MONOLITH & MICROSERVICES

Examples on how to redesign you existing monolith to microservices while keeping it operational using real life experience.

BERGLER
SOFTWARE SOLUTIONS

WHO ARE WE?

PATRICK VROEGH &
PATRICK BES

Total population of Bergler

+/- 15% patrick's

40+
Years of experience

TLC

Microsoft

BENEFITS MICROSERVICES

Short recap, do we really want microservices, and if so why?

FLEXIBILITY

With microservices, you can add new components to the system painlessly or scale services separately from one another.

AUTONOMOUS

Microservices grant the developers more independence to work autonomously and make technical decisions quickly in smaller groups. So, if the solution you're developing is expected to be large, be sure to consider the microservice architecture.

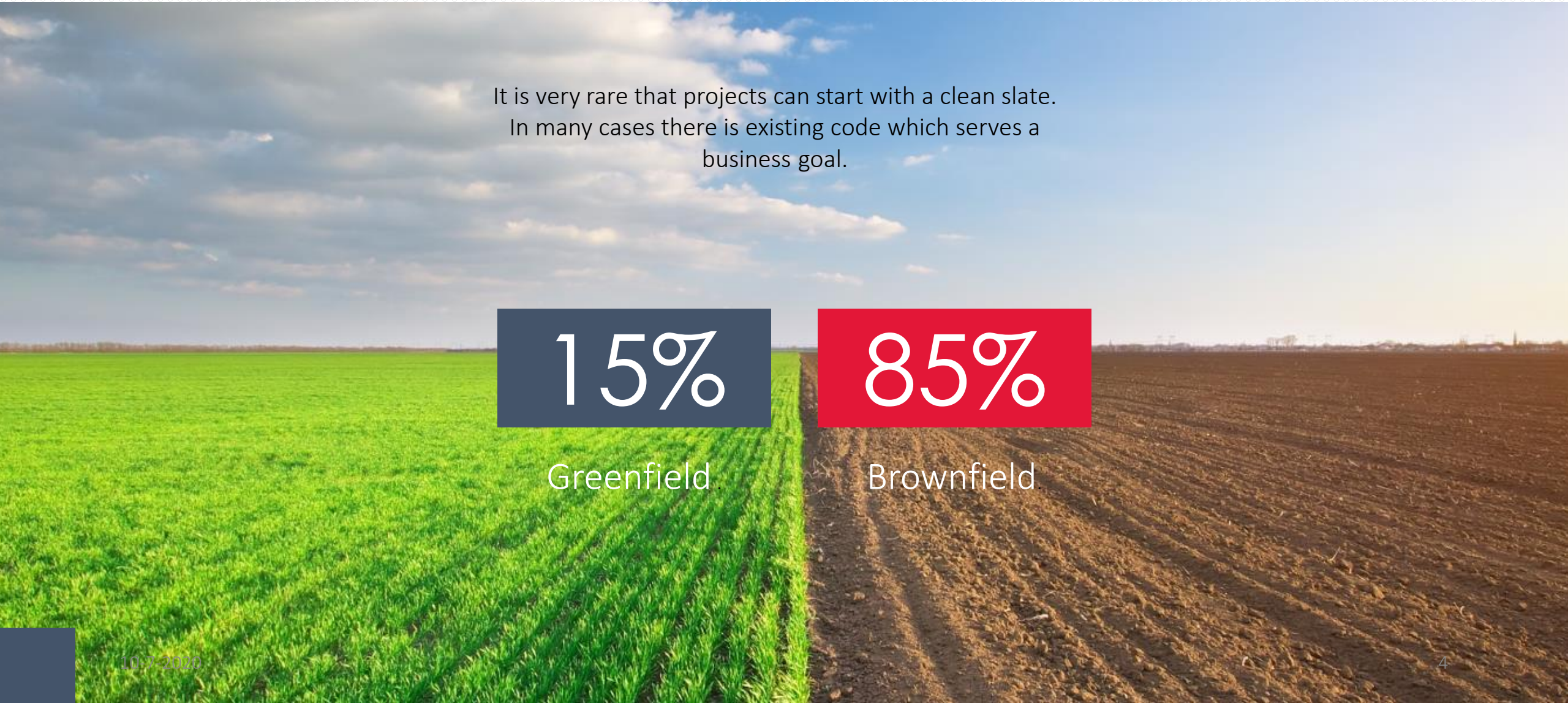
The key principle of microservices is simplicity. Applications become easier to build and maintain when they're split into a set of smaller, composable fragments.

EASY TO BUILD AND MAINTAIN

Microservices architecture tackles the problem of productivity and speed by decomposing applications into manageable services that are faster to develop.

IMPROVED PRODUCTIVITY SPEED

GREENFIELD VS BROWNFIELD STATISTICS



It is very rare that projects can start with a clean slate. In many cases there is existing code which serves a business goal.

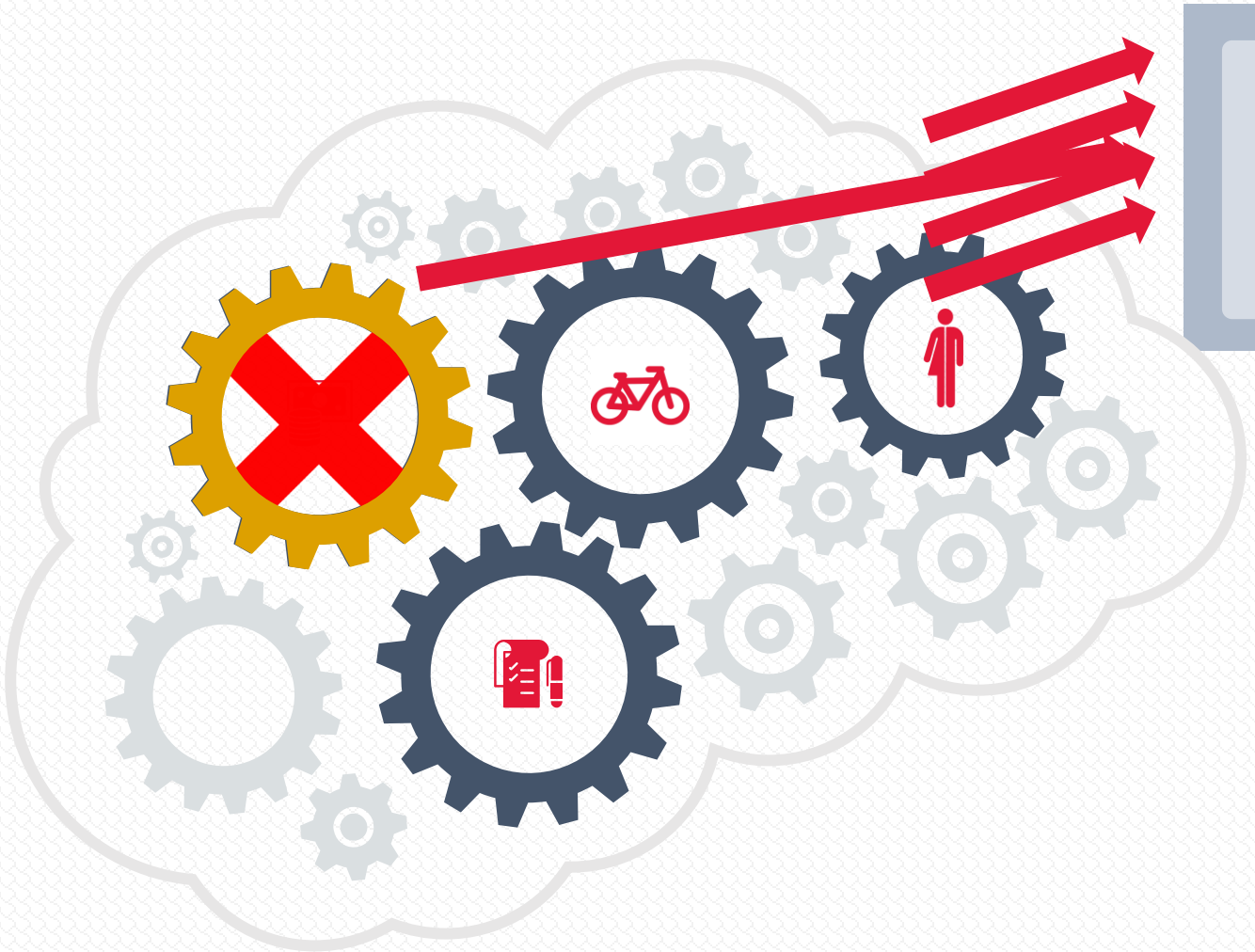
15%

Greenfield.

85%

Brownfield.

MONOLITH DISENTANGLEMENT



Sales service

Product service

Person service

Production service

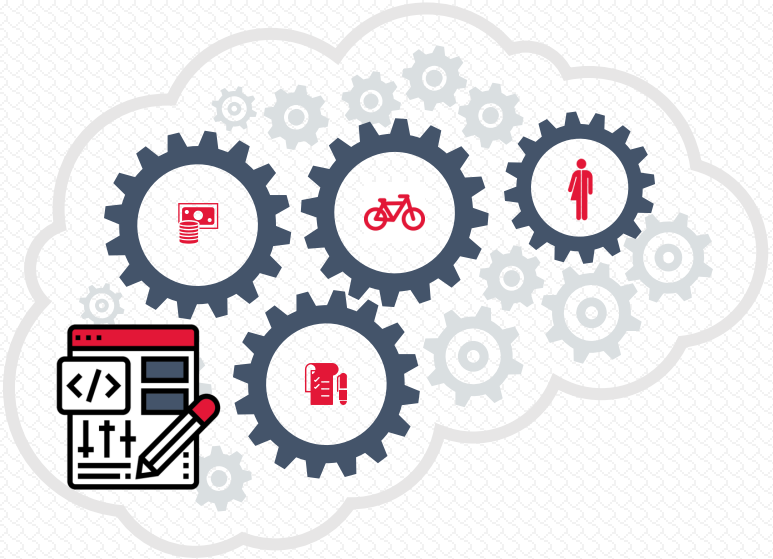
POSSIBLE PATTERNS

POSSIBLE MONOLITH DISENTANGLEMENT PATTERNS

- Big Bang Pattern
  - Big bang pattern where we switch to the new implementation at one specific moment in time.
- Strangler Pattern
  - The Strangler Pattern is a popular design pattern to incrementally transform your monolithic application into microservices by replacing a particular functionality with a new service. Once the new functionality is ready, the old component is strangled
  - ACL
    - Implement a façade or adapter layer between different subsystems that don't share the same semantics. This pattern was first described by Eric Evans in *Domain-Driven Design*.
  - Feature Flags
    - Feature Toggles (often also refered to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code.
  - Database patterns
    - One big database is probably not the way to go, but how do we handle database decomposition?

BIKESTORE CASE

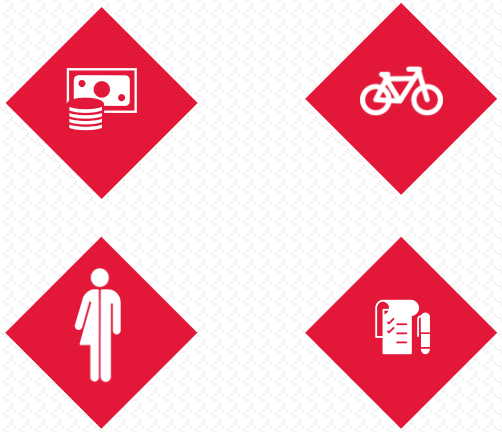
THE MONOLITH



- All business logic in one solution
- Monolithic database
- ASP.NET MVC 5 Application

Transition

THE MICROSERVICES



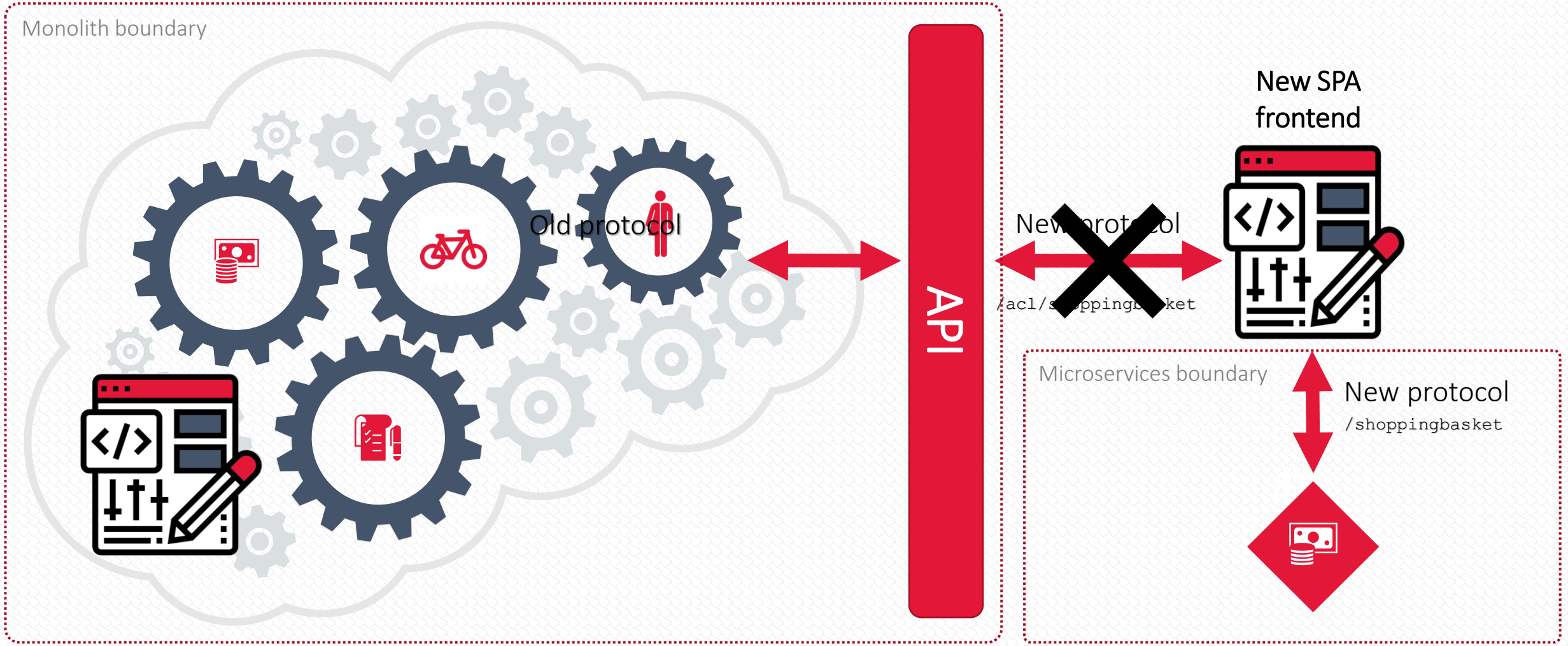
- Businesslogic separated into loosely coupled microservices
- No shared databases
- ASP.NET Core Services
- Blazor WebAssembly SPA

ANTI CORRUPTION LAYER

DRAWING THE LINE

ANTI CORRUPTION LAYER

DRAWING THE LINE

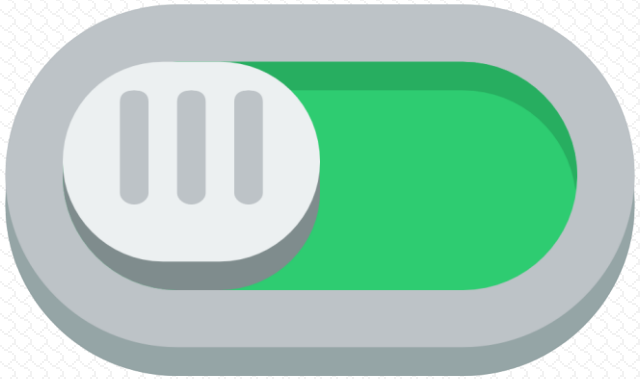


DEMO ACL


Example on how you cloud implement a new frontend, which still communicates to the old backend without sharing the semantics.

FEATURE FLAGS

SWITCH FUNCTIONALITY ON/OFF



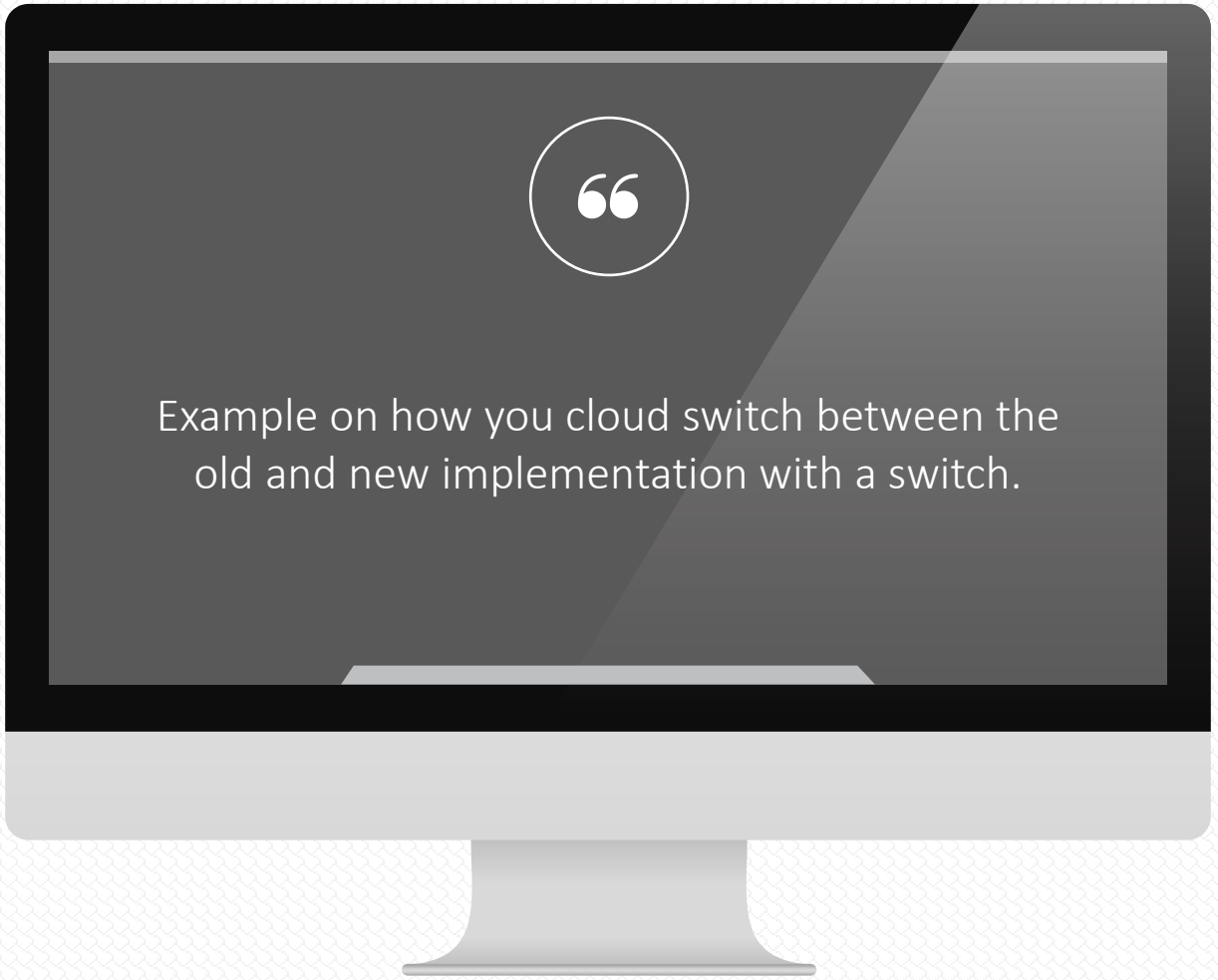
FEATURE FLAGS

- 

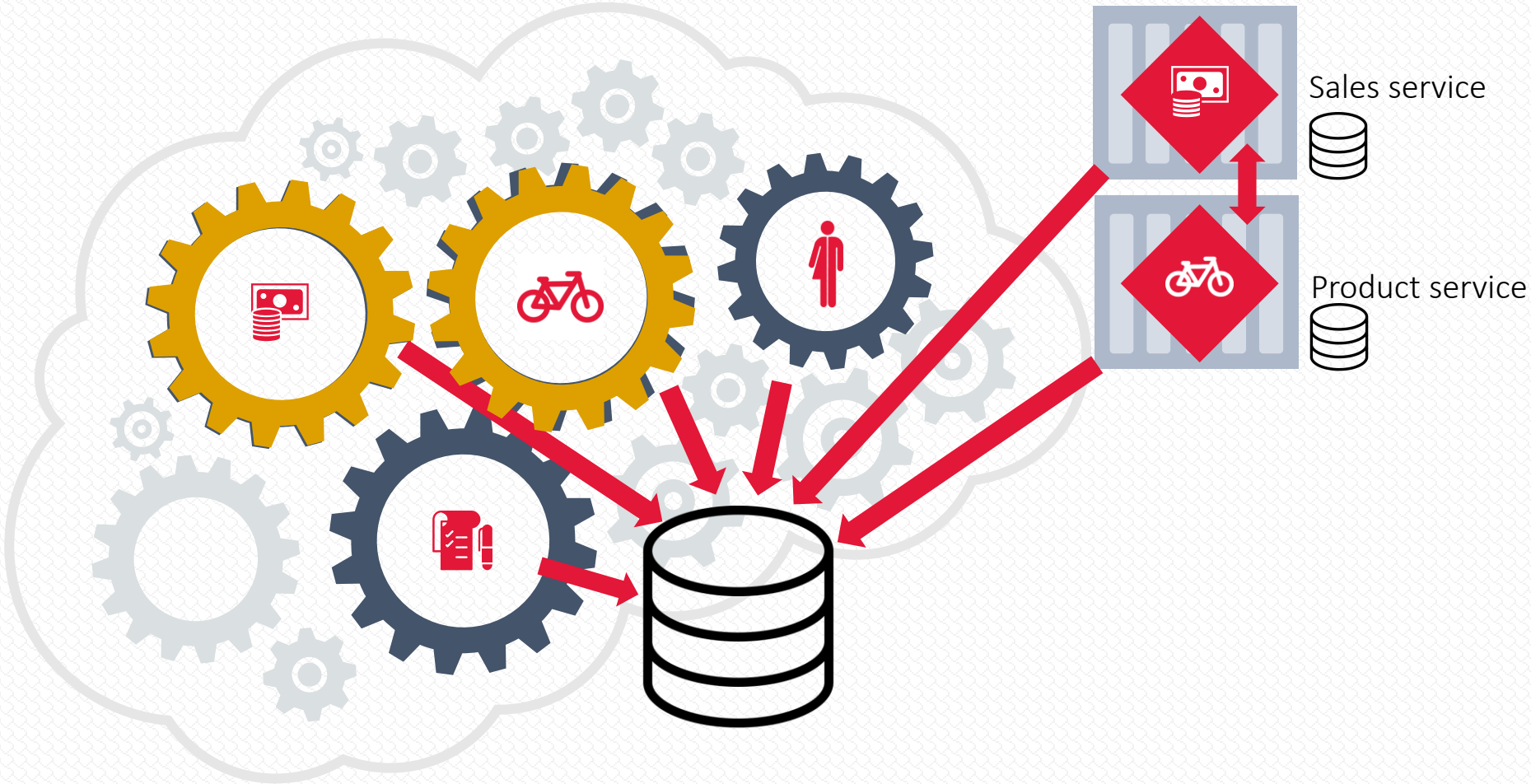
- 

- 
Azure App Configuration

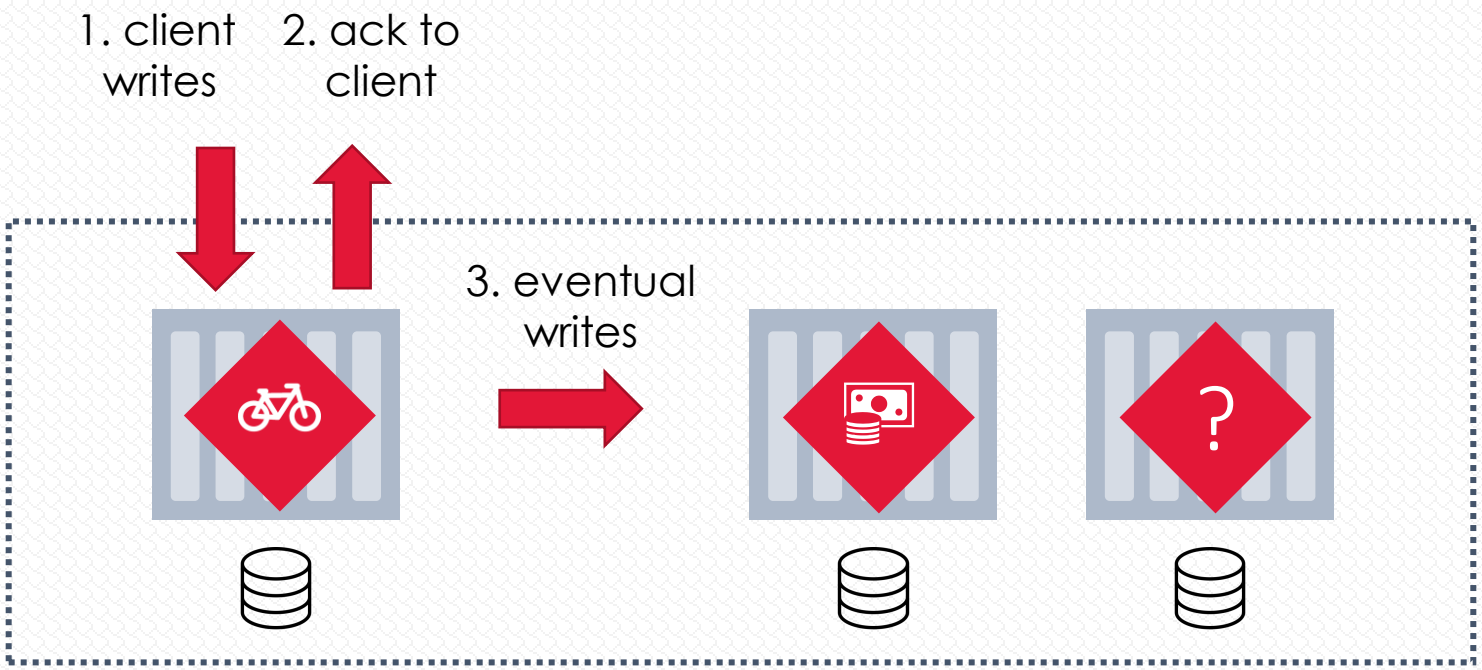
DEMO FEATURE FLAGS



DATABASE PATTERNS



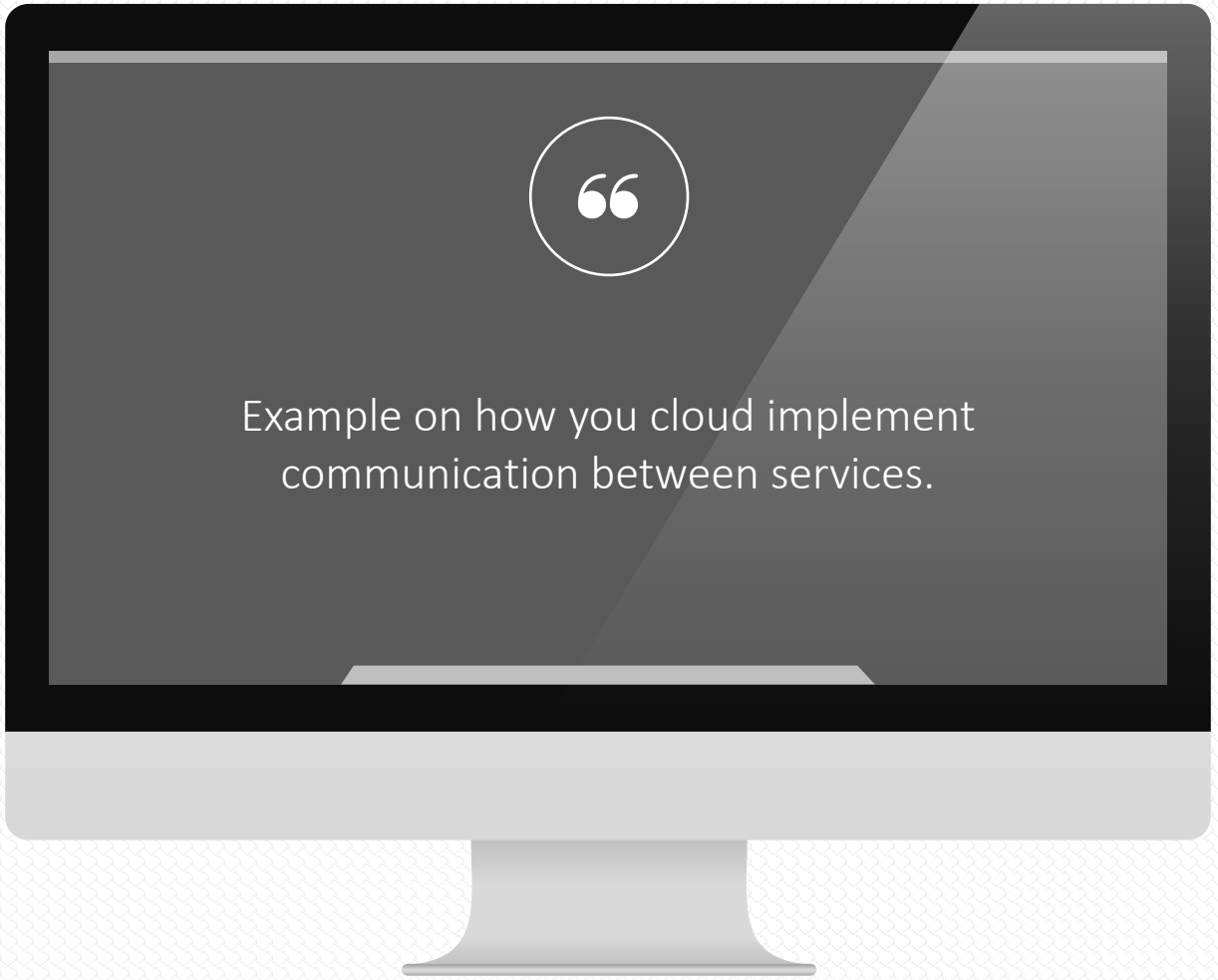
EVENTUAL CONSISTENCY



MESSAGEBUS SCENARIO'S

RabbitMQ

DEMO EVENTS


Example on how you cloud implement
communication between services.

BERGLER
SOFTWARE SOLUTIONS

SOURCES & MORE INFO

CONTACT US AS PVROEGH@BERGER.NL OR PBES@BERGLER.NL

THANKS

BUILDING MICROSERVICES – Sam Newman

MONOLITH TO MICROSERVICES – Sam Newman

DOMAIN-DRIVEN DESIGN – Eric Evans