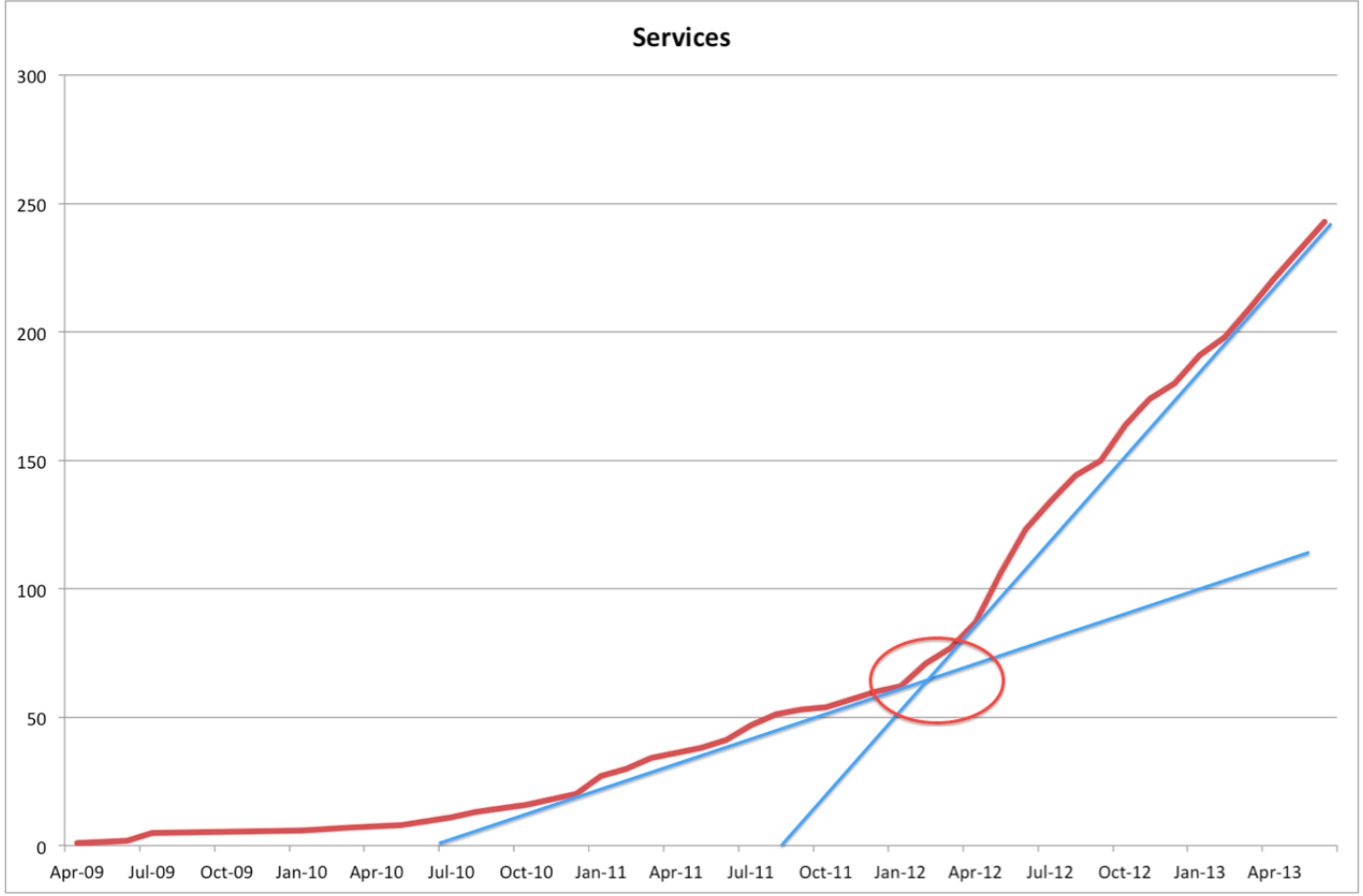
Enter: µ-services



2015: LOSA (Lots of Small Apps) & Microservices

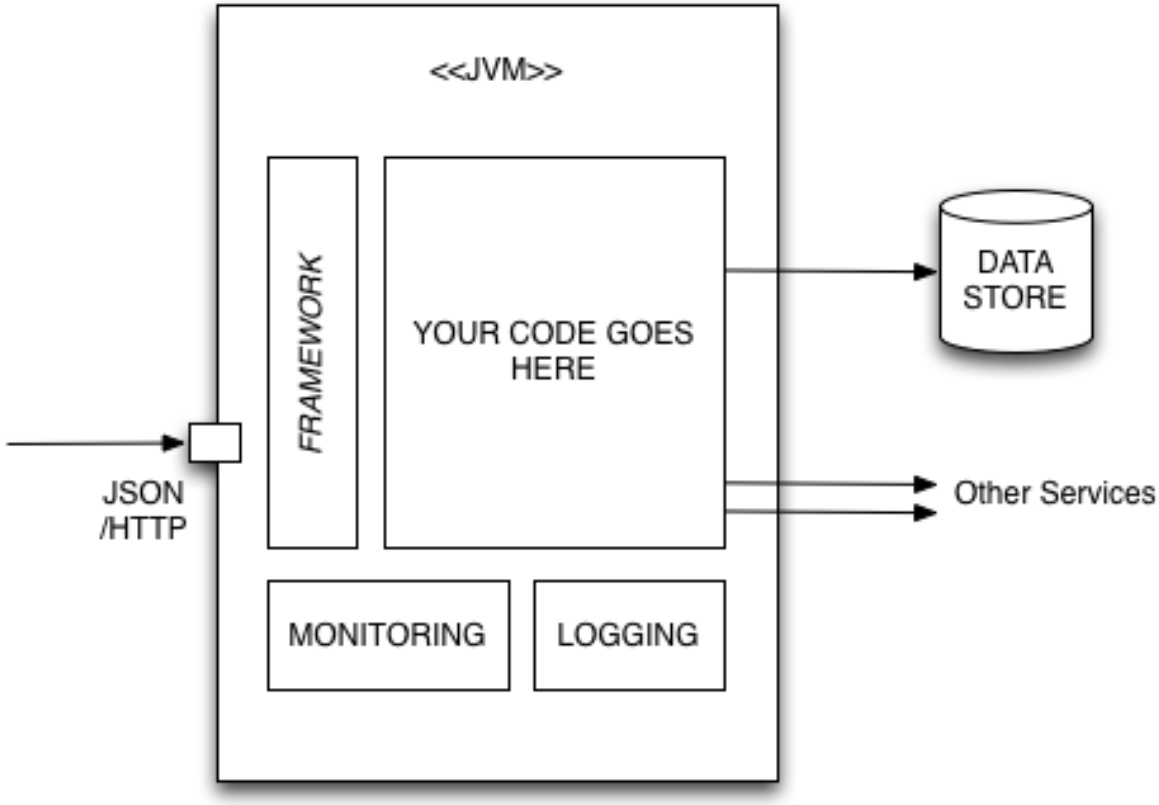
Driving Forces Behind Gilt's *Emergent* Architecture

- Team autonomy
- Voluntary adoption (tools, techniques, processes)
- KPI / goal-driven initiatives
- Failing fast and openly
- Being open and honest, even when it's difficult

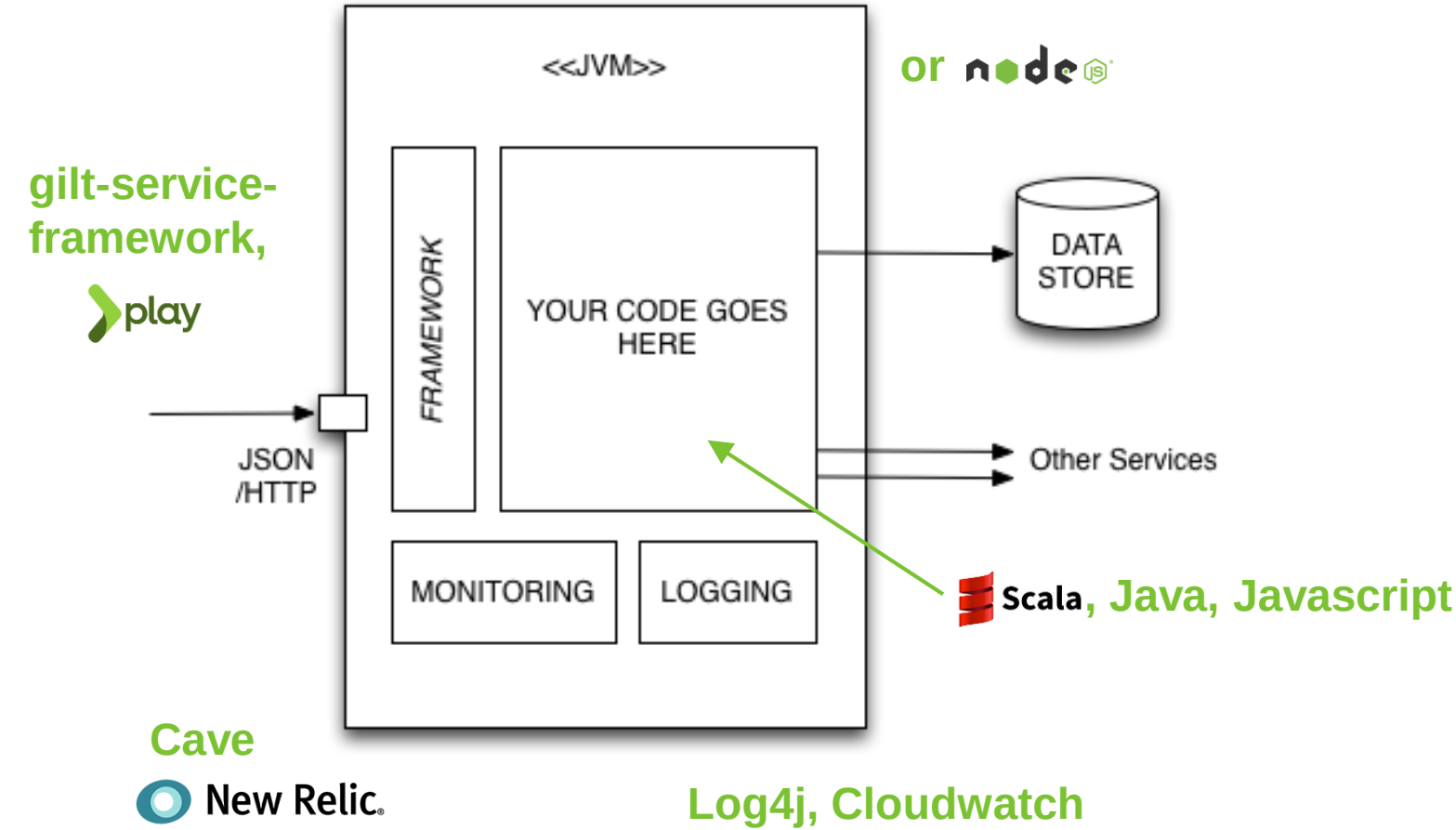


Service growth over time: point of inflexion === Scala.

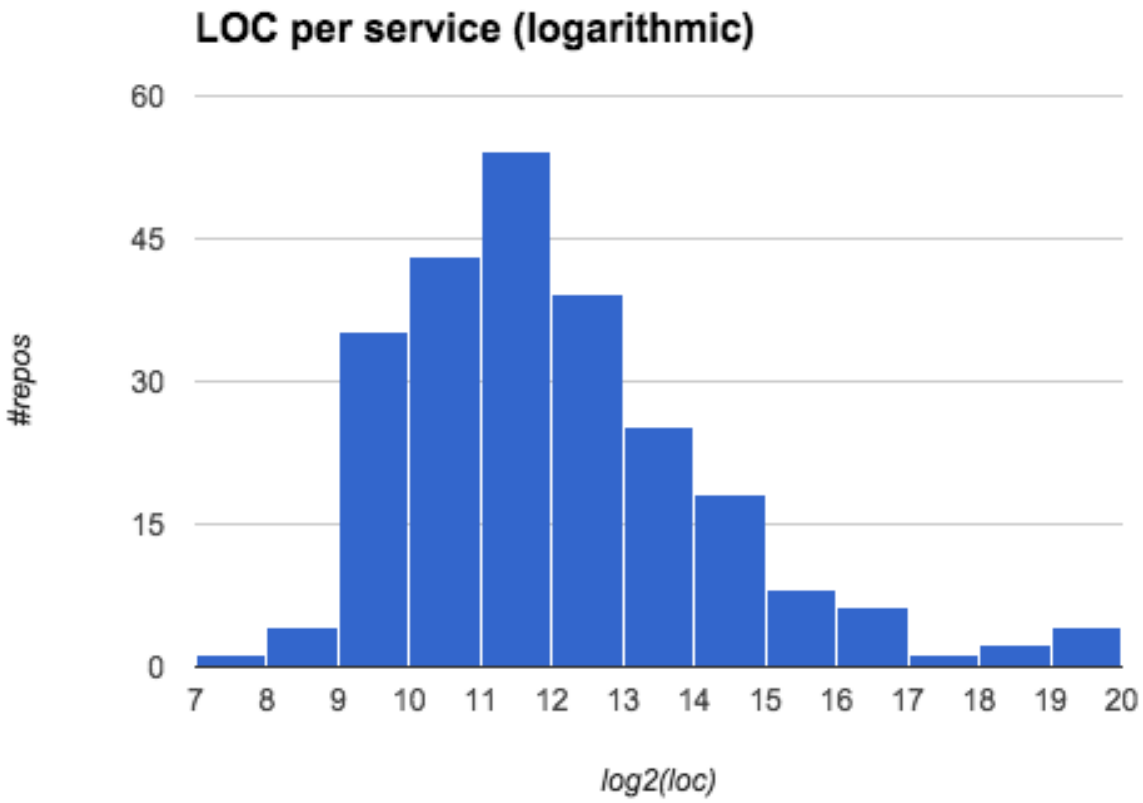
What are all these services doing?



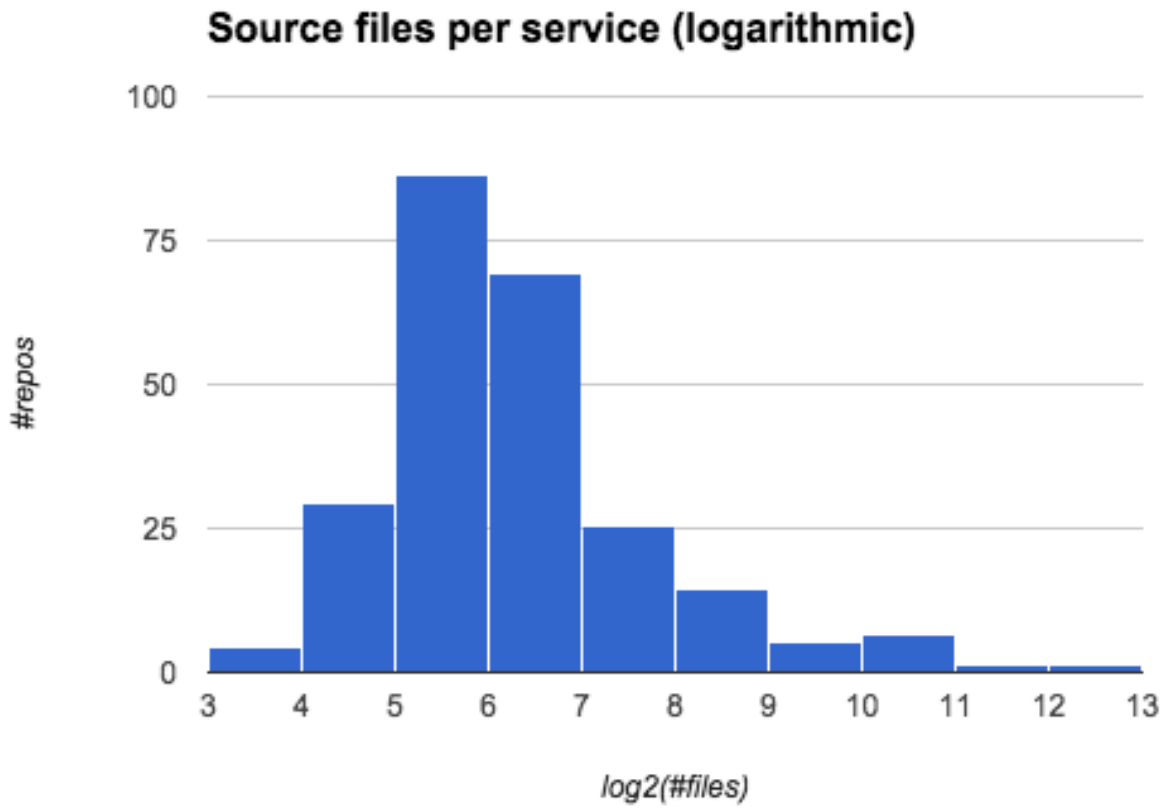
Anatomy of a service



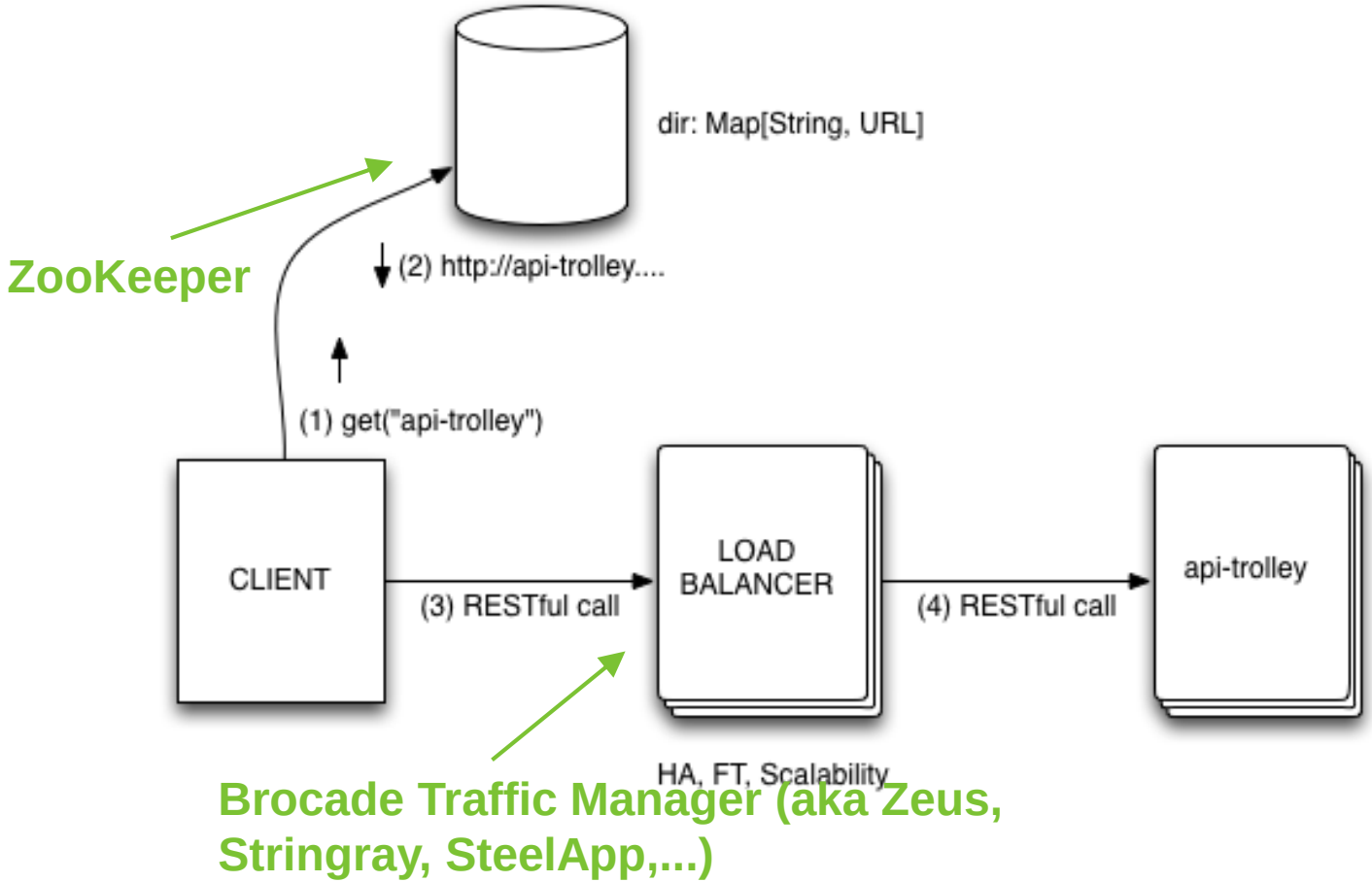
Anatomy of a gilt service – typical choices



Lines of code per service (logarithmic scale)

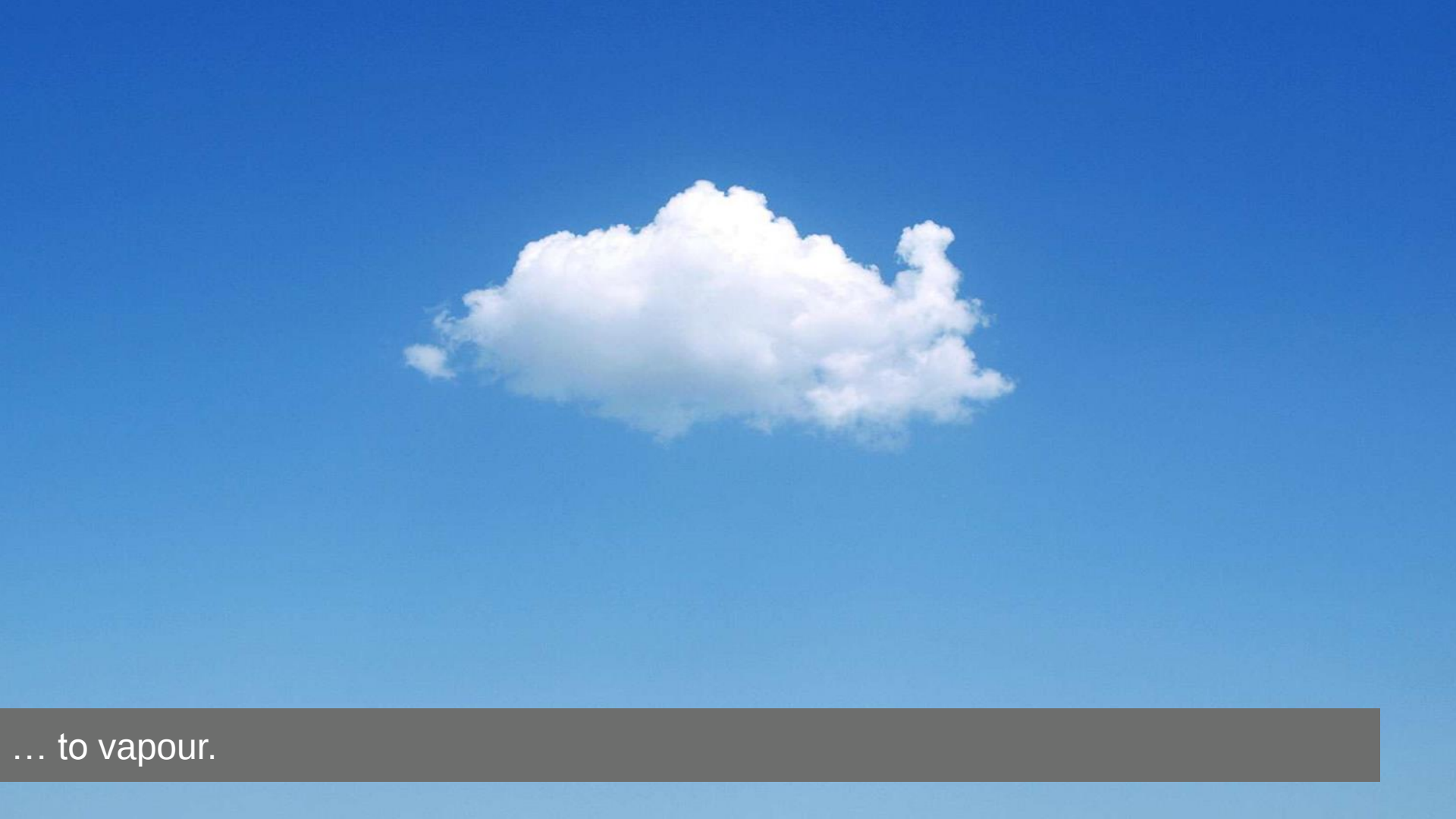


source files per service (includes build, config, xml, Java, Scala, Ruby...)

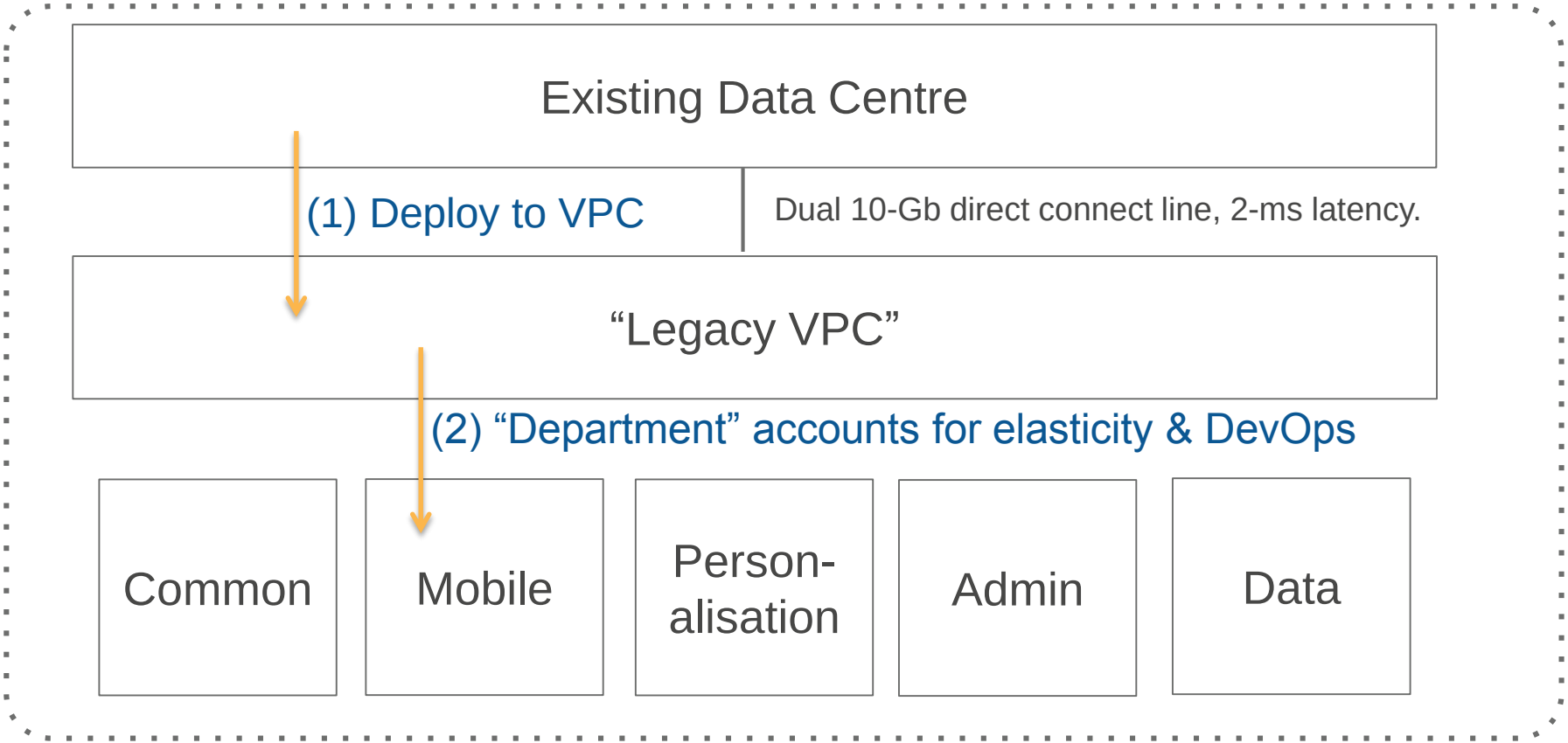


Service discovery: straightforward

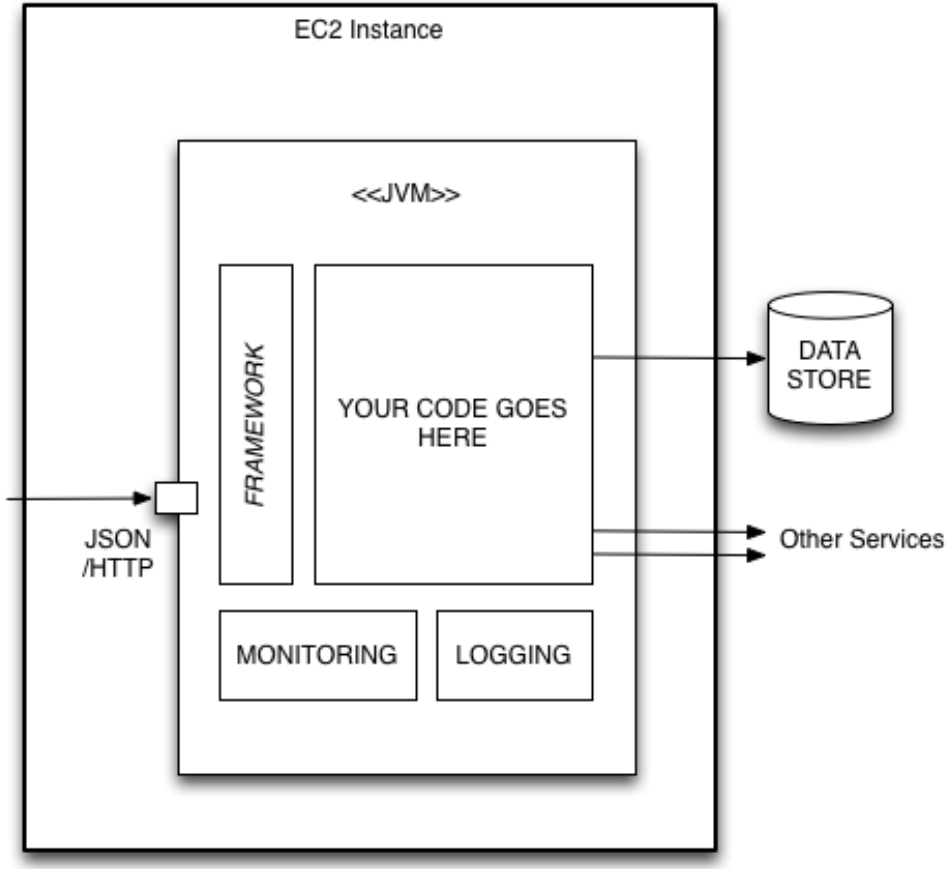
From bare metal…



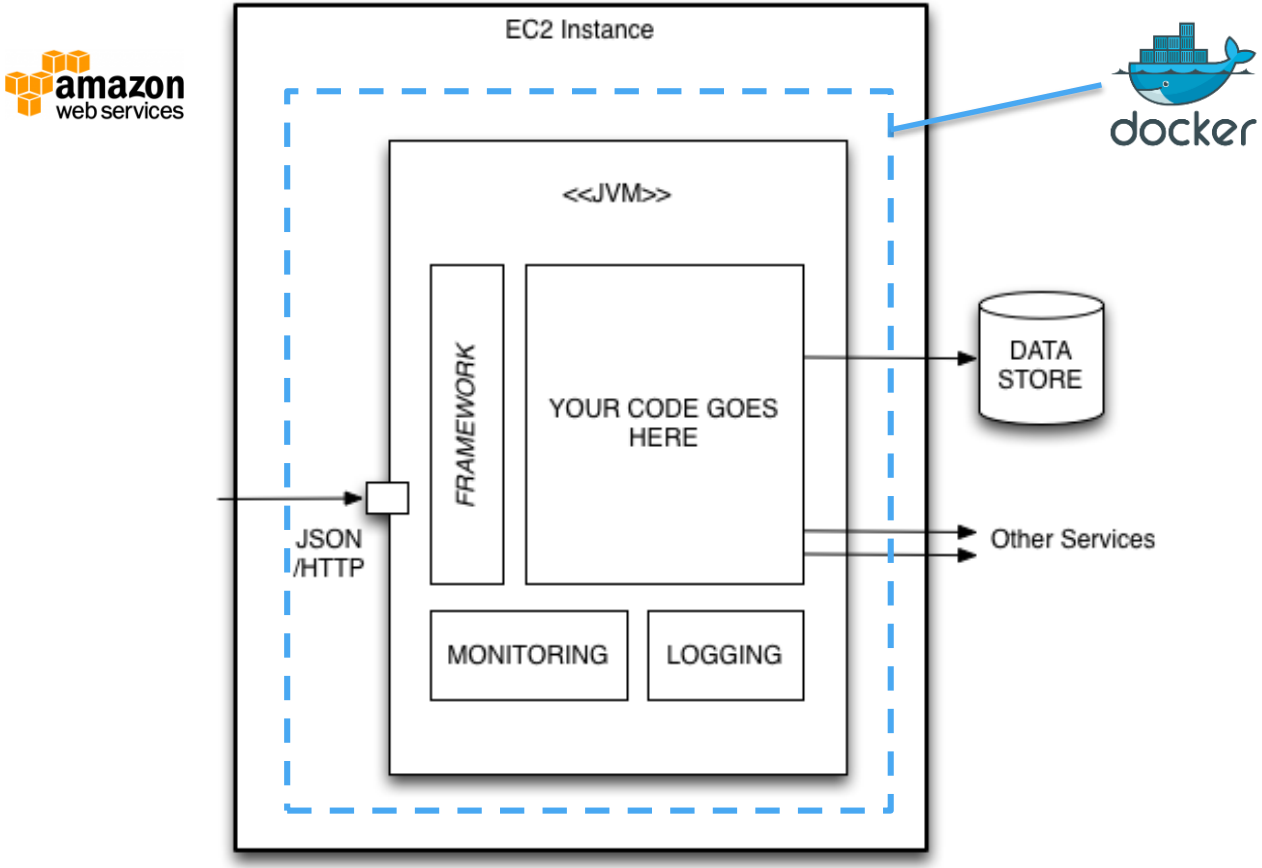
… to vapour.



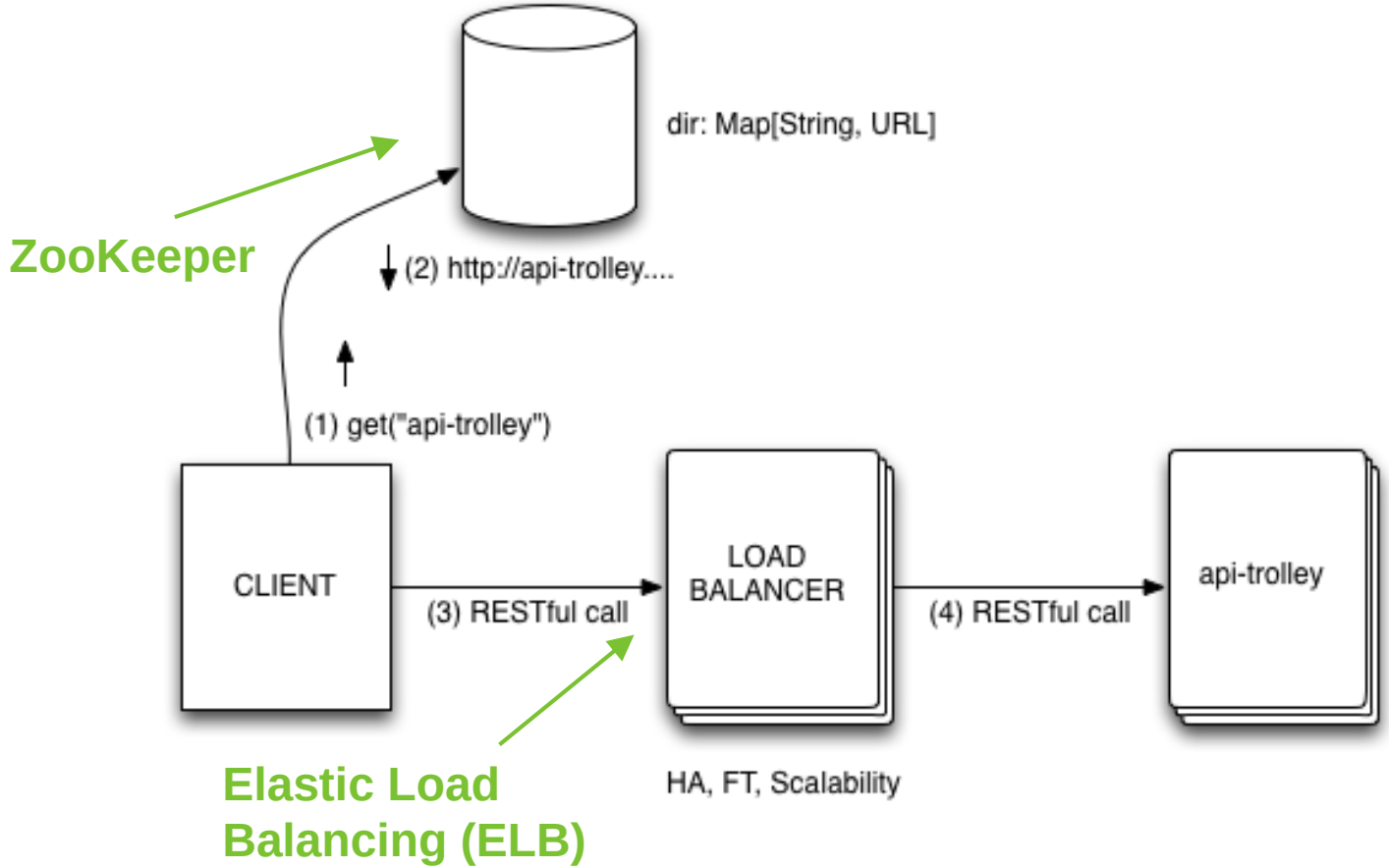
Lift-and-shift + elastic teams



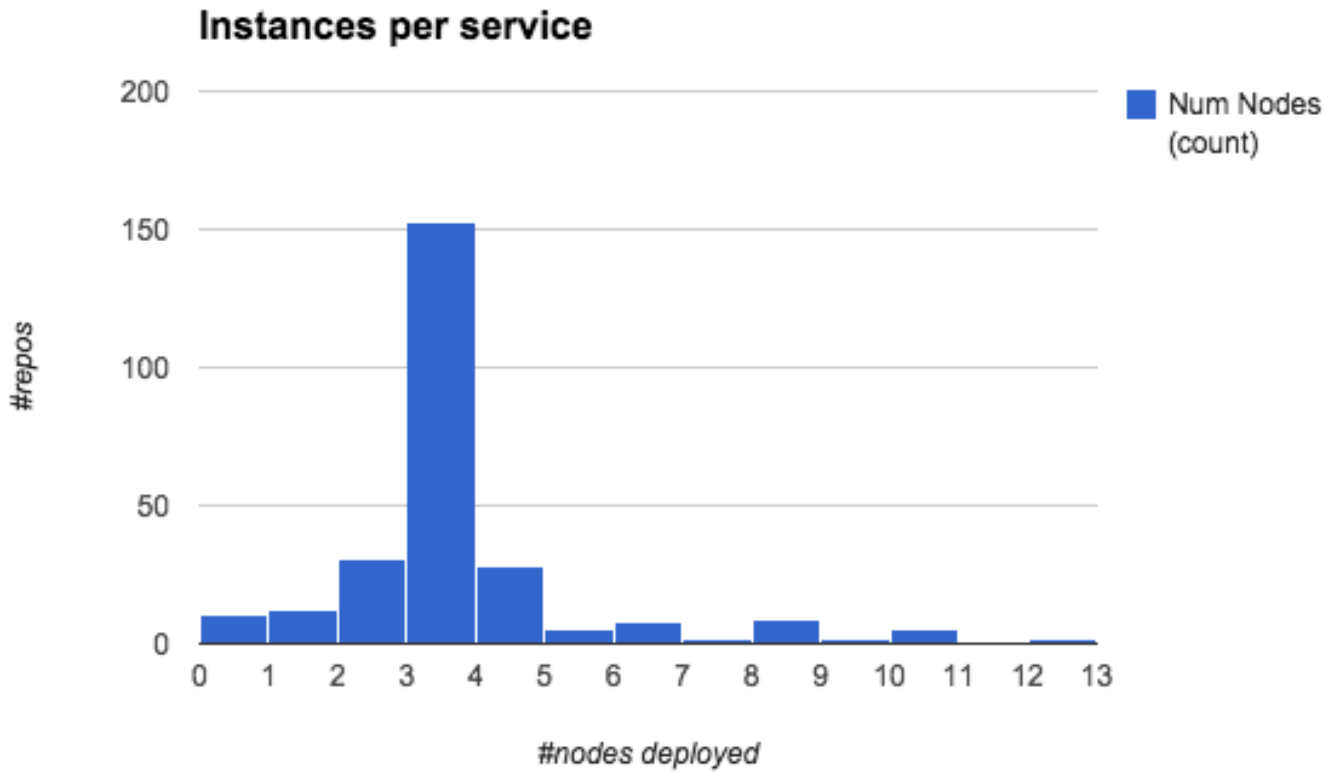
Single-tenant deployment: one service per EC2 instance



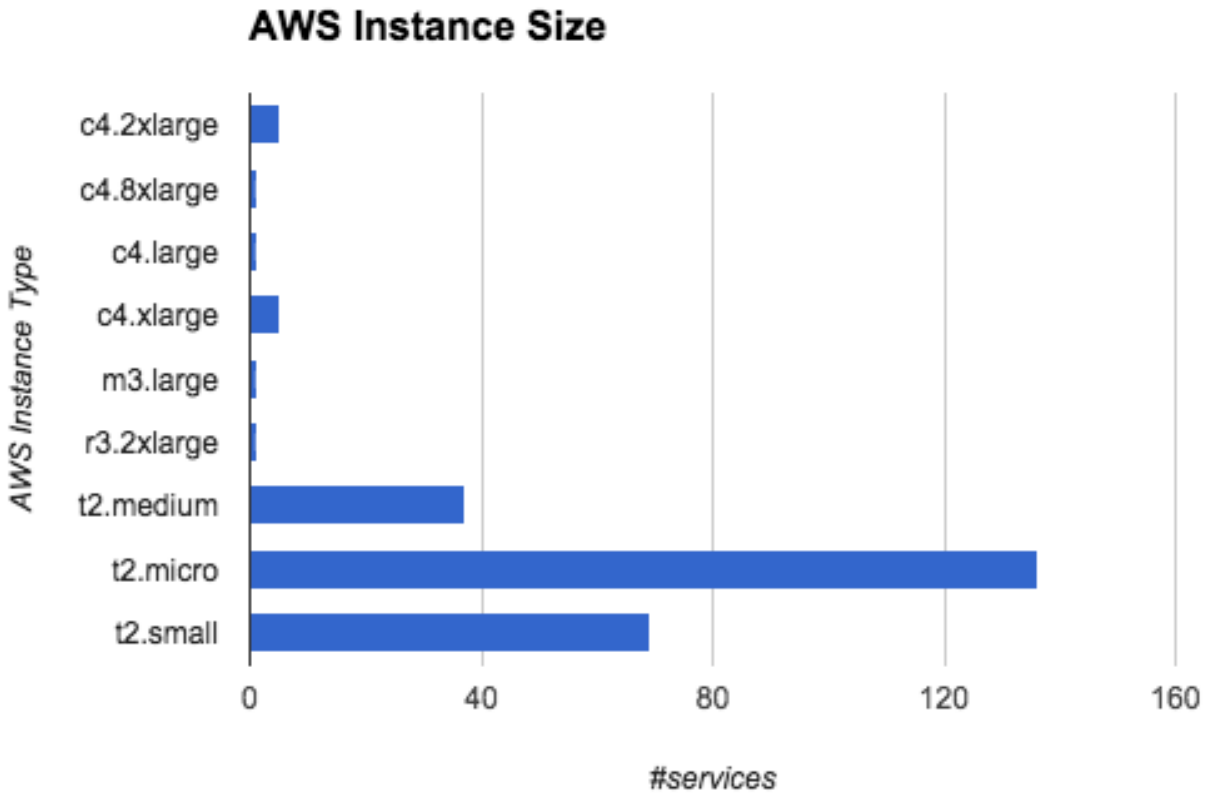
Reproducible, immutable deployments: Docker



Service discovery: new services use ELB

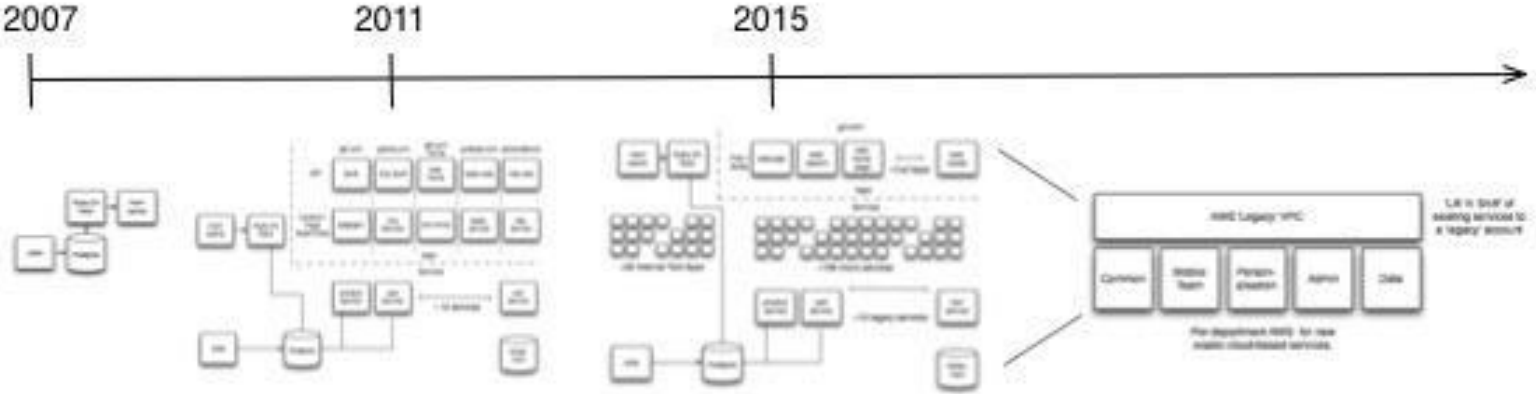


running instances per service: “rule of three” (previously “rule of four”)



EC2 instance sizing: lots of small instances

GILT: Evolution of a Micro-Services Architecture in a $1B startup



RoR Monolith	Java,Scala Loosely Typed Services	Lots of Small Apps (LOSA) Lots of Micro Services	Get thee to the cloud.
A few big teams	Team-per-business.	Team-per-initiative. Lots of initiatives. Team & Individual autonomy. Individual autonomy. KPIs	Teams of Teams (Depts.) Self-service Infrastructure by Dept. Lots of Initiatives. Lots of Autonomy.

Evolution of architecture and tech organization

We (heart) µ-services

- Lessen dependencies between teams: faster code-to-prod
- Lots of initiatives in parallel
- Your favourite <tech/language/framework> here
- Graceful degradation of service
- Disposable code: easy to innovate, easy to fail and move on.

We (heart) cloud

- Do DevOps in a *meaningful* way.
- Low barrier of entry for new tech (Amazon DynamoDB, Amazon Kinesis,...)
- Isolation

- Cost visibility
- Security tools (IAM)
- Well documented
- Resilience is easy
- Hybrid is easy
- Performance is great

Common Challenges and Patterns

Monolithic

- Simple deployments
- Binary failure modes
- Inter-module refactoring
- Technology monoculture
- Vertical scaling

Microservices

- Partial deployments
- Graceful degradation
- Strong module boundaries
- Technology diversity
- Horizontal scaling

Common Challenges and Patterns

- Organization
- Discovery
- Data management
- Deployment
- I/O explosion
- Monitoring

Organization

Monolithic Ownership

Organized on technology capabilities

UI Team

App Logic Team

DBA Team

Organizational Structure

Application Architecture

Microservices Ownership

Organized on business responsibilities

Login
Registration
Order

Accounts team

Personalization

Personalization team

Mobile

Mobile team

Microservices Ownership

- Requirements
- Technology selection
- Development
- Quality
- Deployment
- Support

How to Be a Good Citizen (Service Consumer)

- Design for failure
- Expect to be throttled
- Retry w/ exponential backoff
- Degrade gracefully
- Cache when appropriate

How to Be a Good Citizen (Service Provider)

- Publish your metrics
- Protect yourself
- Keep your implementation details private
- Maintain backwards compatibility

Amazon API Gateway

- Throttling (global and per-method)
- Caching (with TTLs and invalidation)
- Monitoring (RPS, latency, error rate)
- Versioning
- Authentication

Discovery

Use DNS

Convention-based naming

<service-name>-<environment>.domain.com

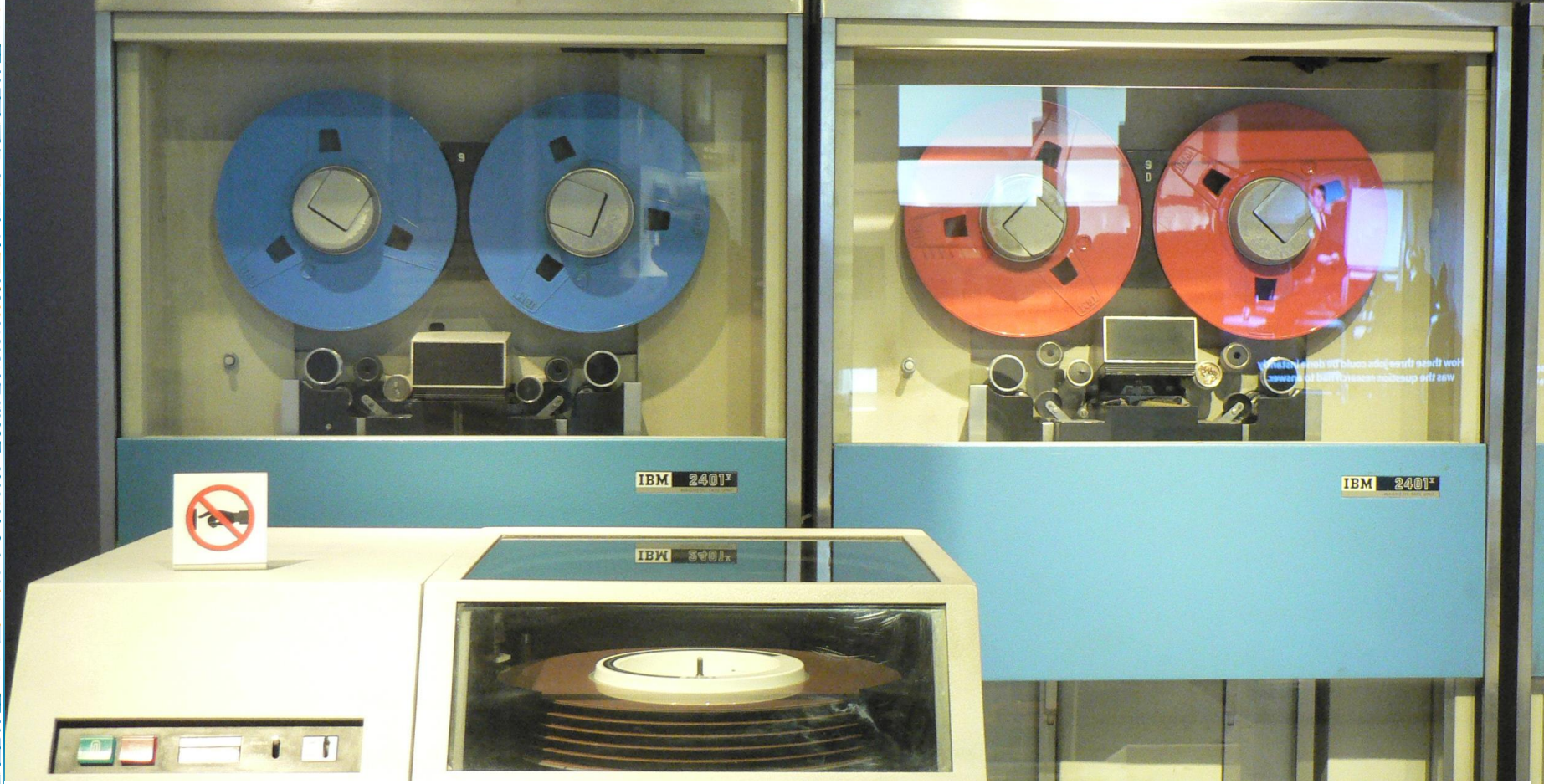
shoppingcart-gamma.example.com

<service-name>.<environment>.domain.com

shoppingcart.gamma.example.com

Use a Dynamic Service Registry

- Avoids the DNS TTL issue
- More than service registry & discovery
  - Configuration management
  - Health checks
- Plenty of options
  - ZooKeeper (Apache)
  - Eureka (Netflix)
  - Consul (HashiCorp)
  - SmartStack (Airbnb)

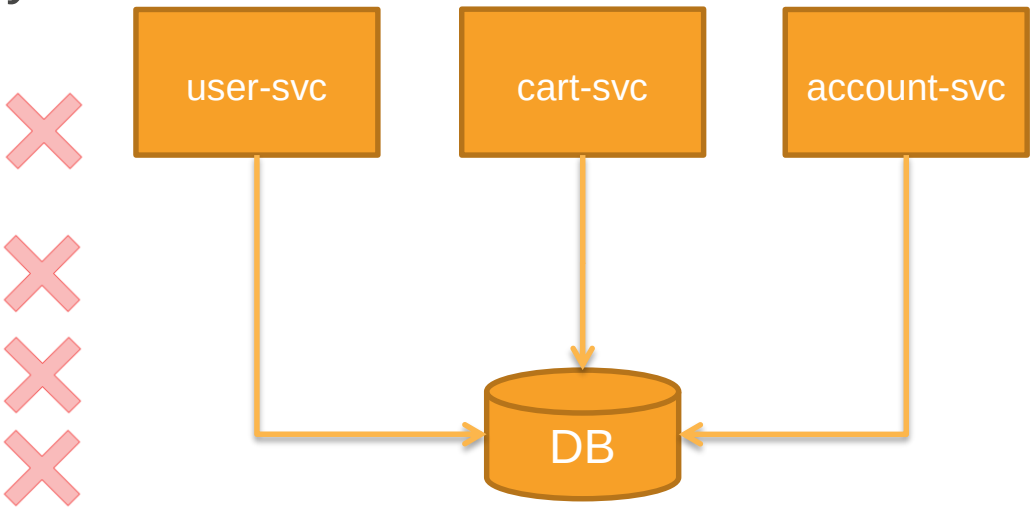


Data management

Challenge: Centralized Database

Monolithic applications typically have a monolithic data store:

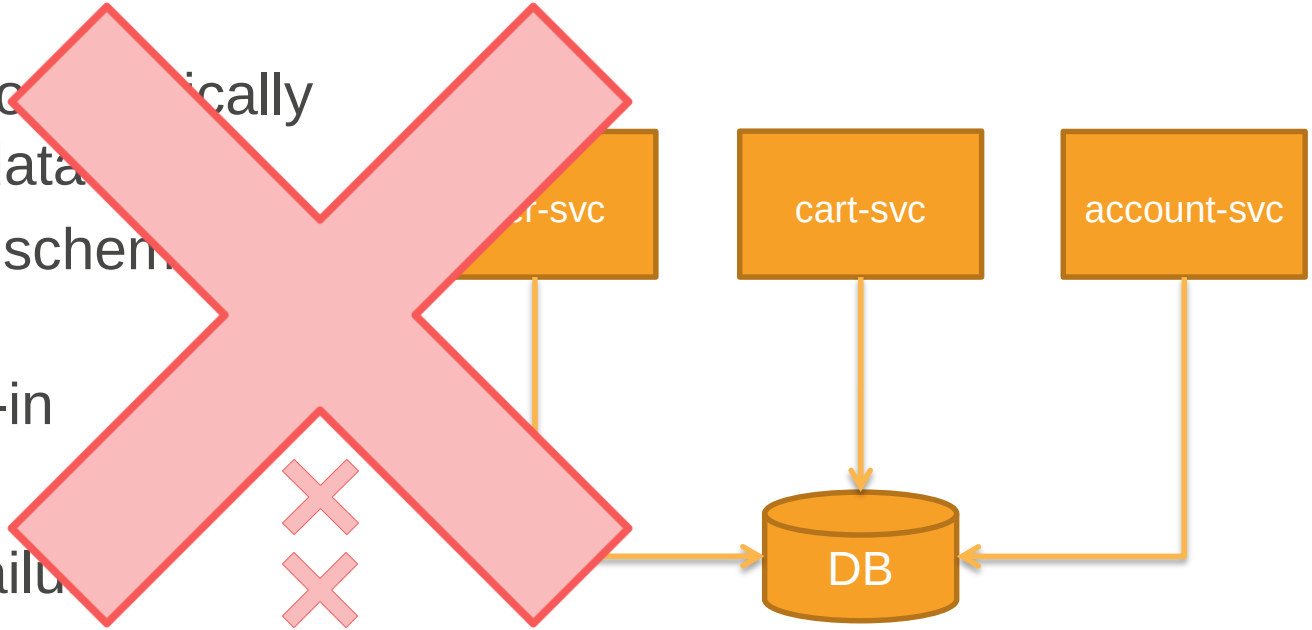
- Difficult to make schema changes
- Technology lock-in
- Vertical scaling
- Single point of failure



Centralized Database – Anti-pattern

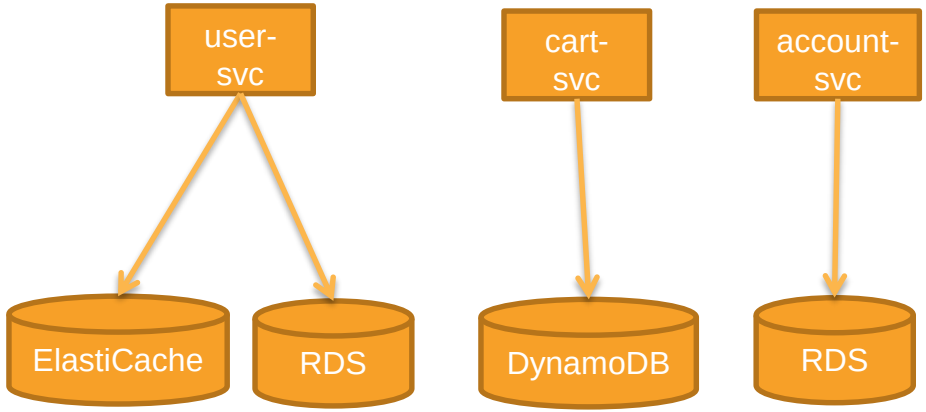
Monolithic applicatio ically have a monolithic data

- Difficult to make schem changes
- Technology lock-in
- Vertical scaling
- Single point of failu



Decentralized Data Stores

- Each service chooses its data store technologies
- Low impact schema changes
- Independent scalability
- Data is gated through the service API



Challenge: Transactional Integrity

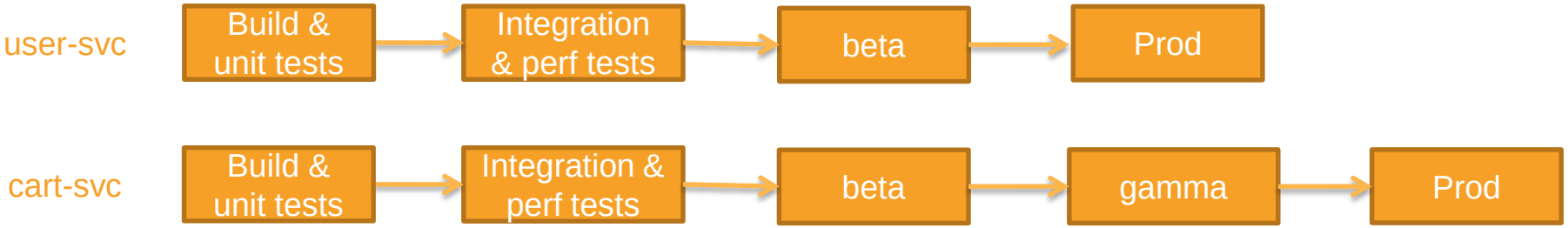
- Use a pessimistic model
  - Handle it in the client
  - Add a transaction manager / distributed locking service
  - Rethink your design
- Use an optimistic model
  - Accept eventual consistency
  - Retry (if idempotent)
  - Fix it later
  - Write it off

Challenge: Aggregation

- **Pull:** Make the data available via your service API
- **Push:** To Amazon S3, Amazon CloudWatch, or another service you create
- **Pub/sub:** Via Amazon Kinesis or Amazon SQS

Deployment

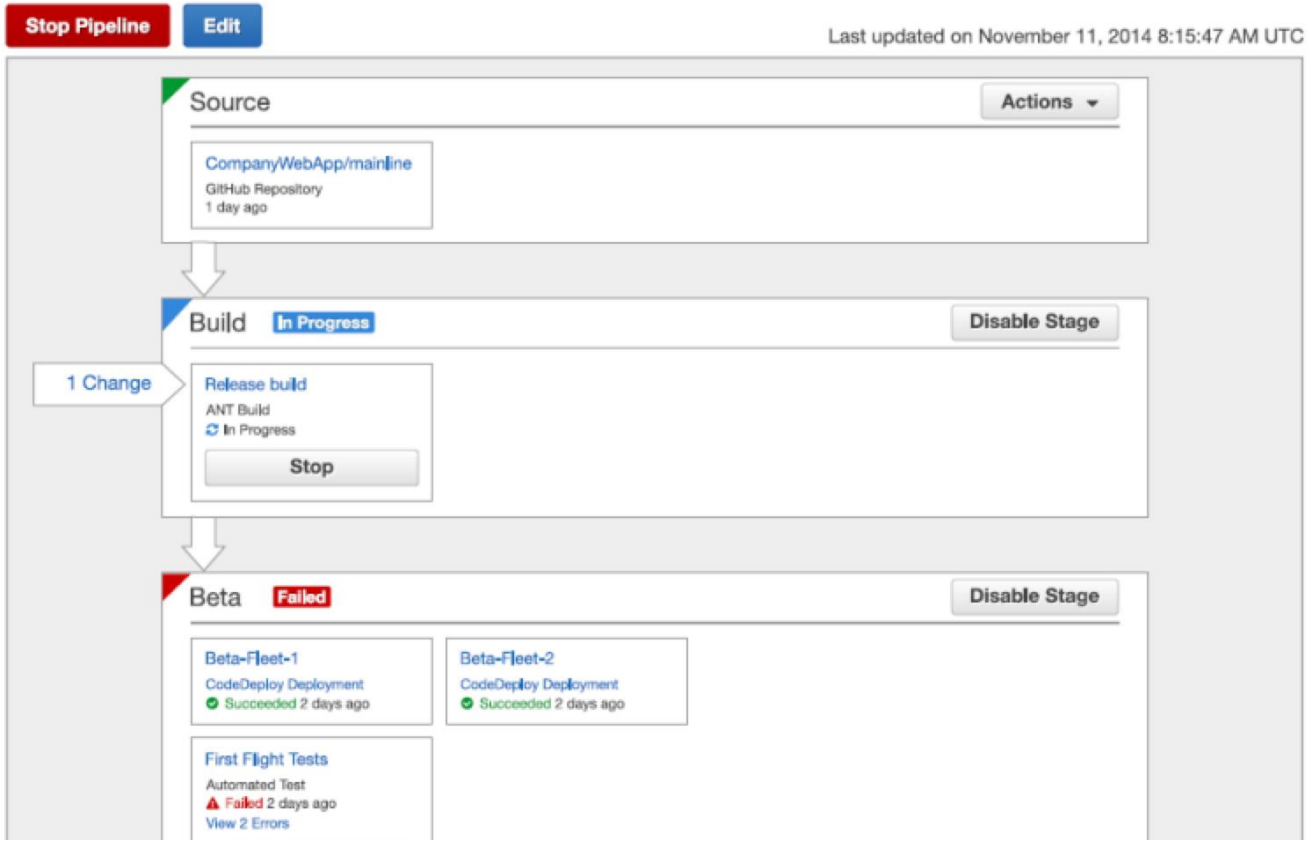
Continuous Delivery & Continuous Deployment



Create the right build pipeline for each service

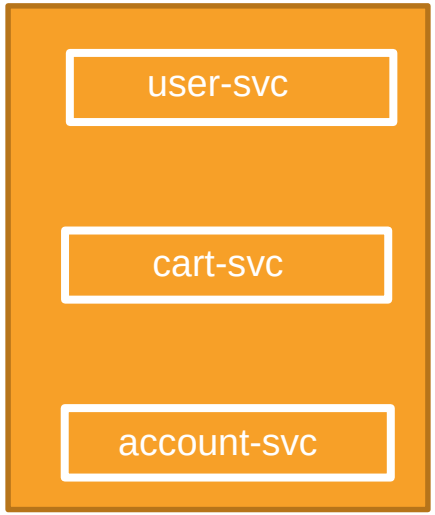
- AWS CodeDeploy
- AWS Elastic Beanstalk
- Jenkins, CircleCI, Travis,…

AWS CodePipeline



Multiple Services per Container/Instance

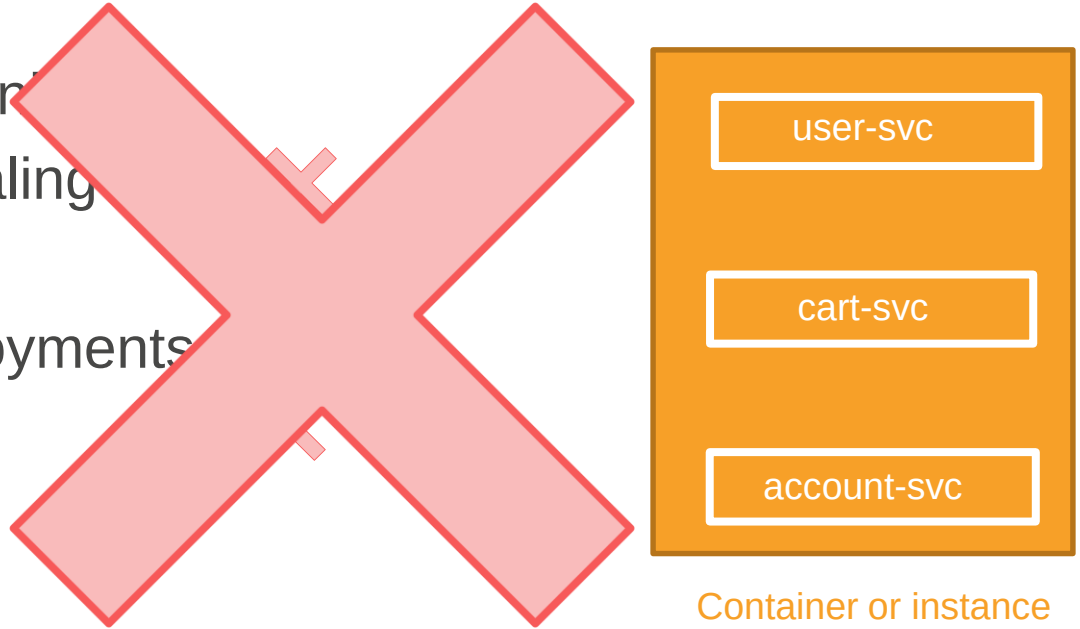
- Independent monitoring
- Independent scaling
- Clear ownership
- Immutable deployments



Container or instance

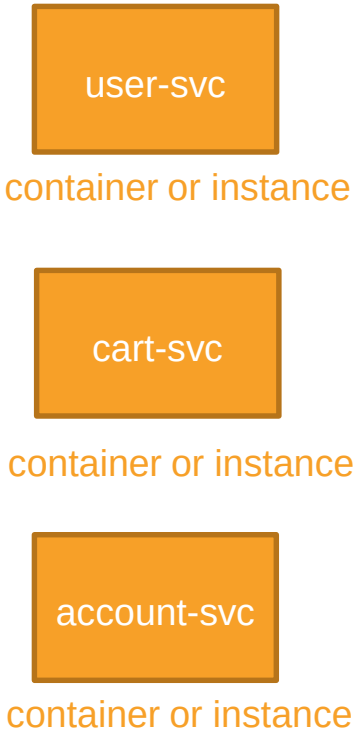
Multiple Services per Container/Instance – Anti-pattern

- Independent mon
- Independent scaling
- Clear ownership
- Immutable deployments

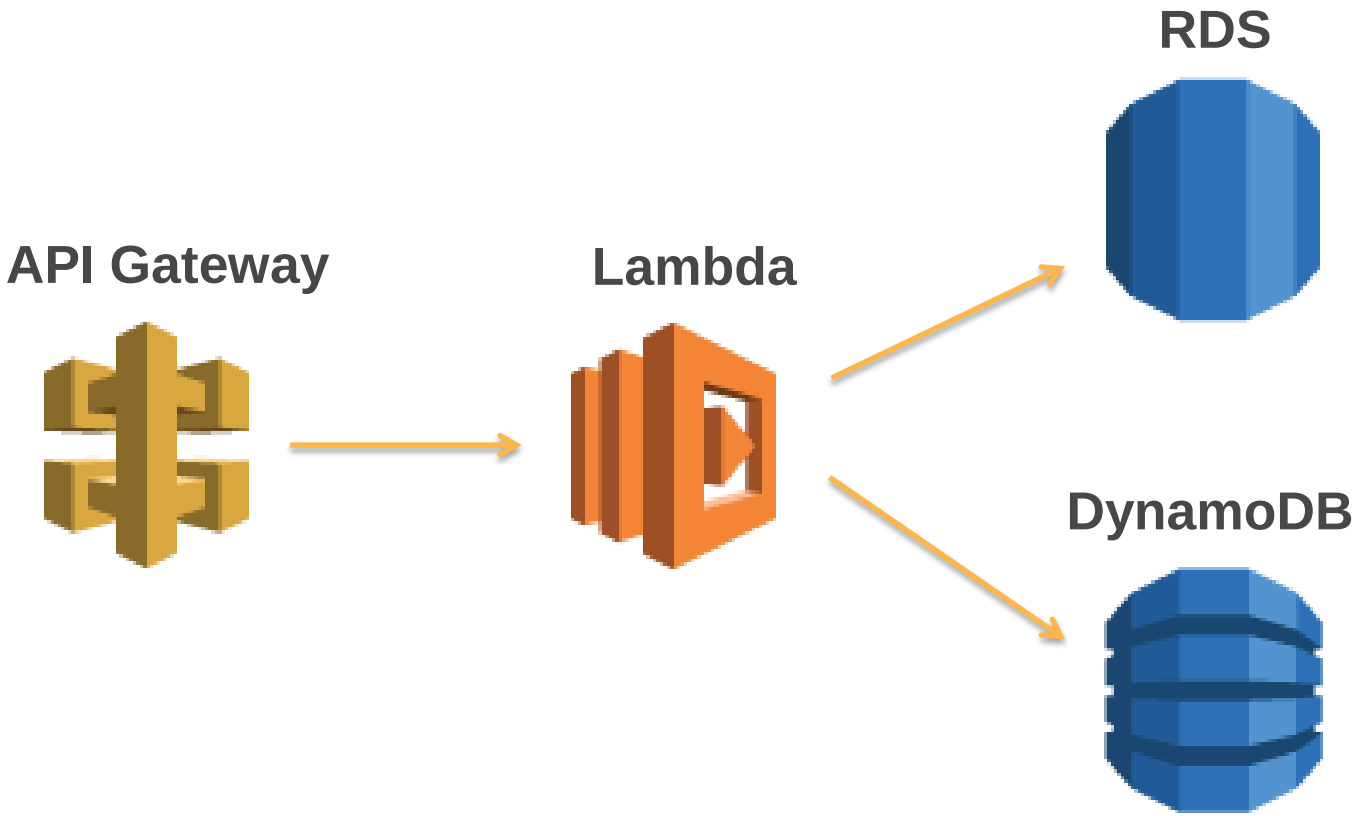


Single Service per Container/Instance

- Independent monitoring
- Independent scaling
- Clear ownership
- Immutable deployments

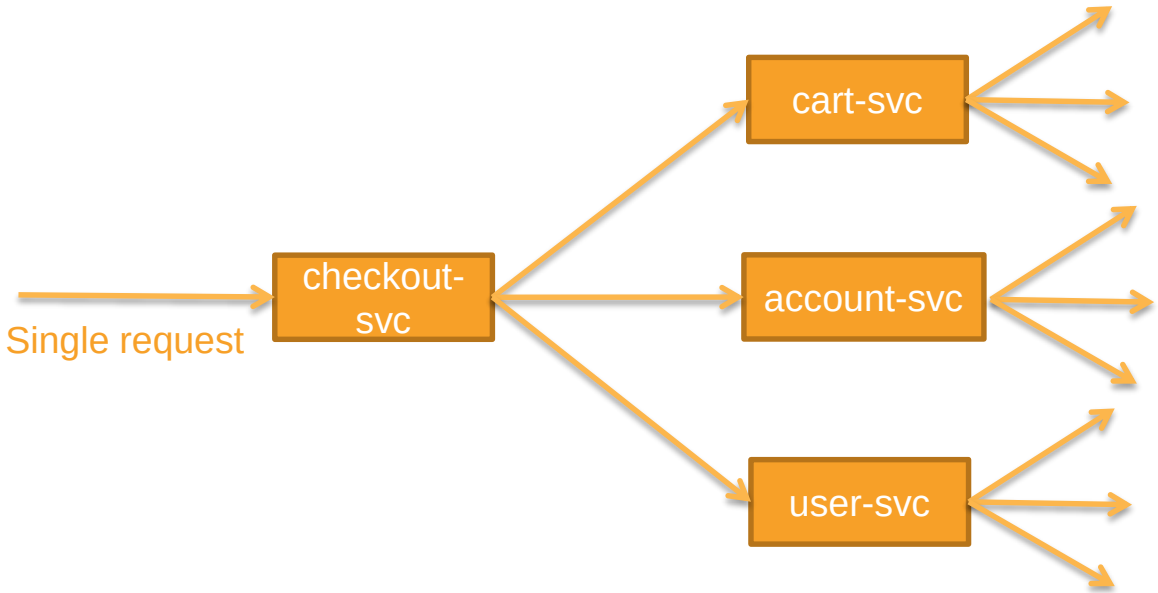


…Or Just Use AWS Lambda

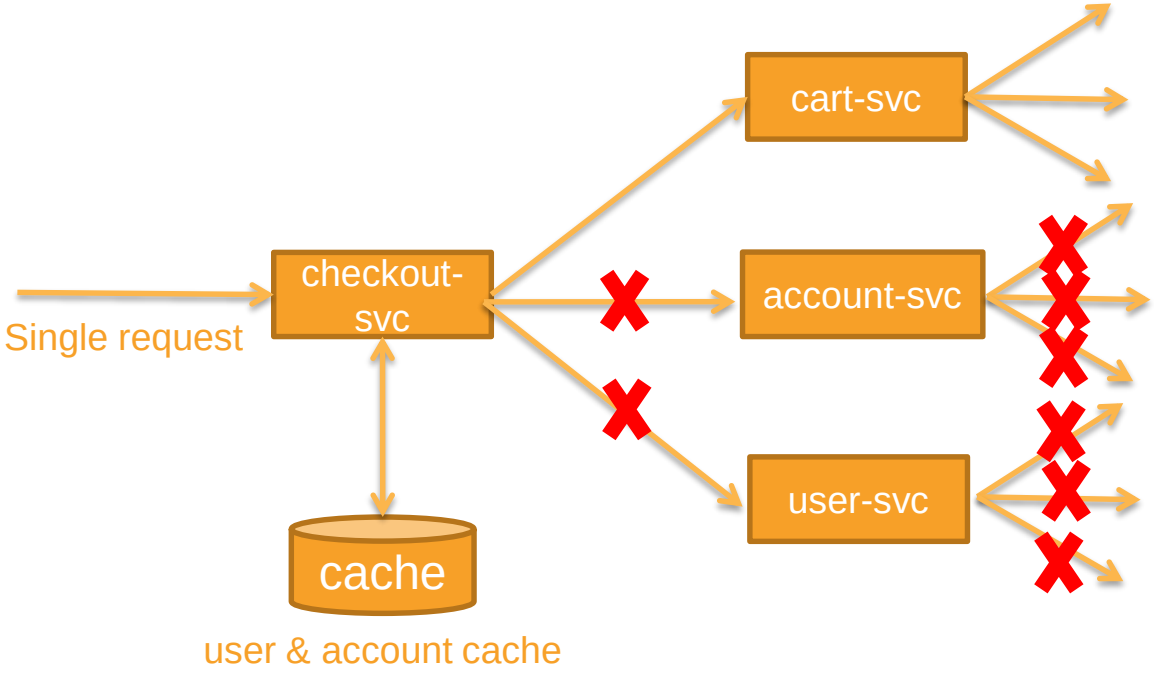


I/O explosion

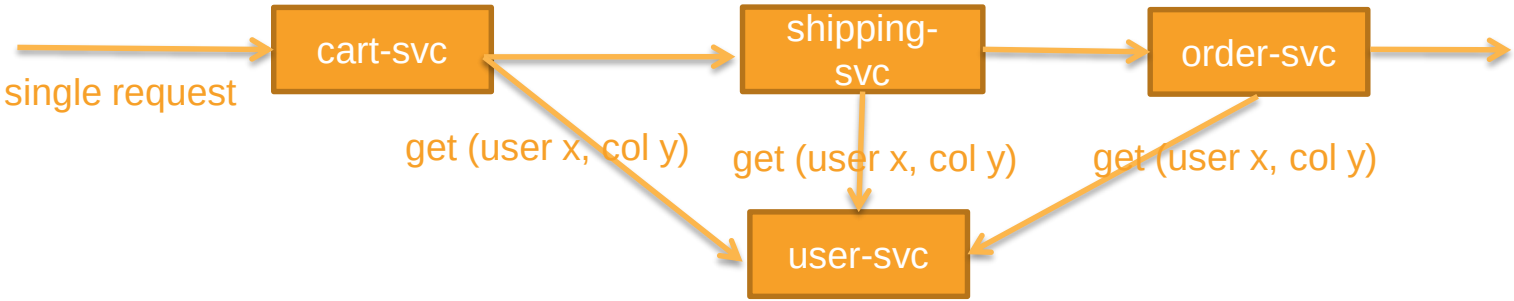
Challenge: Request Multiplication



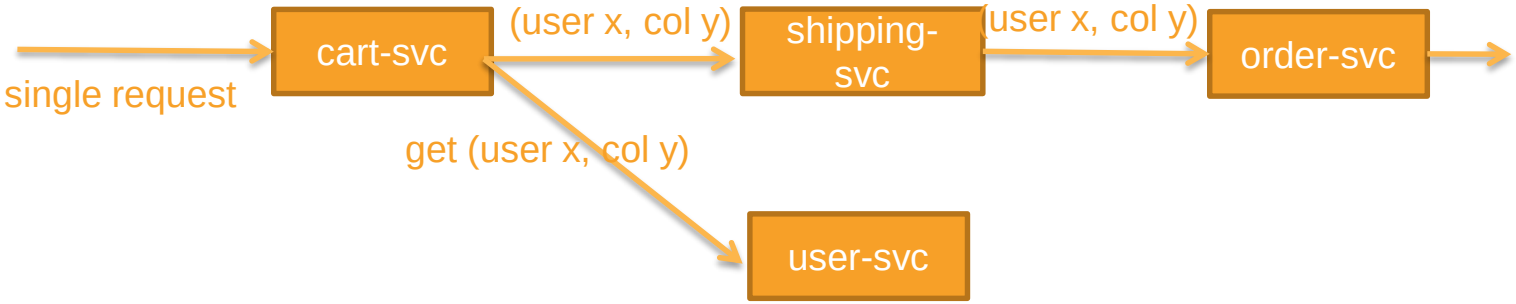
Add Client Caching



Challenge: Hotspots



Use Dependency Injection



Monitoring

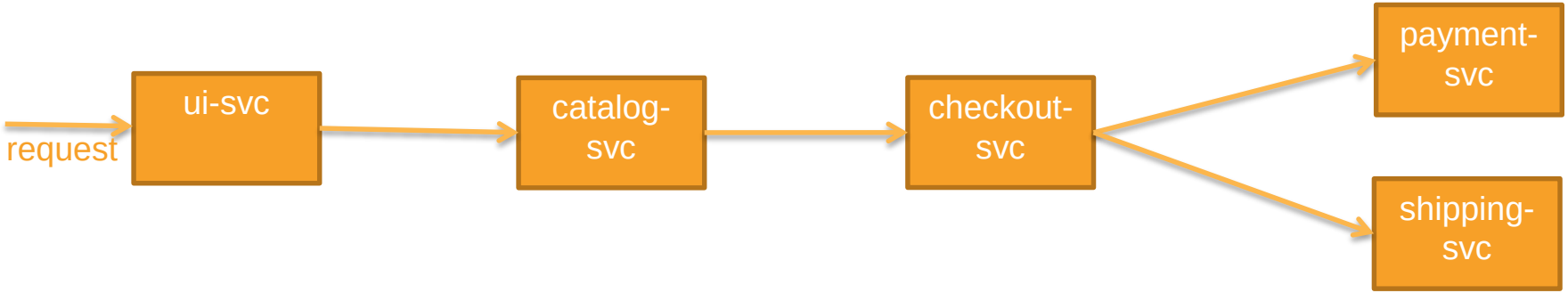
Challenge: monitoring

- *Publish* externally relevant metrics
  - Latency
  - RPS
  - Error rate
- *Understand* internally relevant metrics
  - Basic – CloudWatch
  - OS
  - Application

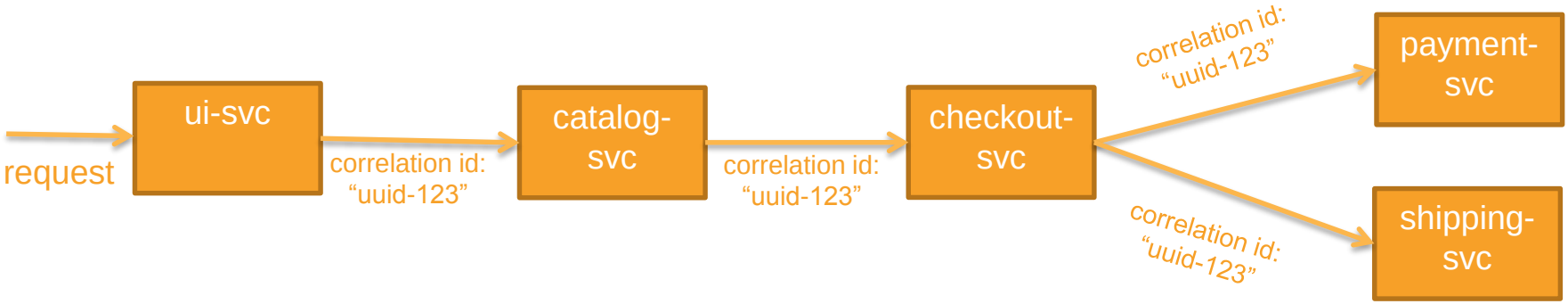
Challenge: Logging

- Pick a common log aggregation solution
- Agree on log entry formats
- Use naming conventions
- Agree on correlation strategy

Challenge: Correlating Requests



Use Correlation IDs




```
09-02-2015 15:03:24 ui-svc INFO [uuid-123] ……
09-02-2015 15:03:25 catalog-svc INFO [uuid-123] ……
09-02-2015 15:03:26 checkout-svc ERROR [uuid-123] ……
09-02-2015 15:03:27 payment-svc INFO [uuid-123] ……
09-02-2015 15:03:27 shipping-svc INFO [uuid-123] ……
```

What did we cover?

- Ownership
- Discovery
- Data management
- Deployment
- I/O explosion
- Monitoring

Related Sessions

- ARC201 - Microservices Architecture for Digital Platforms with AWS Lambda, Amazon CloudFront and Amazon DynamoDB
- CMP302 - Amazon EC2 Container Service: Distributed Applications at Scale
- DEV203 - Using Amazon API Gateway with AWS Lambda to Build Secure and Scalable APIs
- DVO401 - Deep Dive into Blue/Green Deployments on AWS
- SPOT304 - Faster, Cheaper, Safer Products with AWS: Adrian Cockcroft Shares Experiences Helping Customers Move to the Cloud

Remember to complete your evaluations!

Thank you!

Adrian Trenaman
SVP Engineering, gilt.com
@adrian_trenaman

Derek Chiles
Sr. Mgr, Solutions Architecture, AWS
@derekchiles