Microservices

Pain & Gain

Helmes

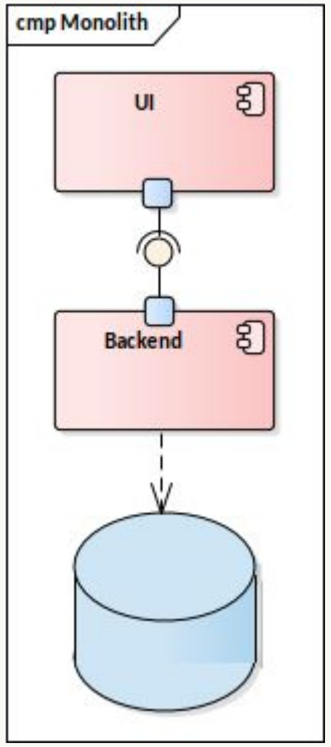
800+ experts	**40m** annual turnover	**28+** years
Tallinn Estonia (HQ)	**Minsk, Brest** Belarus	**San Diego** USA

Monolith

Helmes

Intro into monoliths

- The most simple architecture
- Front-end can be separate or built-in
- Scaleable by adding more instances
- Often the best first architecture
- The best solution for simple projects



Helmes

Monoliths - gain

- Easy to
  - develop
  - test
  - deploy
  - scale

Helmes

Monoliths - pain

- Difficult to understand (the codebase)
- Slow IDE
- Slow web container
- Hard to do CI
- Scaling the application is not efficient
- Scaling the development is hard
- “Married” to initial framework

Helmes

Monoliths - gain from experience

- Having tools and helper classes, like validation rules, available for the entire system
- Effortless to kick-start a project
- System-wide refactoring is easy using IDEs
- Debugging errors is easy - step the entire flow

Helmes

Monoliths - pain from experience

- Deployment time
- Start-up time
- Original encapsulation is breached creating unwanted dependencies
- Increased complexity for the developer
- Effect of the change
- Broken windows theory manifestation

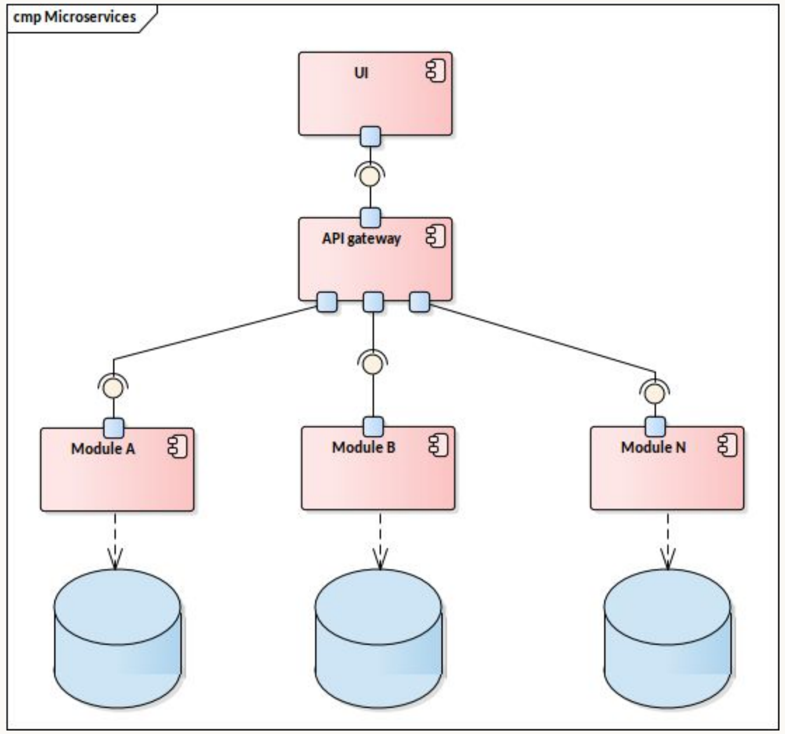
Helmes

Microservices

Helmes

Intro into microservices

- The most complex architecture
- Scaleable by adding services that need to be scaled
- The front-end can be a monolith or composed of stand-alone micro-frontends
- There are only specific use cases when you really need this approach



Helmes

The 8 fallacies of distributed computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

Helmes

Frameworks

- Spring Cloud
- Netflix stack
  - Zuul
  - Eureka
  - Archaius
  - Hystrix
  - Ribbon
- Consul
- GCP, AWS, Azure, Cloud Foundry / Serverless etc.

Helmes

Microservices - gain

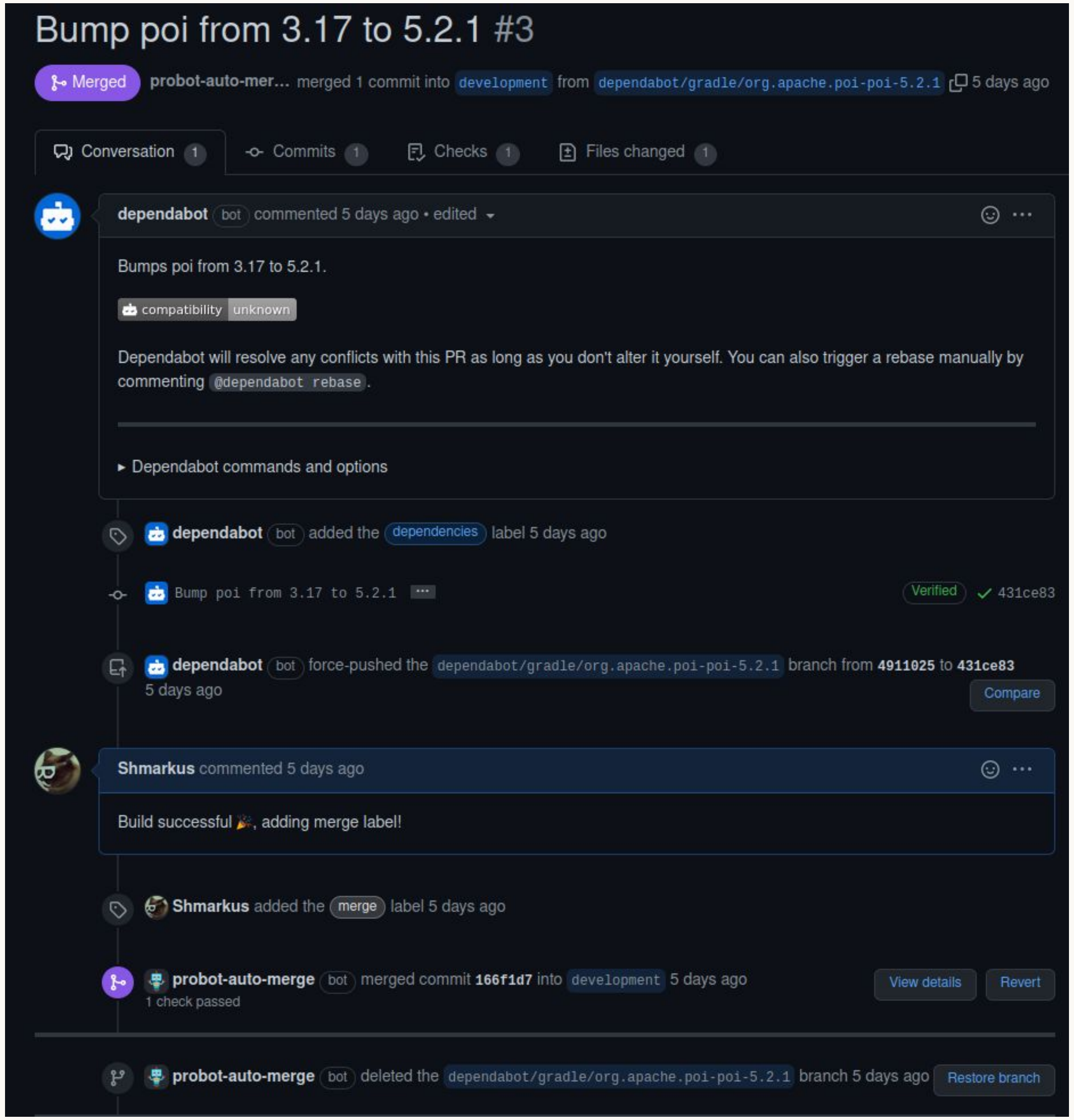
- (the codebase) is easy to understand
- Fast IDE
- Fast web container
- Easy to do CI
- Scaling the application is efficient
- Scaling the development is easy
- Improved separation of concerns
- Easy(er) to change underlying frameworks

Helmes

Microservices - pain

- It's hard to do system wide/multiple services
  - development/refactoring
  - testing
  - deployments
- inter-service communication mechanisms overhead
- accept partial failures
- increased memory consumption
- operations overhead
- keeping versions up-to-date (use bots)
- vendor lock for cloud-native approach

Helmes


Bump poi from 3.17 to 5.2.1 #3

probot-auto-mer... merged 1 commit into `development` from `dependabot/gradle/org.apache.poi-poi-5.2.1` 5 days ago

dependabot bot commented 5 days ago • edited

Bumps poi from 3.17 to 5.2.1.

compatibility unknown

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

▸ Dependabot commands and options

dependabot bot added the dependencies label 5 days ago

`Bump poi from 3.17 to 5.2.1` Verified ✓ 431ce83

dependabot bot force-pushed the `dependabot/gradle/org.apache.poi-poi-5.2.1` branch from 4911025 to 431ce83 5 days ago

Shmarkus commented 5 days ago

Build successful 🎉, adding merge label!

Shmarkus added the merge label 5 days ago

probot-auto-merge bot merged commit 166f1d7 into `development` 5 days ago
1 check passed

probot-auto-merge bot deleted the `dependabot/gradle/org.apache.poi-poi-5.2.1` branch 5 days ago

Microservices - gain from experience

- Isolation of change / Single responsibility
- Testability of the change
- Fast builds
- Small downtime on deployment
- Release faster with smaller changes

Helmes

Microservices - pain from experience

- Refactoring something that’s beyond single module
- Configuration management
- Debugging problems
- Inter-service communication
- Added complexity with additional middleware
- The distributed monolith problem
- (System) design is utmost important

Helmes

Transformation

Helmes

Intro into the transformation

- It’s a rare opportunity to start with a microservice project from scratch
- More and more monoliths are split into microservices
- This is a normal step in the application evolution
    - the sooner this transformation is initiated, the better

Helmes

An example from retail chain

- Started with MVP, 2 weeks (JHipster monolith application)
- Ideas changed, a new monolith system was introduced
- Scope got larger and new independent services were planned so the existing monolith was generated as a microservice project. The second monolith was refactored to be a microservice in the main project.
- Lots of boilerplate code generation helped a lot (OpenAPI + JHipster)

Helmes

An example from government backoffice

- 6 years old monolith refactoring project
- Consisted of 6 separate domains that were seemingly not related
- Overuse of ORM functionalities tangled the whole bundle together
- The transformation phase is hard, since for some time you have the worst of both worlds
- When transforming older applications, the underlying frameworks usually need upgrading as well which adds another problem to the mix

Helmes

An example from legendary SKAIS2

- Started in 2014 as a monolith application
- Constant merge conflicts between different teams due to same codebase
- Codebase got very fast very big and complex
- ~2017 refactoring to services
- Extraction of supporting modules such as authentication, person, parameter, common and x-road using the strangler pattern
- Adding new services independent of the monolith
- Migrating from REST services to MQ based communication

Helmes

Key takeaways

- There’s no silver bullet
- Don’t start with microservices “just because”
- Get to know helping tools and frameworks like log aggregation, distributed tracing, service discovery and central configuration
- Make constant design decisions, avoid distributed monolith
- Understand whether the client is ready for the operations overhead/server overhead

Helmes

Q/A + discussion

Helmes

Useful links

- https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- https://ably.com/blog/8-fallacies-of-distributed-computing
- https://microservices.io/patterns/microservices.html
- https://micro-frontends.org
- https://netflix.github.io/
- https://www.serverless.com/

Helmes