Micro Services

Sebastian Mancke

↪ tarent

Monolythic Software

↪ tarent

↪ Why are monolythic systems 'evil' ?

Because of their dependencies:

- Software is not easy to test
- And hard to refactor
- Effects of changes can not be isolated
- Working with multiple developers/teams is challenging
- No reuse of functionality
- Runtime and deployment dependencies:
  - Performance and scaling
  - Deployment of features and releases

Avoid a Monolyth

↪ tarent

↪ **Monolyths arise from bad design,**
independent of the technology!

- You can build a monolyth with every software framework.
- Even distributed systems with a lot of services can be monolythic.
- And even software with monolythic builds and deployments may have a good internal structure.

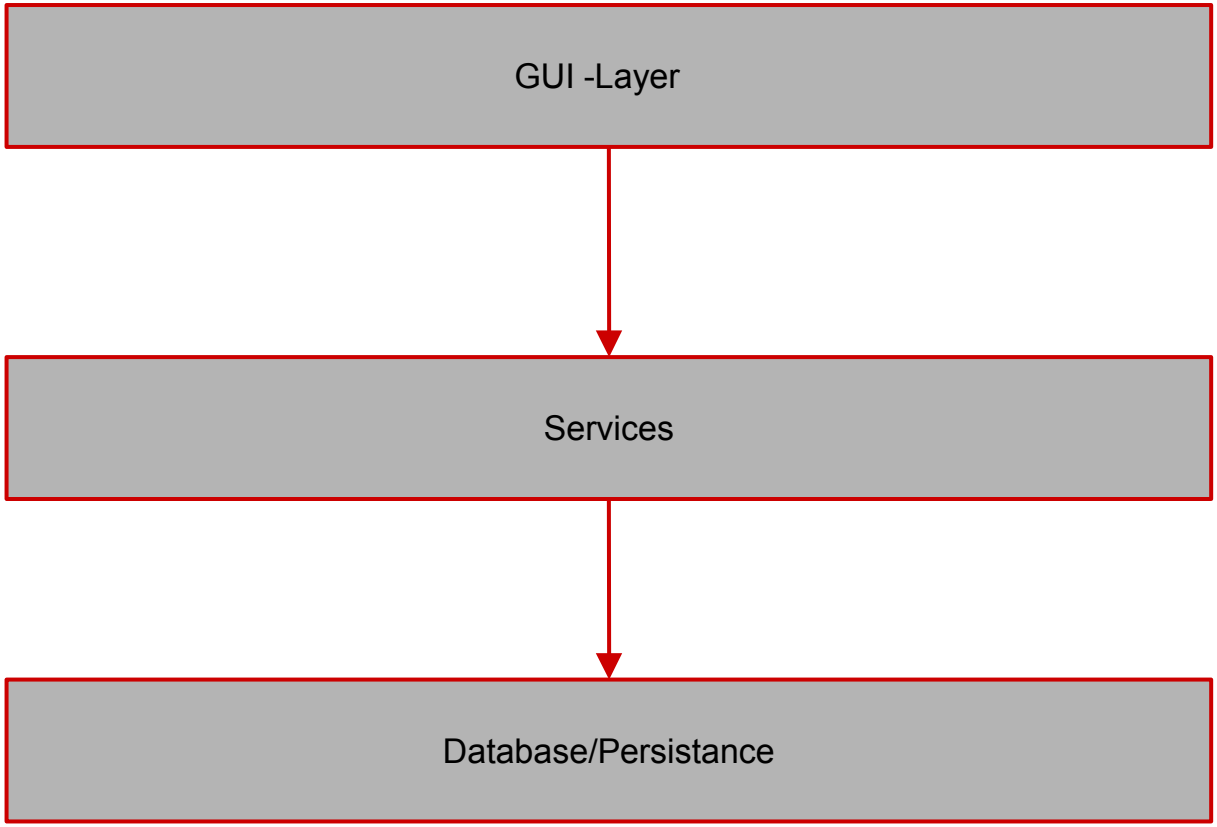
↪ **So, chosing a popular micro services framework is not enough!**

Think Vertical!

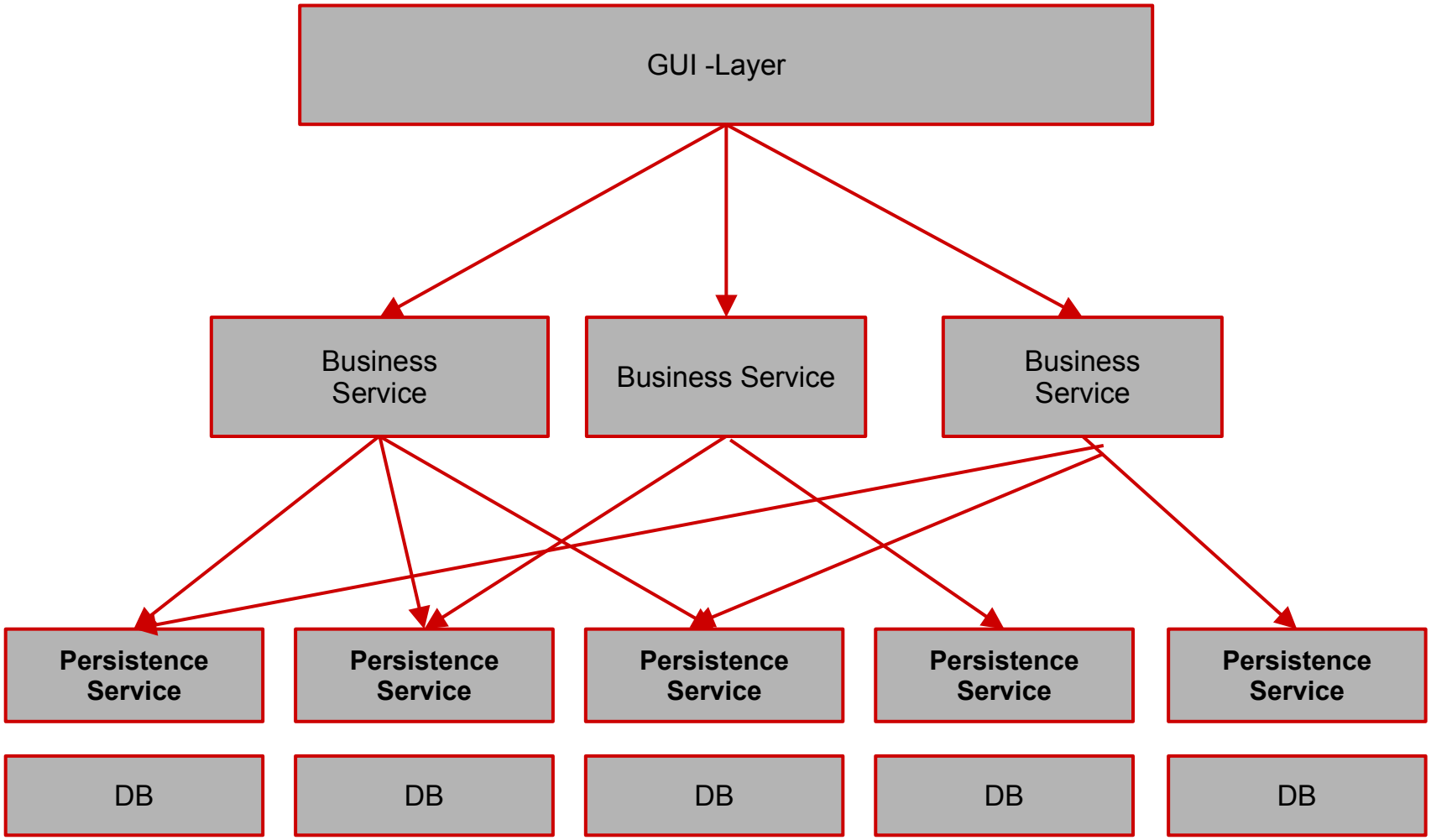
↪ tarent

- Split the application in functional modules
- Maximal reduction of dependencies between different functional parts
- Vertical teams (End-to-End)

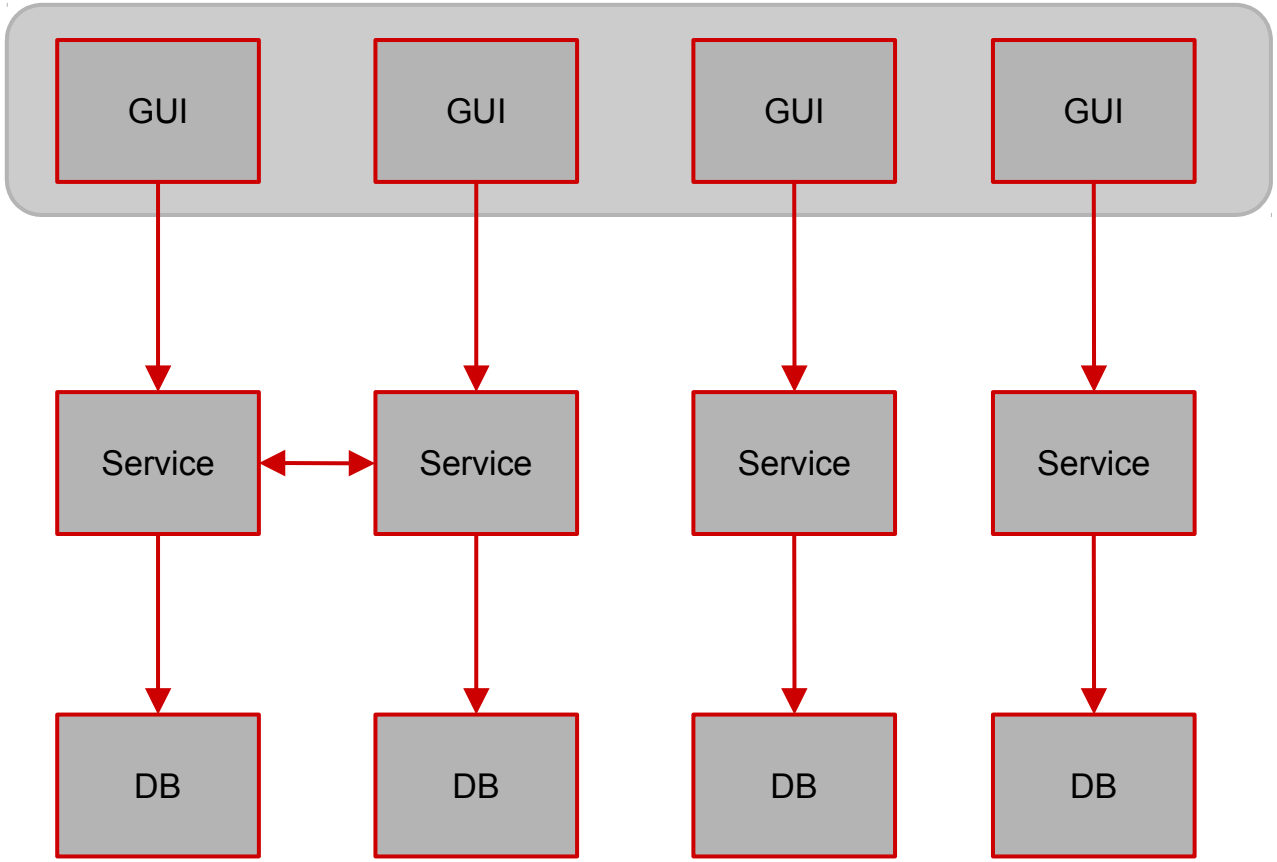
Classical Approach



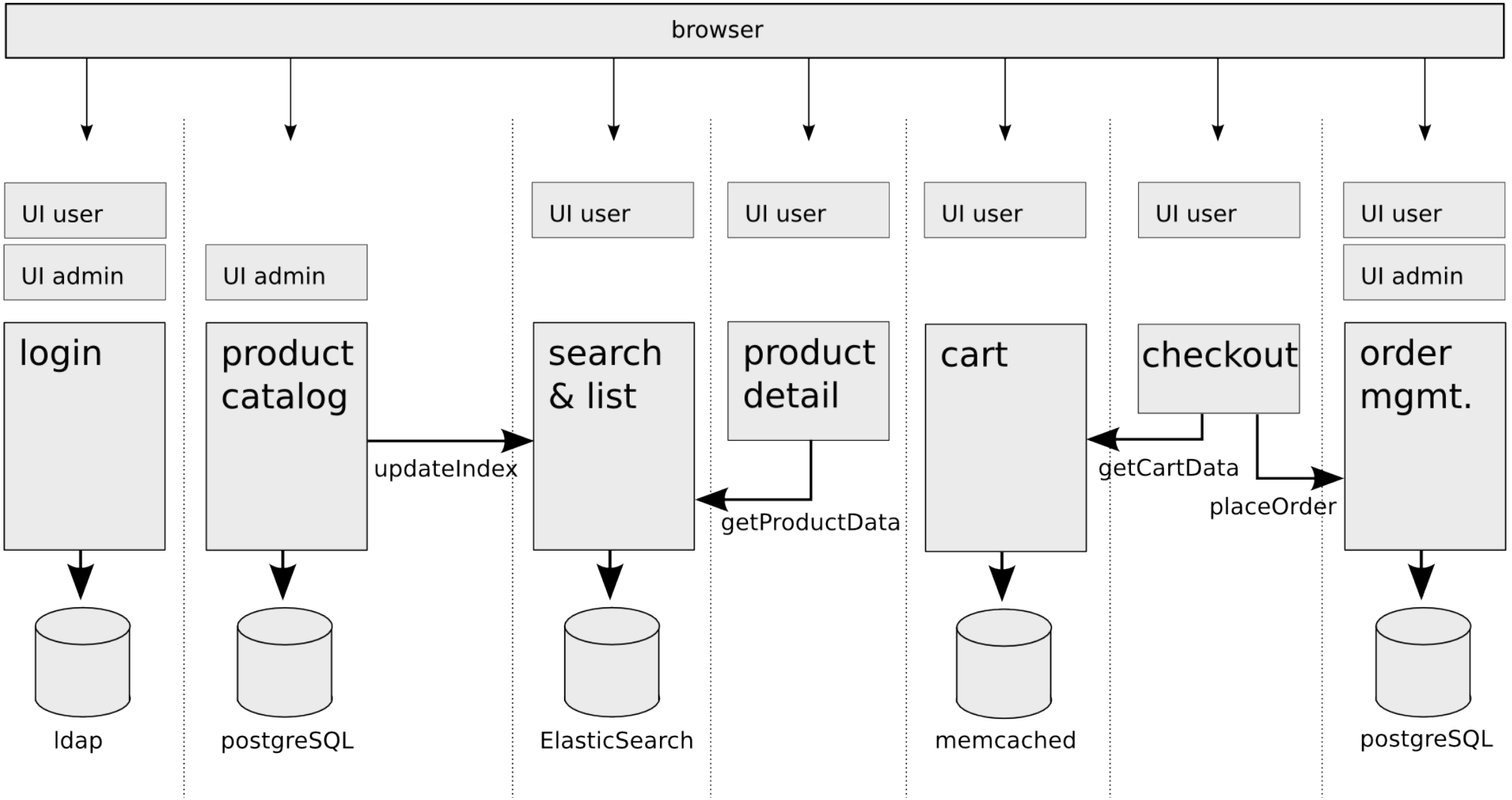
SOA Approach



The Micro Services Way...



Shop Example



↪ tarent

Micro Services

Micro Services Principles 1/2

↪ tarent

by James Lewis:

Small with a single responsibility

- Each application only does one thing
- Small enough to fit in your head
  - "If a service is bigger than your head, than it is too big"
- Small enough that you can throw them away
  - Rewrite or Maintain

Micro Services Principles 2/2

↪ tarent

by James Lewis:

Located in different VCS roots

- Each application is completely seperate
- Domain Driven Design / Conway's law
  - Domains in different bounded contexts shoud be distinct - and it is ok to have duplication
  - Use physical separation to enforce this
- There will be common code, but it should be library and infrastructure code
  - Treat it as you would any other open source library
  - Stick it in a nexus repo somewhere and treat it as a binary dependency

Implementation

↪ tarent

No application servers

- Every service runs in it's own process
- Every service brings it's own environment

Choose the right stack for the requirements

- 1 monolyth → 1 stack, 100 Micro Services → flexibility
- Free choice of: OS, language, framework, database, ..
- But: Be careful!

New feature, new service?

- At first check, if a feature should build a new functional module
- Only in the second step extend an existing service
- **Rule:** Merging services is easy, splitting is hard!

Java Frameworks

↪ tarent

- Spring Boot
- Dropwizard
- Vert.x

Database

↪ tarent

↪ **Design goal:**

Every service should have it's own exclusive database

Strategies

- NoSQL / document oriented design
- Treat foreign keys as REST URI references
- When a service needs external data: Call a service
- Don't fear data redundancy
- Replication of data: Pulling feeds with changelogs

Tradeoff solutions

- Multiple schemas within the same database
- Read-only views for data of other services
- Use DB features for replication (e.g. database link)

UI

↪ tarent

↪ **Design goal:**

Services should provide their UI themselves

Strategies

- Every service serves the full page, including layout and menu bar
- Commitment on one CSS naming schema
- Central asset service (menu, styles, common resources)
- Single page apps only within one service
- GUI composition only on the client (in the browser)
- Use UI fragments / widgets when embedding data of another service

Security

↪ tarent

Problem: The security context is spread over 100 services

Solution: Identity Management System

- Identity Management is also a service module (or even multiple)
  - Service for management of identities
  - Service for login/logout
  - Service for self administration
- OAuth2 allows distribution of the login to different services

Login implementations

↪ tarent

Variant a: Shared Cookie

- All services are available under the same domain
- The login service creates a cookie available to all others
  - Username, timestamp, rolles/permissions
  - Crypted and signed
- All services can verify the cookie by checking the signature

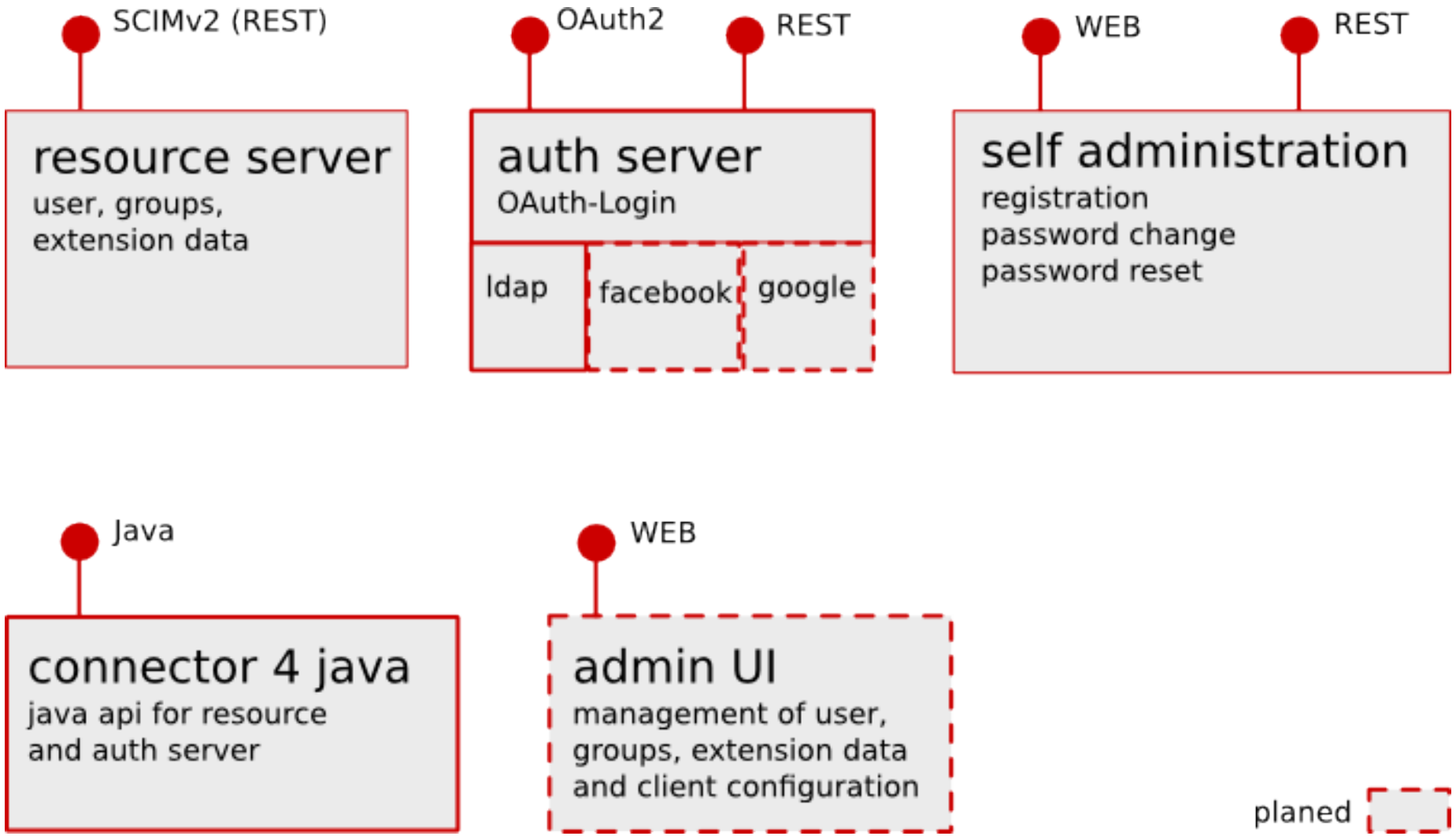
Variant b: Independent Applications

- Every service maintains it's own session
- Login is done by OAuth2
  - Double redirect
  - Token exchange
- The login service maintains a sessions as well
- Multiple logins are done transparent to the user

OSIAM

↪ tarent

https://github.com/osiam/



Communication

↪ tarent

↪ **Everything is allowed**

But: You should establish one standard for your platform.

Principles

- Loose coupling – services should not know about each other
- Smart endpoints, dump pipes
  - No intelligence in the communication channel
  - No ESB

↪ **REST is a good choice for many scenarios**

- Easy consumable with all languages
- Interfaces are maintainable towards compatibility
- URI references are helpful for navigation to different services and abstraction of the physical location of resources.

Communication– further principles

↪ tarent

Asynchronous Messaging

- Reliable event distribution
- High performance
- Load protection of critical services

Resilience

- Tolerance against failures
- Error recovery
- Avoid error cascades

API Versioning

- Don't do it for internal APIs!

Testing

↪ tarent

Unit Tests

- Integration tests suffice in many cases because the services are small
- Test the isolated service (Other services should be mocked)

Consumer Driven Tests

Idea: The integration tests of a service will be defined and implemented by the consumer (not by the service provider).

No release before the service passes all consumer's tests

- Test with the real expectations, not with the service specification
- Very smart concept, but hard to maintain
- Has the risk of high test-redundancy for common APIs

Deployment

↪ tarent

Contiuous Delivery

- Create a deployment pipeline
- Need to automate everything

↪ One monolyth may be easy to deploy, 100 Micro Services may not!

Packaging & Provisioning

- Usage of established standards: DEB, RPM, …
- Robust init scripts
- Configuration management: Puppet, Chef, ...

Deployment as platform

↪ tarent

↪ **1 Micro Service : 1 Linux System**

Docker

- LXC based virtualisation
- Similar to changeroot (but a lot better!)
- Slim and fast
- Based on git, so changes of the images can be tracked

For Hardliners

- Install the Micro Service by shipping and starting the system image
- No packaging
- No init scripts

Monitoring

↪ tarent

Realtime metrics

- Monitor, what currently happens
- Fast reaction to problems
- Do monitoring inside the application, not outside
- Tools: Metrics, Spring Boot Actuator

Logging

- Manual search in logs of 100 services is not possible
- Central log aggregation
- Filtering and analyses in realtime
- Tools: Logstash, Graylog2, Kibana, Apache Flume, fluentd

Risks

↪ tarent

Micro Services are a promising paradigm

↪ **But, you should always be careful with new paradigms!**

- Inventarisation of many services
- Challenging for operations
- High network load (if not done right)
- New way of thinking
- The freedom of technology selection may lead to chaos, if it is no governance

↪ tarent

Thank You!