Arnaud Weil



Learn Microservices

ASP.NET Core and Docker

Learn Microservices - ASP.NET Core and Docker

Be ready for coding away next week

Arnaud Weil

This book is for sale at http://leanpub.com/micro

This version was published on 2018-08-21



This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2017 - 2018 Arnaud Weil

To my parents, for teaching me freedom and making sure I can enjoy it.

To my wonderful family. Your love and support fueled this book.

To my readers who suggested improvements to this book, especially Doğan Kartaltepe and Louis-Guillaume Morand.

Contents

Introduction . . . . . **ii**
What this book is not ii
Prerequisites iii
How to read this book iii
Tools you need iii
Source code iv

1. Why microservices? **1**
1.1 What are they? 1
1.2 Microservices induce complexity 3

2. Microservices as seen from the client **5**
2.1 HTTP 5
2.2 Payloads 8
2.3 Do you speak JSON 9
2.4 Client types 11
2.5 Manual testing 12
2.6 Exercise - Invoke a microservice manually 14
2.7 Exercise solution 15

3. Microservices server-side **16**
3.1 About coding the server 16

3.2 There comes .NET Core 16
3.3 Introduction to ASP.NET Core 17
3.4 ASP.NET Core Web API 17
3.5 Exercise - Create a train scheduling service 18
3.6 Exercise solution 19

4. Database access using Entity Framework Core 22
4.1 Introduction to Entity Framework Core . . 22
4.2 Scaffold your code and save time 22
4.3 Exercise - Extend the train scheduling service 23
4.4 Exercise solution 23

5. Adding UI server-side 24
5.1 To UI or not to UI 24
5.2 UI scaffolding with Visual Studio 24
5.3 Exercise - Add a UI to the train scheduling service 25
5.4 Exercise solution 25

6. Invoking a service from .NET and .NET Core . . 26
6.1 .NET Clients 26
6.2 Demo: consume the microservice from a WPF client 26

7. Invoking a service from another service 27
7.1 Same but different but same 27
7.2 Exercise - Create a ticketing service . . . 27
7.3 Exercise solution 28
7.4 Exercise - Consume the scheduling service from the ticketing service 28
7.5 Exercise solution 28

7.6 Integration testing of the microservices created 29

8. Introduction to containers and Docker 30
8.1 Why do we need containers? 30
8.2 Virtualization could have been the panacea 31
8.3 Why containers perform better 32
8.4 Containers in practice 32
8.5 How containers solve problems 33
8.6 Containers as microservice hosts 34
8.7 Exercise - Run a Docker container 34
8.8 Exercise solution 34

9. Creating and running Docker images 35
9.1 How images are built 35
9.2 An ASP.NET Core Dockerfile 35
9.3 Multi-stage Dockerfiles 36
9.4 Exercise - Build and run your own Docker image 36
9.5 Exercise solution 36

10. Publishing images to Docker registries 37
10.1 Why bother with Registries? 37
10.2 Available Registries 37
10.3 Publishing an image 38
10.4 Exercise - Publish your services on Docker Hub 38
10.5 Exercise solution 38

11. Executing linked microservice containers in a Docker Swarm 39
11.1 The need for orchestrators 39

11.2 Orchestrators commonly used with Docker 39
11.3 Docker Swarm concepts 40
11.4 Creating a swarm 40
11.5 Configuring a stack 40
11.6 Deploying and managing a stack on a swarm 41
11.7 Exercise - Run the two services in a Docker Swarm 41
11.8 Exercise solution 41
11.9 Integration testing of the stack created . 42

12. Microservice clusters using Docker Swarm . . 43
12.1 Demo: load-balancing the stack 43

The Learn collection 44



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Introduction

What this book is not

I made my best to keep this book small, so that you can learn microservices quickly without getting lost in petty details. If you are looking for a reference book where you'll find answers to all the questions you may have within the next 4 years of your microservices practice, you'll find other heavy books for that.

My purpose is to swiftly provide you with the tools you need to code your first microservice application using Docker and ASP.NET Core, then be able to look for more by yourself when needed. While some authors seem to pride themselves in having the thickest book, in this series I'm glad I achieved the thinnest possible book for my purpose. Though I tried my best to keep all of what seems necessary, based on my 16 years experience of teaching .NET.

I assume that you know what microservices are and when to use them. In case you don't, read the following *Why microservices* chapter.

Prerequisites

In order for this book to meet its goals, you must:

- Have basic experience creating applications with C#
- Know what a Web application is

How to read this book

This book's aim is to make you productive as quickly as possible. For this we'll use some theory, several demonstrations, plus exercises. Exercises appear like the following:



Do it yourself: Time to grab your keyboard and code away to meet the given objectives.

In code samples, a backslash is used in order to wrap long lines. Do not type theses backslashes when you copy code from the book.

Tools you need

The only tools you'll need to work through this book are the following:

- Windows 10
- Visual Studio 2017 (Community, Professional or Enterprise)
- Docker 17.09 or higher (Community)

When installing Visual Studio, make sure you select the *ASP.NET* and *.NET Core* components.

Source code

All of the source code for the demos and do-it-yourself solutions is available at https://bitbucket.org/epobb/microservicesexercises

It can be downloaded as a ZIP file[1], or if you installed GIT you can simply type:

```
git clone https://bitbucket.org/epobb/microservicesexer\
cises.git
```

[1] https://bitbucket.org/epobb/microservicesexercises/downloads

1. Why microservices?

If you're in a hurry, you can safely skip this chapter and head straight to the Microservices as seen from the client chapter. This *Why microservices* chapter is there for those that want to know why microservices should be used.

1.1 What are they?

Microservices are an architectural solution to several challenges faced by modern software. Among other things, they help solve either of the following requirements:

- over time there may be several front-ends for similar business processes (thick clients, SPAs, mobile applications, IoT devices, …), some of which unpredicted;
- applications may be hosted on-premises or in a Cloud infrastructure;
- applications should be resilient: they should remain available when outages occur over parts of its components;
- applications should be able to scale up: when demand is high it should be possible to run them over

several machines in order to handle each request faster;
- applications should allow for replications, i.e., having duplicates across different machines or data centers so that computing power outage doesn't make them unavailable;
- large teams working on the same application should be able to work as independently as possible;
- future applications are likely to interact with a part of the application you are building.

In order to achieve such goals, microservices' architectures split an application into smaller blocks. If you've been architecting applications correctly you probably already separated your application code in projects, but that still makes for one big monolithic application.

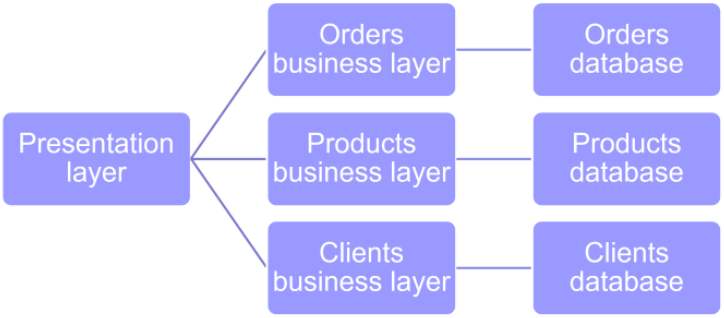
A monolithic application bundles its pieces together into a single unit. that cannot easily meet the requirements expressed above. Here's what a fictitious monolithic application could look like:



On the positive side, a monolithic application is easy to code and deploy.

In order to meet the requirements expressed earlier, a microservice architecture splits the application into several services. The following could be a microservice

architecture corresponding to the monolithic application shown above:



How you split the functionality can be debated. It is generally good practice to make sure each service meets a functional need of its own.

1.2 Microservices induce complexity

Services are likely to depend on each other; which doesn't appear in the schema above. For instance, considering the application above an order would most probably reference a client and several products. Which brings some questions like *how do services reference each other?* There are answers for that depending on what needs to be achieved. For instance there may be a service-discovery service, or a mediation layer.

This is just an example of the complexity that a microser-

vice architecture brings. We'll see more later but the scope of this book is not to cover it all, especially since there are potentially many answers to each problem.

What I want you to understand is that a microservice architecture is nothing simple. As the needs grow, the complexity will grow faster than in an equivalent monolithic architecture. My advice: use microservice architecture when you need to meet requirements like the ones listed at the beginning of that chapter. Otherwise keep with monolithic architectures, be it a service-based monolithic architecture.

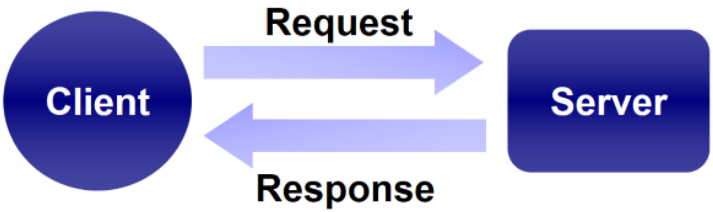
Anyway, my point is about complexity not impossibility. There's a lot of awesome tooling out there today that makes a microservice architecture much easier to code, assemble and deploy. We'll see some of them in this book.

2. Microservices as seen from the client

A microservice architecture is made of services. Those services handle HTTP communication that request or send data. In case you're familiar with this, you can head straight to the exercise. Otherwise, let's review the fundamentals so you can build your knowledge over solid basis.

2.1 HTTP

Services, whether they are microservices or not, are usually called over HTTP. Here's the schema of an HTTP request:



A *client* sends a request to a *server*. The client may be a mobile application, an IoT object, a SPA application

(Angular, React, ...) or whatever you please - including another service. By the way, services will be calling each other in a microservice architecture. The server is what you code, and it contains the business logic.

HTTP in itself is pretty simple. It's a text-based protocol where the client:

1. opens a TCP connection to the server;
2. sends data (the *request*, see below) to the server;
3. receives data (the *response*, see below) from the server;
4. closes the connection.

The request is made of two parts: headers and possibly data, separated by a blank line (did I tell you HTTP was simple?). It may look like that:

```
GET /index HTTP/1.1
Host: www.mysite.com
```

In the above example there is one single header (Host) and no data. The first line states a *verb*, a URL, and the version of the HTTP protocol being used. The verb corresponds to what the client wants to do, for instance:

- GET is used to fetch data without modifying server data
- POST is used to update server data
- DELETE is used to - ahem! - delete server data, or at least request deletion.

GET requests do not contain a data payload in the request since the headers and URL are sufficient, but the other types of requests contain data. Here's an example POST request, but an UPDATE request would look similar:

```
POST http://server:51426/api/TrainSchedules HTTP/1.1
Content-Type: application/json
Host: server:51426
Content-Length: 94

{"departureTime":"2018-04-03T21:56:48.6711219+01:00","d\
istanceKm":1054,"destination":"Berlin"}
```

The payload here is JSON, a widely used format for services as we'll see very soon.

The server processes the request and returns a response using the same TCP connection that was used for the request. That TCP connection is then closed. Here's an example response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 81

<html>
    <body>
        <p>Welcome to our site</p>
    </body>
</html>
```

Just like the request, the response is made up of two parts: headers and possibly data, separated by a blank line. Plus a first line that confirms the protocol and provides a status code and message. The status code's first digit is the most important: 2 means everything was alright, 4 means there was a problem processing the request (you know, that ubiquitous *404 Not found*).

There can be many more headers. In the above example you can see the two headers that are commonly used in responses. *Content-Type* tells the client how the data (what comes below the blank line) should be interpreted. It's a MIME type. MIME types are well established[1]. Though there are many possible MIME types, only some of them are used for services.

2.2 Payloads

We are focusing on services, right? A (micro)service is mostly used for computer-computer interaction, so it needs a MIME type that's appropriate for its payloads (the data that is sent in requests and received in responses).

Extensive work from the OASIS consortium in order to standardize services at the beginning of the century (wow, that sounds *so vintage*) established the XML format for payloads; the consortium even wrapped it into the SOAP standard (which is made of XML). So services

[1] https://www.iana.org/assignments/media-types/media-types.xhtml

used the *text/xml* MIME type for quite a while, and many still do.

Then came SPA applications (Angular, React and the likes), which became widely used endpoints for user consumption of services. Those applications run in the browser using JavaScript that happens to be slow at XML processing. Sure, JavaScript can parse and produce XML, but it's a painful process for it. What JavaScript can easily parse and produce is its own format: JSON. Since JSON is as easily read by humans as XML and less verbose, it quickly became the norm for services.

So that's where we stand today: (micro)services use mainly JSON as their payload format for requests and answers, plus XML sometimes.

2.3 Do you speak JSON

If you aren't familiar with JSON, don't worry: it can be learnt very quickly. Think of it as anonymous C# objects. For instance, consider the following declaration:

```
var myObject = new() {
  Foo = "bar",
  Value = 5.5,
  Really = true
}
```

This anonymous object instantiation in C# is almost

JSON. Let's replace the equal signs with semicolons, and wrap property names with quotes. This is what we get:

```
var myObject = new() {
  "Foo": "bar",
  "Value": 5.5,
  "Really": true
}
```

We're almost there. We currently have an hybrid C#-JSON code. Now let's remove the variable declaration and call to *new* since they don't describe a value (they describe what we want to do with the value). This is what we get:

```
{
  "Foo": "bar",
  "Value": 5.5,
  "Really": true
}
```

Guess what? This is valid JSON. It simply represents the data for an object that has 3 properties.

Now suppose you want to describe an array of object values: all you need to do is use brackets around them. And line returns count as spaces. Yes, all just like C#. Here's one such array:

```
[
  { "Foo": "bar", "Value": 5.5, "Really": true },
  { "Foo": "tball", "Value": 2.3, "Really": false }
]
```

That's almost all there is to JSON. I could show an example of a complex property value, but that would be just recursive use of the first sample and I'll spare you that.

2.4 Client types

Once a (micro)service is published and listens over HTTP (that's what we'll do in this book), many different types of clients can consume it:

- heavy clients like UWP, WPF, Windows Forms;
- browsers using SPA applications like Angular, React, Vue.js, Knockout or jQuery, or even plain JavaScript
- mobile applications;
- IoT devices, e.g. your fridge ordering milk when it senses you'll soon be running out of milk;
- other services, which makes for a services-based architecture;
- utilities like command-line applications.

In this book you'll use WPF, command-line and services as the clients of your services.

2.5 Manual testing

Once you start building a microservice, you'll want to test it. A microservice usually has no user interface, hence testing requires you to send HTTP requests and check for the response.

> That way of interacting with a service (HTTP) is often called an API for Application Programming Interface.

An obvious tool is your browser. When you type a URL in the address bar, your browser issues an *HTTP GET* request to that URL. Downside: you have no control over the headers it sends. Which means that you really need another tool.

Good news: there are many tools out there that will enable you to issue HTTP requests like *Fiddler* or *Postman*. You can go on and use any tool you like really; for simplicity sake in this book we'll use the *Invoke-WebRequest* command that is part of PowerShell since it's already present on your Windows machine. If you're using a Linux-based machine, you can use the *curl* command. Even PowerShell accepts the *curl* command (and maps it to its *Invoke-WebRequest* command).

Suppose you want to make the following HTTP request: *GET http://someserver/api*

You could issue the following PowerShell command:

```
Invoke-WebRequest http://someserver/api
```

This is the actual HTTP request that gets issued:

```
GET http://someserver/api HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; f\
r-FR) WindowsPowerShell/5.1.16299.98
Host: someserver
Proxy-Connection: Keep-Alive
```

You aren't required to provide other parameters since *GET* is the default HTTP verb used by the *Invoke-WebRequest* command. The response is provided in full details, but if you are interested only in the data payload, you may get the *Content* property from the result:

```
(Invoke-WebRequest http://someserver/api).Content
```

When you need to use a verb other than *GET* you'll surely need to send data. The *Invoke-WebRequest* command allows you to provide your data using the *Body* parameter and the verb to be used is passed through the *Method* parameter.

```
Invoke-WebRequest -Method POST -Body '{"foo":"bar","val\
ue":10}' -Header @{"Content-Type"="application/json"} h\
ttp://someserver/api
```

Note that in the command above we're also sending a *Content-Type* header in order to tell the server that we're using JSON as the payload data format - this depends on the specific API you call.

This is the actual HTTP request that gets issued when the command above is executed:

```
POST http://someserver/api HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; f\
r-FR) WindowsPowerShell/5.1.16299.98
Host: someserver
Content-Length: 24

{"foo":"bar","value":10}
```

Let's try this on a real-world example.

2.6 Exercise - Invoke a microservice manually



Using PowerShell, make an HTTP GET request to the service located at the following URL:

```
https://en.wikipedia.org/w/api.php?action=query&format=\
json&prop=info&list=allpages
```

2.7 Exercise solution

- From a PowerShell command-line, type the following and check the returned *Content*:

```
Invoke-WebRequest "https://en.wikipedia.org/w/api.php?a\
ction=query&format=json&prop=info&list=allpages"
```

- Make sure the HTTP response status code is *200 OK*.

3. Microservices server-side

3.1 About coding the server



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

3.2 There comes .NET Core



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

3.3 Introduction to ASP.NET Core



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

3.4 ASP.NET Core Web API

Project structure



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Controllers, actions and routes



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Complex return types



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

3.5 Exercise - Create a train scheduling service



Create a new micro service named *ScheduleService* using the *ASP.NET Core Web Application* template. Your project should use *ASP.NET Core 2.0* and the *Web Application (Model-View-Controler)* template.

Your microservice should listen on the following URL: http://localhost:[port]/api/NextDeparture

When that URL is invoked, your microservice should return JSON that appears like:

```
{"departureTime":"2018-01-03T21:56:48.6711219+01:00","n\
umber":"5423","destination":"Paris"}
```

3.6 Exercise solution

- Start Visual Studio.
- Click on the *File / New / Project…* menu entry.
- In the *New Project* dialog box, select the *ASP.NET Core Web Application* template making sure that you select *Templates / Visual C# / .NET Core* on the left-hand side. In the *Name* zone at the bottom, type "ScheduleService". Click the *OK* button.
- In the *New ASP.NET Core Web Application* dialog box, select the *.NET Core* and *ASP.NET Core 2.0* at the top. Select the *Web Application (Model-View-Controler)* template in the middle. Make sure the *Enable Docker Support* checkbox is unchecked. Click the *OK* button.
- Open the *Solution Explorer* clicking on the *View / Solution Explorer* menu entry.
- In the *Solution Explorer*, right-click the *Controllers* directory, and select *Add / Controller…* from the context menu.
- In the *Add Scaffold* dialog box, select the *API Controller - Empty* template. Click the *Add* button.
- In the *Add Empty API Controller* dialog box, type *NextDepartureController* as the controller name. Click the *Add* button.
- Locate the following code:

```
[Produces("application/json")]
[Route("api/NextDeparture")]
public class NextDepartureController : Controller
{
}
```

- Replace it with the following code:

```
[Produces("application/json")]
[Route("api/NextDeparture")]
public class NextDepartureController : Controller
{
  public IActionResult GetNext()
  {
    return Ok(new {
      DepartureTime = DateTime.Now.AddMinutes(50),
      Number = "5423",
      Destination = "Paris"
    });
  }
}
```

- From the *Debug* menu click *Start Without Debugging*
- Point your browser to the following URL (port number varies):

http://localhost:51426/api/NextDeparture

- From a PowerShell command-line, type the following and check the returned *Content*:

```
Invoke-WebRequest http://localhost:51426/api/NextDepart\
ure
```

4. Database access using Entity Framework Core

4.1 Introduction to Entity Framework Core



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

4.2 Scaffold your code and save time



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

4.3 Exercise - Extend the train scheduling service



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

4.4 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

5. Adding UI server-side

5.1 To UI or not to UI



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

5.2 UI scaffolding with Visual Studio



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

5.3 Exercise - Add a UI to the train scheduling service



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

5.4 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

6. Invoking a service from .NET and .NET Core

6.1 .NET Clients



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

6.2 Demo: consume the microservice from a WPF client



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

7. Invoking a service from another service

7.1 Same but different but same



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

7.2 Exercise - Create a ticketing service



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

7.3 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

7.4 Exercise - Consume the scheduling service from the ticketing service



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

7.5 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

7.6 Integration testing of the microservices created



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8. Introduction to containers and Docker

8.1 Why do we need containers?

The deployment hassle



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Continuous integration



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Upgrading common dependencies



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

DevOps



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.2 Virtualization could have been the panacea



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.3 Why containers perform better



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.4 Containers in practice

Containers



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Some vocabulary: containers and images



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Running a simple image



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

Running a server image



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.5 How containers solve problems



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.6 Containers as microservice hosts



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.7 Exercise - Run a Docker container



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

8.8 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

9. Creating and running Docker images

9.1 How images are built



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

9.2 An ASP.NET Core Dockerfile



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

9.3 Multi-stage Dockerfiles



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

9.4 Exercise - Build and run your own Docker image



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

9.5 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

10. Publishing images to Docker registries

10.1 Why bother with Registries?



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

10.2 Available Registries



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

10.3 Publishing an image



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

10.4 Exercise - Publish your services on Docker Hub



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

10.5 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11. Executing linked microservice containers in a Docker Swarm

11.1 The need for orchestrators



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.2 Orchestrators commonly used with Docker



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.3 Docker Swarm concepts



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.4 Creating a swarm



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.5 Configuring a stack



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.6 Deploying and managing a stack on a swarm



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.7 Exercise - Run the two services in a Docker Swarm



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.8 Exercise solution



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

11.9 Integration testing of the stack created



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

12. Microservice clusters using Docker Swarm

12.1 Demo: load-balancing the stack



This is just a sample of the full book.

If you like it, get your full version here: https://leanpub.com/micro

The Learn collection

This book is part of the *Learn collection*.

The *Learn collection* allows developers to self-teach new technologies in a matter of days.

Published books

- Learn ASP.NET Core MVC[1]
- Learn ASP.NET MVC[2]
- Learn Meteor[3]
- Learn Microservices[4]
- Learn WPF MVVM[5]

To be published

- Learn Docker
- Learn Kubernetes
- Learn Unit Testing

[1] https://leanpub.com/netcore
[2] https://leanpub.com/aspnetmvc
[3] https://leanpub.com/learnmeteor
[4] https://leanpub.com/micro
[5] https://leanpub.com/learnwpf