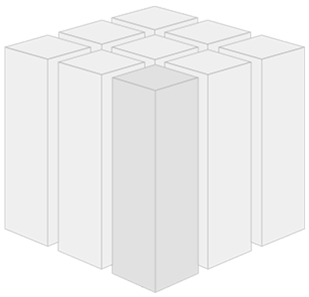

Self-Contained Systems

ASSEMBLING SOFTWARE FROM INDEPENDENT SYSTEMS

Frequently Asked Questions

Why the term “SCS”?

“Self-Contained System” describes best what’s at the core of the concept: Each system should work by itself. Together they form a “System of Systems”. We also haven’t seen the term used anywhere else, allowing us to capture it to mean one specific, hopefully well-defined thing.

Is the SCS architecture’s intent to replace other concepts?

No. For example, the SCS approach can be combined with finer-grained microservices — see SCS vs. Microservices.

How do I know which systems to build?

Each SCS is responsible for a part of the domain. Dividing the domain into bounded contexts and understanding their relationships is what we refer to as *domain architecture*. Ideally, there is one SCS per bounded context.

SCSs are very isolated — how can they still form one system?

The goal of the SCS approach is to localize decisions. However, to make a system of SCSs work together, some decisions affecting all of them need to be made. We can distinguish:

- Local decisions in one SCS are called *micro architecture*. This includes almost all technical decisions e.g. the programming language or the frameworks.
- Decisions that can only be made on the global level are called *macro architecture*. Actually, only very few things fall into this category, most importantly the protocol SCSs can use to communicate with each other, the approach used for UI integration, and possibly a way to do data replication.

Of course it is still possible to decide parts of the micro architecture on the global level. For example, you can decide that all SCSs have to use the same programming language, frameworks, or other technical dependencies. Whether or not you do this is a matter of your organization’s culture and the flexibility each SCS team is supposed to have. However, you should avoid making mixing up decisions for the SCSs’ micro architecture with those made at the macro architecture level.

Each SCS has its own ideas and technology. Isn’t that bound to end in chaos?

See above — you can limit the decisions each SCS can make by defining common decisions for all SCSs. Note, however, that this contradicts the idea of independent decisions. You need to find some compromise. Defining a common technology for all SCS might not be a bad idea, e.g. it makes refactoring across SCS easier and developers can work on more than one SCS at the same time. On the other hand, you want to make sure you don’t lose the flexibility to change your mind later for one or more SCSs only. If you’re unable to do that, you get the negative effects of a monolith again.

What about Mobile Clients or Single Page Apps (SPA)?

The SCS approach focuses on Web UIs that are split across the SCSs. However, each SCS might also provide an API that can be accessed by a single mobile client or a monolithic single page app. If you do this, the functionality is split across the SCSs containing the logic and the mobile client or single page app containing the UI. That way a change cannot be confined to an SCS, but would affect the SCS as well as the mobile clients and the single page app. This problem can be mitigated by modularizing the mobile client or the SPA so that changes to the UI are simplified.

Even then the mobile client or SPA remains a deployment unit by itself, i.e. in addition to the SCS also the mobile client / SPA must be redeployed and deployment of the client / SPA must be coordinated with the deployment of the SCS. Essentially, this approach violates no shared UI.

We believe ROCA is a good approach for web front ends for SCS because that approach makes it easy to combine UIs from several SCSs to one. You could also have one SPA per SCS, but this means switching from one SCS/SPA to another one loads another application and you cannot easily share parts of the SPA/SCS in other SPA/SCS. That said, we don't rule out SPAs as a useful approach for some of the SCSs.

Can SCSs share a database?

Multiple SCSs might use the same database – but they must not share any data. So they might have completely separate schemas in the same physical database. Sharing a schema is not allowed because that would mean schema can hardly be changed. The changes would need to be coordinated between all components that use the data and introduces a strong coupling that SCSs should avoid.

Even using separate schemas in the same database causes some coupling, e.g. an update to the database software would influence all SCSs. However, all SCS can use the same backup and disaster recovery and thereby save resources. Whether or not to connect all SCSs to the same database system or database cluster is one of the decisions that need to be made as part of the macro architecture.

How can I use this document?

This document is licensed under a Creative Commons license, i.e. you can essentially use it as you see fit, as long as you include proper attribution and share your modifications under the same license. We explicitly encourage you to recommend, compare or develop frameworks according to this style, and intend to be as open as reasonably possible while maintaining conceptual integrity. We welcome pull requests to this site: You can find the repository on GitHub.

Can I provide feedback?

Of course, please use the comments to share your thoughts. We welcome criticism as well as suggestions for improvement.