Microservice architecture

Žilvinas Kuusas

estina

VilniusPHP 0x19, 2014

Who am I?

Žilvinas Kuusas

lead developer @ Estina / iSign.io

t: @kuusas
e: zilvinas@kuusas.lt

What is a microservice?

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

Martin Fowler

http://martinfowler.com/articles/microservices.html

Microservice

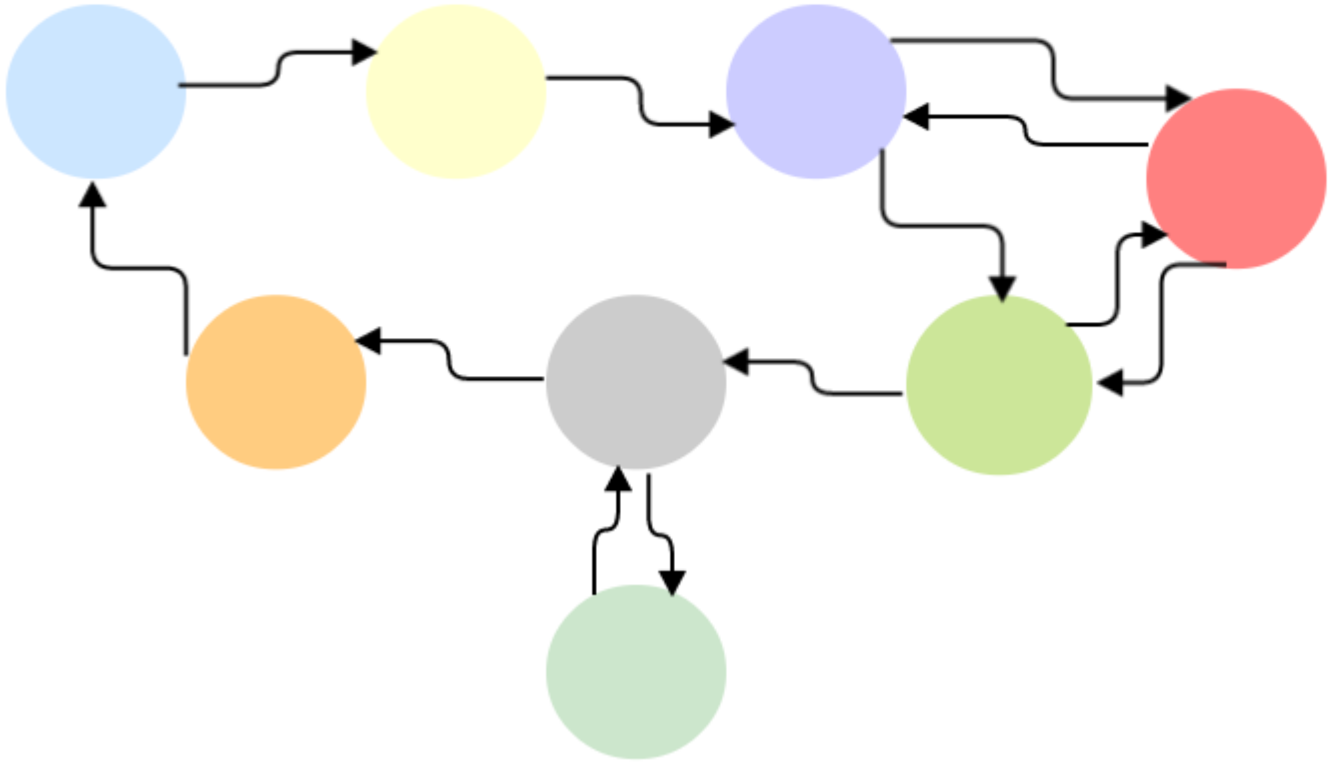
- No long-term relationships with technology stack
- Easy to adopt emerging technologies
- Loose coupling
- Single responsibility
- Fault isolation
- Scalability

UNIX philosophy

```
rename 's/Airplane/Flight/' `find -name "*Airplane*.php”`
```

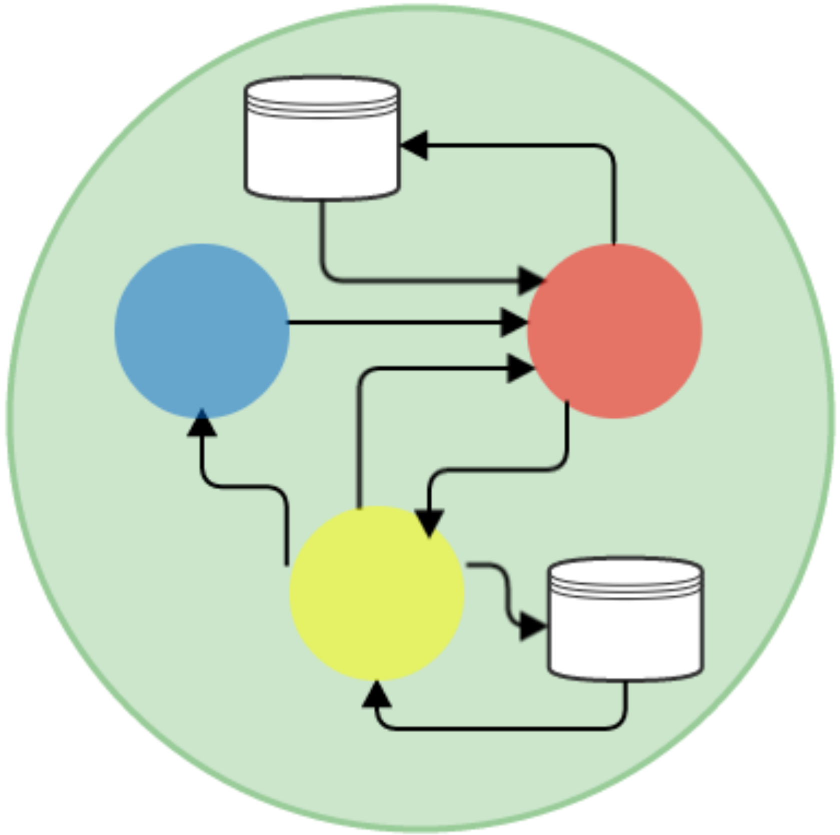
UNIX philosophy

ls . | **sort** | **tail** -n 2 | **sort** -r | **tail** -n 1 | **cut** -c8-21



Microservice architecture

Reality - monolithic application



Service Oriented Architecture (SOA)

Benefits of monolith

- Quick development
- Simple deployments
- Easy to scale
- Everything in one place

Drawbacks of monolith

- Lots of LOC
- Slow builds
- Development is hard to scale
- Continuous deployments becomes difficult
- Scaling application can be difficult
- Requires a long-term commitment to a technology stack

Microservices

Think about microservice as small, single-purpose application. Simple?

Small application

- Runs as individual process
- Smaller means easier for developers to maintain
- Changes does not affect whole system
- Faster to build and deploy
- ...or throw away and rewrite

Small application

- Each service has it’s own database
- Code duplication vs. code coupling
- Shared code - libraries

Small application

“If service is bigger than your head then it’s too big”

Internet

Deploy independently

- Each microservice runs in it’s own process, so deployment of one service won’t affect the whole application
- Easier to scale development
- Faster feature releases
- Less downtime
- Develop, build and deploy!

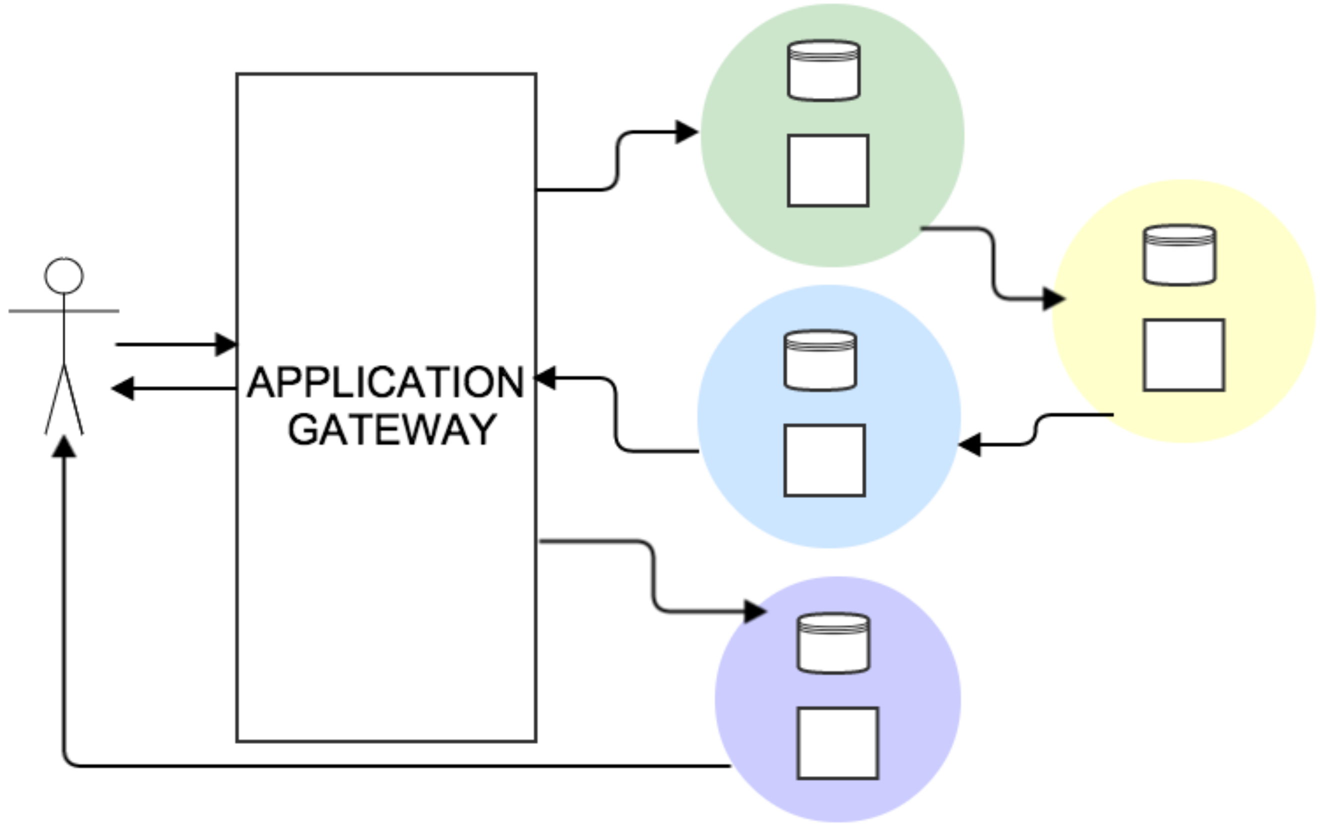
Flexible solutions

- Modular
- Polyglot data persistence
- Multi-framework

Right tool for the job

Application: two layers

- System layer
  - gateway: defines interfaces, communication mechanisms
  - rarely changes
- Service layer
  - services with different internal architectures
  - different technology stacks
  - evolves rapidly



Application: two layers

When to use it?

- In the beginning it will slow down the development
- Later - refactoring might be painful
- It’s easier to merge services than split monolith into services
- ...unless monolith already has loosely-coupled modules

Be realistic

“Focus on building services that make development and deployment easier - not just tiny services”

Nanoservice antipattern

A nanoservice is a service whose overhead (communications, maintenance, and so on) outweighs its utility.

How services communicate?

- HTTP/REST
- AMQP for asynchronous requests
- Event Sourcing
- Streams

Databases

- DB instance per service
- Relational databases, NoSQL, others

How to start?

- ESI (Edge Side Includes)
- RabbitMQ
- Gearman
- PHP multithreading

Shared data problem

- ServiceA needs to read data which is managed by ServiceB

Solution A

ServiceA calls ServiceB for data

- Benefits
  - quick implementation
  - data is always fresh
- Drawbacks
  - slows down ServiceA
  - ServiceB might be down at the moment

Solution B

Data replication

- Benefits
  - availability
  - speed
- Drawbacks
  - data replication overhead

Performance

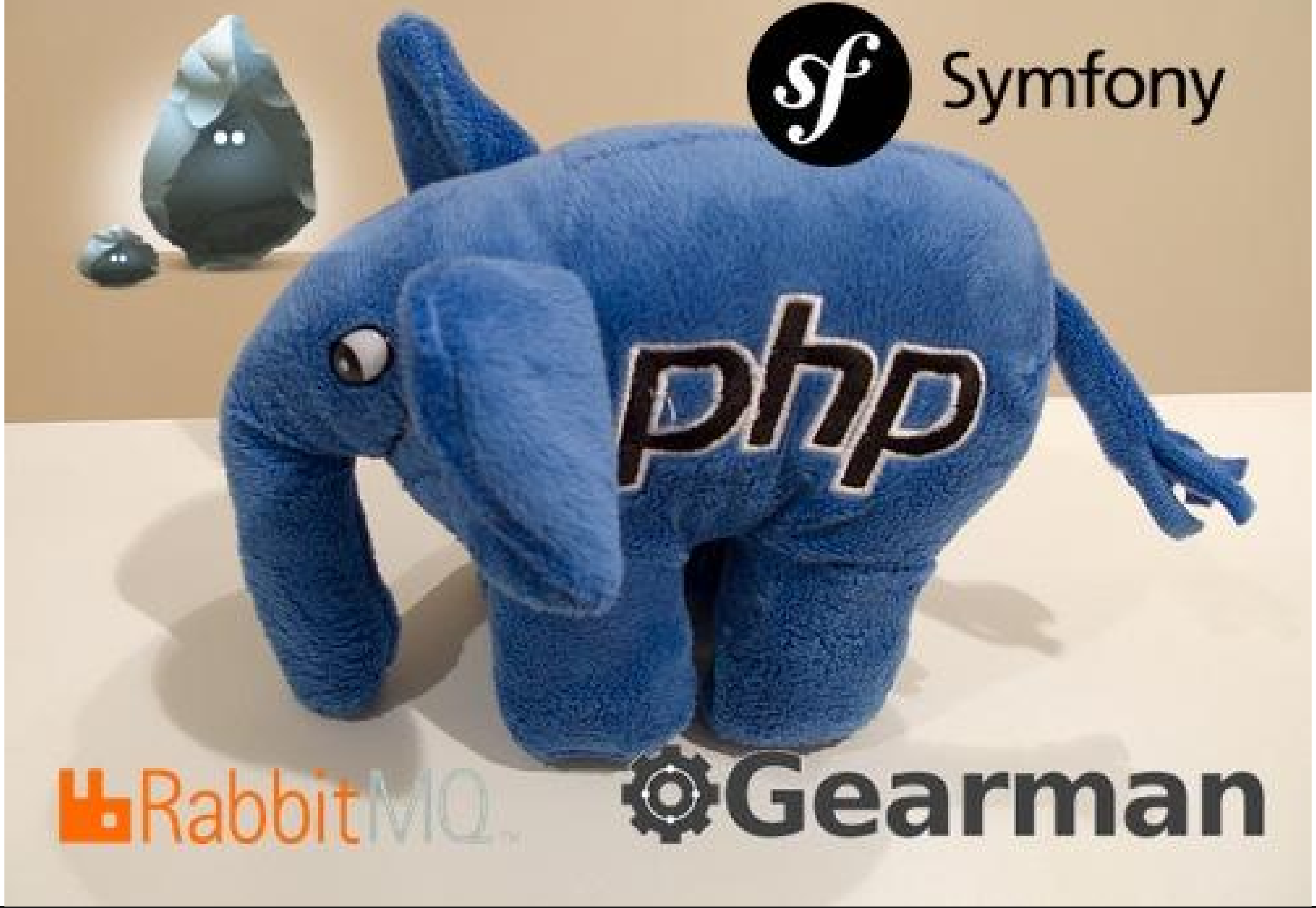
- Latency is your foe
- Everything done asynchronously - no problem
- Keep communication between services as effective as possible. No chit-chats.

Drawbacks

- High level of distributed complexity

Automation

- Automate everything
  - CI
  - deployments
  - configuration
  - error logging
  - monitoring



PHP world

Symfony2 app as service

- Symfony2 isn’t heavy…
- ...if you know how to circumcise it
- Avoid standard edition
- Create your own minimal application

  http://www.whitewashing.de/2014/04/24/symfony_hello_world.html

Symfony2 benefits

- **HttpKernel** component is one of the greatest things happened in PHP world in years
- **SF2 DIC**: flexible and extendable way to grow your project
- **Console** component for CLI utilities
- Standardised solutions

Symfony2 as gateway

- Basic SF2 application with dumb controllers for routing services
  - via messaging
- Rendering main views for ESI

Challenges

- Define service boundaries
- Continuous Integration
    - Continuous Deployment
- Error logging
- Monitoring
- System tests
    - Consumer tests

More challenges...

- Security layer
- Shared configuration
- Shared assets
- Graceful degradation

Who is using

amazon

~120 services to generate 1 page
http://highscalability.com/amazon-architecture

NETFLIX

Has 600+ services in total
http://techblog.netflix.com/

ebay

http://www.ebaytechblog.com/

Why microservice?

- Scale development
- Scale your application
- Application availability
- Use right tools for the job
- Whole system becomes faster if done right

Dig more

- Martin Fowler http://martinfowler.com/articles/microservices.html
- Fred George https://www.youtube.com/watch?v=2rKEveL55TY
- http://blog.arkency.com/2014/07/microservices-72-resources/

What’s next?

Reactive architecture?

http://www.reactivemanifesto.org/

Questions?

Join us

estina

hello@estina.com