Welcome to

6.1 SOA book

KEA System Integration F2020 10 ECTS

Henrik Kramselund Jereminsen hkj@zencurity.com @kramse

Slides are available as PDF, kramse@Github

6-1-SOABOOK-system-integration.tex in the repo security-courses

This weeks Agenda in system integration

- Follow the plan:
  `https://zencurity.gitbook.io/kea-it-sikkerhed/system-integration/lektionsplan`
- Thursday 13:00 - 14:00 Meeting to discuss SOA book
- Service-Oriented Architecture (SOA) Read chapters 1-5 in the SOA book, less pages than it seems - large figures on many pages!
- Exercises in database and cloud computing, Start the hand-in assignment I

Exercises

- Run PostgreSQL
- Why go to SOA
- Cloud Computing Introduction, Cloud Deployment
- Download the Microservices ebook

Goals for this week in system integration

This weeks goals:

- Meet me in Zoom at least once, watch Fronter for more meeting times
- Get an understanding of the SOA book and SOA
- Find time to do some exercises, communicate with friends, students and instructor

I know it can be hard to find the time, with Corona news, kids etc. Do your best and stay safe, wash your hands.

Photo by Thomas Galler on Unsplash

Reading Summary

SOA ch 1-5:

- CHAPTER 1: Introduction
- CHAPTER 2: Case Study Backgrounds
- CHAPTER 3: Understanding Service-Orientation
- CHAPTER 4: Understanding SOA
- CHAPTER 5: Understanding Layers with Services and Microservices

Service-oriented architecture (SOA)

Service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. A SOA service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online. **SOA is also intended to be independent of vendors, products and technologies.[1]**

A service has four properties according to one of many definitions of SOA:[2]

- It logically represents a business activity with a specified outcome.
- It is self-contained.
- It is a black box for its consumers, meaning the consumer does not have to be aware of the service's inner workings.
- It may consist of other underlying services.[3]

Source:

`https://en.wikipedia.org/wiki/Service-oriented_architecture`

The SOA Manifesto

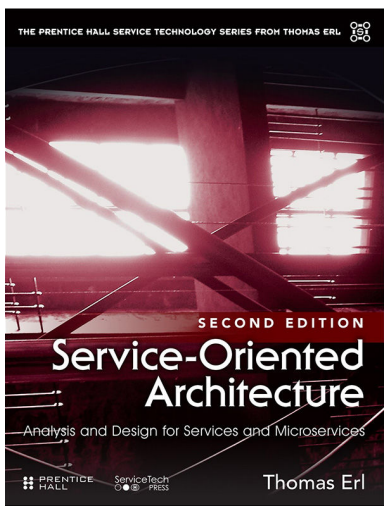
Through our work we have come to prioritize:

- **Business value** over technical strategy
- **Strategic goals** over project-specific benefits
- **Intrinsic interoperability** over custom integration
- **Shared services** over specific-purpose implementations
- **Flexibility** over optimization
- **Evolutionary refinement** over pursuit of initial perfection

Book references, in Appendix D the *SOA Manifesto* `www.soa-manifesto.org`.

I recommend reading the explanation in *The SOA Manifesto Explored*

Book: Service-Oriented Architecture



Service-Oriented Architecture: Analysis and Design for Services and Microservices, Thomas Erl, 2017 ISBN: 978-0-13-385858-7

If you go to informit.com, register and log in or create an account, you can get a substantial discount on the eBook in PDF format. Enter the product ISBN, 9780133858587 and I filled out a small survey.

Chapter 1: Introduction

- Since publishing the first edition, they published a series of 11 books of which 8 were dedicated to SOA
- Multiple parts updated from that work, so editions are different!
- Patterns have been identified and presented on `www.soapatterns.org` which now redirect to `https://patterns.arcitura.com/soa-patterns` as part of a SOA Certified Professional (SOACP) program
- Similarly for cloud computing `www.cloudpatterns.org` and Big Data `www.bigdatapatterns.org`
- Note: getting certified is not a requirement for working with SOA

Capitalization for Principles, Constraints, and Patterns

To maintain an immediately recognizable distinction between constraints, principles, and patterns throughout this book, each uses a different delimiter for page numbers. The page number for each constraint is displayed in curly braces, for each principle it is placed in rounded parentheses, and for patterns, square brackets are used, as follows:

- Principle Name (page number)
- Constraint Name {page number}
- Pattern Name [page number]

For example, the following statement first references a service-orientation design principle, then an SOA design pattern, and finally a REST constraint: "...the Service Loose Coupling (293) principle is supported via the application of the Decoupled Contract [337] pattern and the Stateless {308} constraint ..."

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

SOA in Denmark

- SOA is used in Denmark too
- Misc links:
- HVAD ER SOA? `http://arkitekturguiden.digitaliser.dk/soa/hvad-er-soa`
- Danish National Interoperability Framework (NIF) `https://joinup.ec.europa.eu/sites/default/files/inline-files/NIFO%20-%20Factsheet%20Denmark_12_2015.pdf` "Denmark describes an architecture based on ServiceOriented architecture principles and puts forward standards for Service-Oriented infrastructure."
- Jobs relating to SOA in Denmark too

Chapter 2: Case Study Backgrounds

- Very short chapter with 2 back stories for two different organizations
- You need to make one for your own assignment:
  Hand-in assignment I: Describe the system environment for an organisation

Chapter 3: Understanding Service-Orientation

Chapter 3: Understanding Service-Orientation

This chapter provides detailed coverage of the service-orientation design paradigm, including its underlying design philosophy and design principles, as well as a comparison to traditional silo-based design approaches. The chapter concludes with coverage of typical critical success factors for adopting service-orientation within organizations.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Services in Business Automation

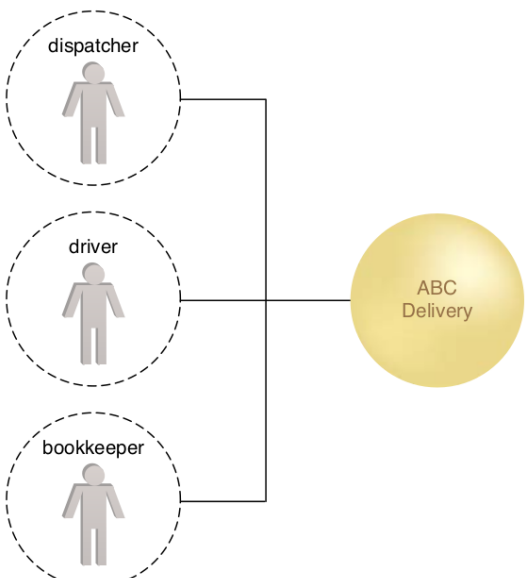


Figure 3.2
A company that employs these three people can compose their capabilities to carry out its business.

- Different capabilities are composed/combined into a business/service
- with computers we use the terms *service* and *service contract* which includes published Application programming interface (API)

Common technologies

We already mentioned some of the technologies used for this:

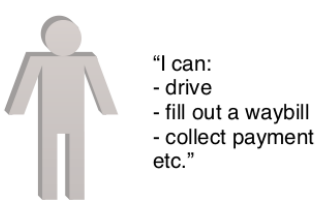
- Extensible Markup Language (XML) used in Web service (WS) with Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP)
- Allowing us to do Remote Method Invocation (RMI) aka Remote Procedure Call (RPC)
- Sometimes incorporating XML schema (XSD), Extensible Stylesheet Language Transformations (XSLT) and even producing HyperText Markup Language (HTML) documents or perhaps JavaScript Object Notation (JSON)
- Too many acronyms? Use Wikipedia!
- See the patterns and recognize common use-cases
- Or let Camel and Python convert data ☺

Services Are Collections of Capabilities

Services Are Collections of Capabilities

When discussing services, it is important to remember that a single service can offer an API that provides a collection of capabilities. They are grouped together because they relate to a functional context established by the service. The functional context of the service illustrated in Figure 3.5, for example, is that of "shipment." This particular service provides a set of capabilities associated with the processing of shipments.

Figure 3.5
Much like a human, an automated service can provide multiple capabilities.



Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Service-Orientation is a Design Paradigm

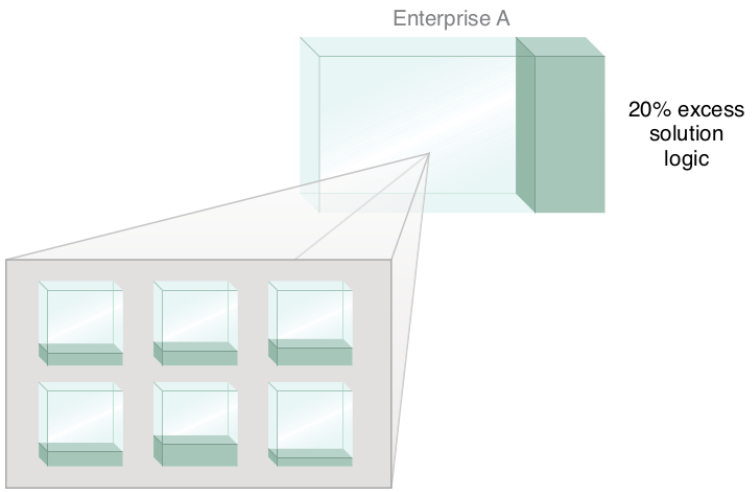
> A design paradigm is an approach to designing solution logic. When building distributed solution logic, design approaches revolve around a software engineering theory known as the "separation of concerns." In a nutshell, this theory states that a larger problem is more effectively solved when decomposed into a set of smaller problems or concerns. This gives us the option of partitioning solution logic into capabilities, each designed to solve an individual concern. Related capabilities can be grouped into units of solution logic.

- A *service composition* is a coordinated aggregate of services
- A *service inventory* is an independently standardized and governed collection of complementary services within a boundary that represents an enterprise or a meaningful segment of an enterprise.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Problems Solved by Service-Orientation

Figure 3.14
This simple diagram portrays an enterprise environment containing applications with redundant functionality. The net effect is a larger enterprise.



- Redundant functionality is costly in the long run
- Integration Becomes a Constant Challenge

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

The Need for Service-Orientation

The consistent application of the eight design principles we listed earlier results in the widespread proliferation of the corresponding design characteristics:

- increased consistency in how functionality and data is represented
- reduced dependencies between units of solution logic
- reduced awareness of underlying solution logic design and implementation details
- increased opportunities to use a piece of solution logic for multiple purposes
- increased opportunities to combine units of solution logic into different confi gurations
- increased behavioral predictability
- increased availability and scalability
- increased awareness of available solution logic

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Reusable Solution Logic

Increased Amounts of Reusable Solution Logic

Within a service-oriented solution, units of logic (services) encapsulate functionality not specific to any one application or business process (Figure 3.17). These services are therefore classified as reusable (and agnostic) IT assets.

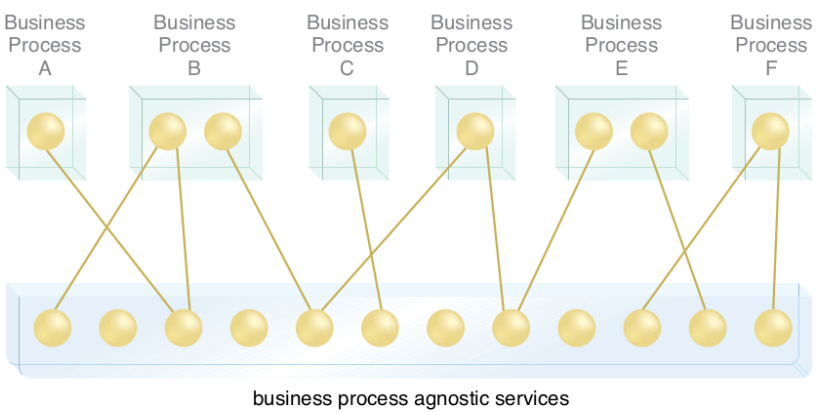


Figure 3.17

Business processes are automated by a series of business process–specific services (top layer) that share a pool of business process–agnostic services (bottom layer). These layers correspond to service models described in Chapter 5.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

3.5 Four Pillars of Service-Orientation

The four pillars of service-orientation are

- Teamwork – Cross-project teams and cooperation are required.
- Education – Team members must communicate and cooperate based on common knowledge and understanding.
- Discipline – Team members must apply their common knowledge consistently.
- Balanced Scope – The extent to which the required levels of Teamwork, Education, and Discipline need to be realized is represented by a meaningful yet manageable scope.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Chapter 4: Understanding SOA

Chapter 4: Understanding SOA

This chapter delves into the distinct characteristics and types of service-oriented architecture and further explores the links between the application of the service-orientation design paradigm and technology architecture. The chapter concludes with brief coverage of common SOA project lifecycle stages and organizational roles, with an emphasis on the service inventory analysis, service-oriented analysis, and service-oriented design phases.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

4.2 The Four Common Types of SOA

To better understand the basic mechanics of SOA, we now need to study the common types of technology architectures that exist within a typical service-oriented environment:

- *Service Architecture* – The architecture of a single service.
- *Service Composition Architecture* – The architecture of a set of services assembled into a service composition.
- *Service Inventory Architecture* – The architecture that supports a collection of related services that are independently standardized and governed.
- *Service-Oriented Enterprise Architecture* – The architecture of the enterprise itself, to whatever extent it is service-oriented.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Service-oriented technology architecture

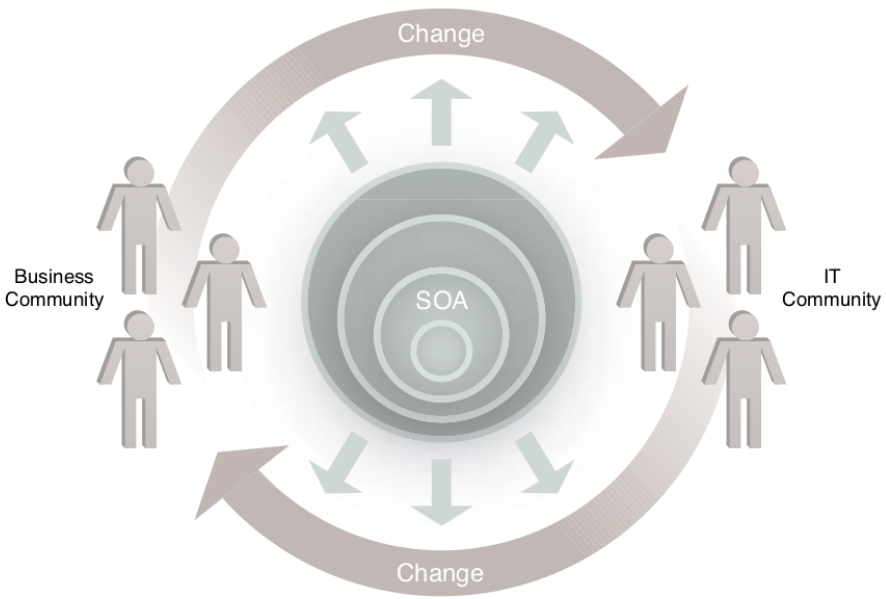


Figure 4.22
Service-oriented technology architecture supports the two-way dynamic between business and IT communities, allowing each to introduce or accommodate change throughout an endless cycle.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

SOA Project and Lifecycle Stages

SOA Adoption Planning

- Scope of planned service inventory and the ultimate target state
- Milestones representing intermediate target states
- Timeline for the completion of milestones and the overall adoption effort
- Available funding and suitable funding model
- Governance system
- Management system
- Methodology
- Risk assessment

book then continues with the rest of the SOA Project Stages

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Chapter 5: Understanding Layers with Services and Microservices

Chapter 5: Understanding Layers with Services and Microservices

This chapter provides an updated version of the standard service models and corresponding service layers. It incorporates this new content into a new service definition process with the addition of the microservice model and micro task service layer. The relevance of service deployment bundles and containerization are also briefly mentioned in relation to microservice implementation requirements.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Common Service Models

- *Task Service* – A service with a non-agnostic functional context that generally corresponds to single-purpose, parent business process logic. A task service will usually encapsulate the composition logic required to compose several other services to complete its task.
- *Microservice* – A non-agnostic service often with a small functional scope encompassing logic with specific processing and implementation requirements. Microservice logic is typically not reusable but can have intra-solution reuse potential. The nature of the logic may vary.
- *Entity Service* – A reusable service with an agnostic functional context associated with one or more related business entities (such as invoice, customer, or claim). For example, a Purchase Order service has a functional context associated with the processing of purchase order-related data and logic.
- *Utility Service* – Although a reusable service with an agnostic functional context as well, this type of service is intentionally not derived from business analysis specifications and models. It encapsulates low-level technology-centric functions, such as notification, logging, and security processing.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

End result: functional decomposing large problems

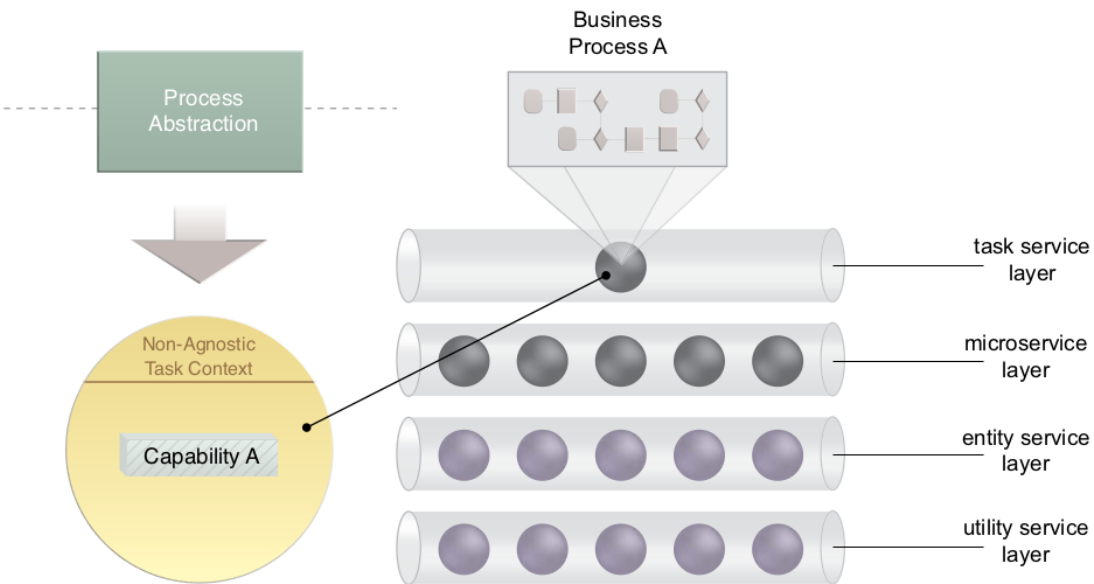


Figure 5.11

The task service represents a part of a parent service layer and is responsible for encapsulating the remaining logic specific to the parent business process.

Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Common objective of all service-orientation design principles

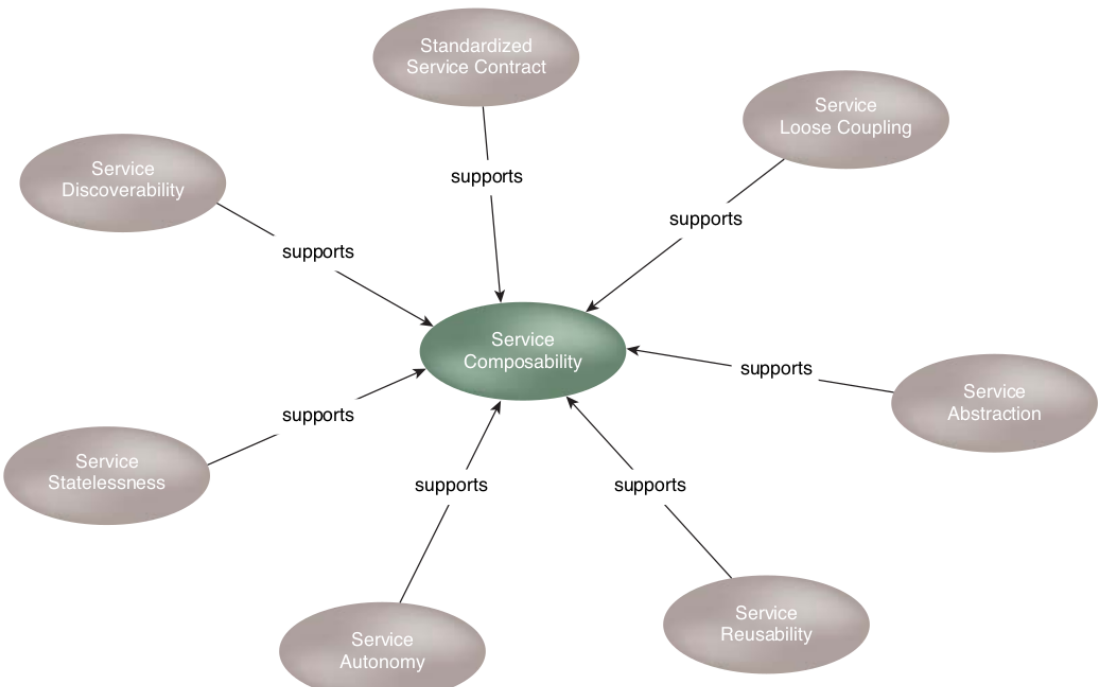
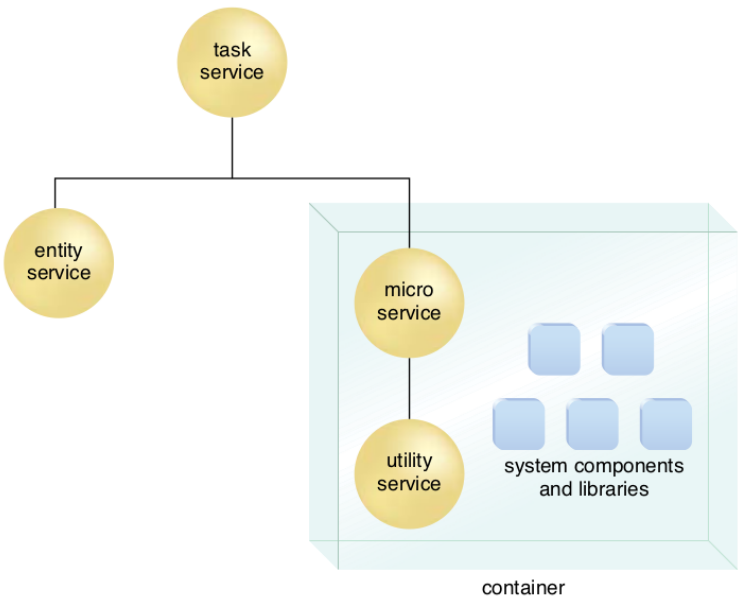


Figure 5.14

A common objective of all service-orientation design principles is the shaping of services in support of increased composability potential.

Using containers for microservices

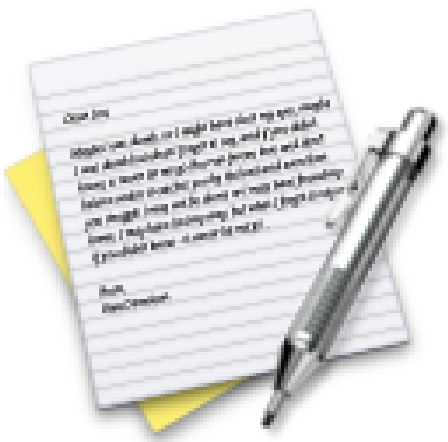
Figure 5.18
The microservice and the redundant implementation of the utility service are positioned within a container that also includes system components and libraries. This is an example of how containerization technology can be used to further increase the autonomy and mobility of services. The extent to which autonomy is increased depends on the extent to which redundant implementations of external resources the service may need to call are included in the container.



Source: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, Thomas Erl, 2017

Dont forget the exercises

Exercise

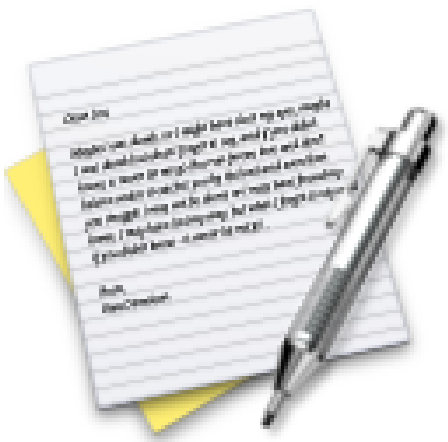


Now lets do the exercise

Run PostgreSQL

which is number **10** in the exercise PDF.

Exercise

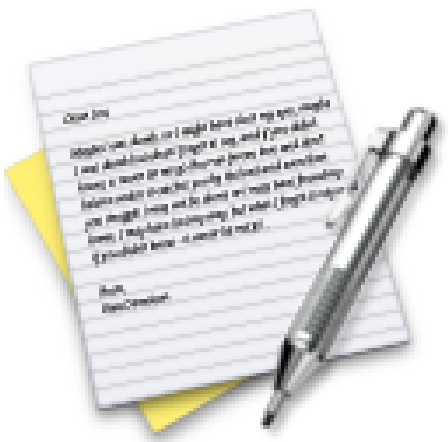


Now lets do the exercise

Why go to SOA 45 min

which is number **11** in the exercise PDF.

Exercise

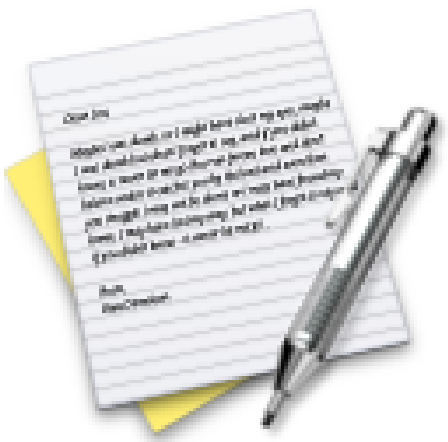


Now lets do the exercise

Cloud Computing Introduction 45 min

which is number **12** in the exercise PDF.

Exercise

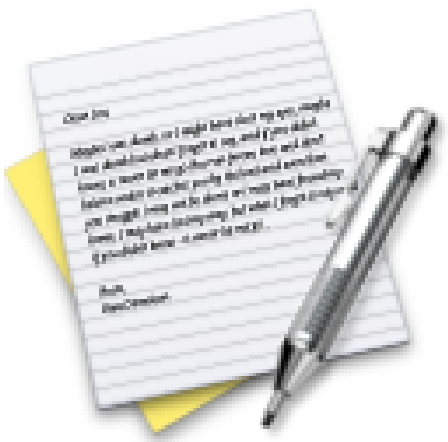


Now lets do the exercise

Cloud Deployment 45 min

which is number **13** in the exercise PDF.

Exercise

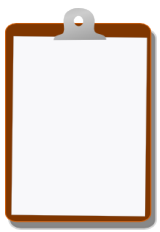


Now lets do the exercise

Download the Microservices ebook 20 min

which is number **14** in the exercise PDF.

For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools