<https://www.youtube.com/watch?v=DFDbh1c9zyE>

<https://www.youtube.com/watch?v=DFDbh1c9zyE>

D:\git\.NET\Microservices\mslearn-dotnetmicroservices\mslearn-dotnetmicroservices

40,974 views Premiered Oct 18, 2021 In recent years, enterprises are choosing microservices over monolithic for their large consumer applications to meet user demand, increase scalability and availability. In this episode, Nish Anil will take you through Microservices concepts, what are containers and how they're relevant. You will learn to build your first Microservices endpoint in .NET, containerize it and run them locally within 30 mins. 👉 Learn how to build your first microservice: [https://docs.microsoft.com/learn/modu...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbF92b1Bld3luYXdnbVpKRG80YmpZQ2JaOU9IQXxBQ3Jtc0tsZmVEYko4MWJBNXl0MXk5cFNCbjE1US1wNDVCeG1KUkNDbXFFRlJVMDBpclktVHpsNmV1WmxQUmQxcV82M2xMLTdfck1uR1VtVk50UjVsbnFvWU93RVZXcml0OEd6Y0k2ZjhjUzMtc3ZJX3c4UnZ1MA&q=https%3A%2F%2Fdocs.microsoft.com%2Flearn%2Fmodules%2Fdotnet-microservices%2F%3FWT.mc_id%3Dfriends-0000-NANIL&v=DFDbh1c9zyE) [00:00](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=0s) - Introduction [01:40](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=100s) - What are Microservices? [03:04](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=184s) - Comparing to Monolithic Architecture [04:02](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=242s) - Decomposing an app into multiple services [05:46](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=346s) - How are microservices packaged and deployed? [06:08](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=368s) - What role do containers play? [06:43](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=403s) - What is Docker? [07:22](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=442s) - Docker Architecture [08:01](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=481s) - Containers vs Virtual Machines [08:49](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=529s) - What is an Orchestrator/Kubernetes? [09:33](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=573s) - Demo: Build a Microservice endpoint in .NET [13:56](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=836s) - Dockerfile explanation [16:14](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=974s) - Common Docker error - the docker daemon is not running [17:51](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=1071s) - What are Docker multi-stage builds? [19:58](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=1198s) - Run multiple containers locally using Docker-Compose 👉 ASP.NET Core Microservices learn path [https://aka.ms/aspnet-microservices](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbm12MHRZbEZJaGRmTUMtZUlRMVFUTmJhOEgyd3xBQ3Jtc0tueTFSTS1VSUxYWDNuRGpXdXU0Nkl3d1hRNHhjRnBKbWlaSjBuTmFNeGlQMUFRMWhWZHhpUF9NSGdETUdWbGZmM1I4N05iZTF0Sm1lWlRPeUlmX3FkcUlLOEJDNWhqX2FhUlBVVC1ZQm9iaTlKamJSYw&q=https%3A%2F%2Faka.ms%2Faspnet-microservices&v=DFDbh1c9zyE) 🥳📚Free E-Books - go Zero to expert 📚 [https://dot.net/architecture](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbW5uLWVDUGdSN1VIcTRGSF9yNEVzQUZnbGhUd3xBQ3Jtc0tsWmliU0hDaUtOaks4UmRGdVN6T3l3TU5BaU53a0JiMVhLSGpuSy11dWgxQXdURDBiZFJuekJFVnF5VHBselUzclowemFXYVQwVnR2eE5oLU05cXltNHNxWjRWMG5RRFJNV3NTLWE1al9zWjBWcmhUZw&q=https%3A%2F%2Fdot.net%2Farchitecture&v=DFDbh1c9zyE) ♥️ Are you new to .NET? Start here: [https://dot.net](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbUlCUmNjOEx6MXpkNmNIWFg1VjF0MGhVNmluQXxBQ3Jtc0tsUXNnQkxOdmQxVUNDRUgtSUluU3QwelJnbjl0ZlJrZ2VleVBEbEF3SVJuTThPSUJCTEJxb0E5NG1lZWFmcXNxUUN4QXFJejh4TzFDTFNEdERRWGM0UmhzVlUzY0NpSFlXQ1prSHRFMWNBTTBDS1VOOA&q=https%3A%2F%2Fdot.net%2F&v=DFDbh1c9zyE) 🙋‍♀️🙋‍♂️ Get your questions answered on the Microsoft Q&A for .NET: [https://aka.ms/dotnet-qa](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbnhQbDd1VHVrY3BOQ1FXSmdmM2F0OXlWY1llZ3xBQ3Jtc0tsT3JOWldlR2FuQWNkYTdhZk4taER6T0o3OG1FSVAzUnRwb18tR0haR1J2YkhweG1sUlRuQnNqZzhFdk5LX3pWbGlwZ255Vk5fZHRob3M0c3FZREs3bzdLUUdNZmJLelMtSEVtakZ0UGxvVWwxOTVYOA&q=https%3A%2F%2Faka.ms%2Fdotnet-qa&v=DFDbh1c9zyE) 🏫 Learn C#, F#, and .NET with free self-guided learning from Microsoft Learn: [https://aka.ms/learndotnet](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEowa1VDNFNfYzJ4cWhVcFh1QWk3eldPZjJiUXxBQ3Jtc0ttb2wzMjJ3ZF9VcVB0MTdYMWJqVFRaSlBKanhFa3oxbzNuZmMxNXh0REVLZHZkM1JxRDdDa0ItLURYU0I5d084NU5PY1dfVEtEME40RG5KVVhraVkzdlNhc3BScnRiZGU3eVo5MGNjNjlIeWlPdFkxTQ&q=https%3A%2F%2Faka.ms%2Flearndotnet&v=DFDbh1c9zyE) [#microservices](https://www.youtube.com/hashtag/microservices) [#DotNet](https://www.youtube.com/hashtag/dotnet) [#Docker](https://www.youtube.com/hashtag/docker) [#aspnet](https://www.youtube.com/hashtag/aspnet) [#cloudnative](https://www.youtube.com/hashtag/cloudnative) [#architecture](https://www.youtube.com/hashtag/architecture) [#Dockerfile](https://www.youtube.com/hashtag/dockerfile)

What are Microservices?

Micro services –also Microsserviecs styled architecture

* Architectural style that structure an application as collection of smaller services
* Highly Maintainable and Testable
* When the services are small the code base is much easier to maintain and test loosely.
* Service connect to each other on a servicing point
* Microservices are organized around business capabilities
* Decoupled them based on the business problem
* Pattern as Domain-Driven Design
* Owned by small team
* Managed by independent team give the autonomy to decide when to feature or fix a bug with a weighting on the entire team released.
* Having services model based on domain business, domain, autonomous
* Packaged and deployed independently.

Comparing to Monolithic Architecture

Decomposing an app into multiple services

Monolith – are tightly coupled.

* Feature roll-outs must have a specific cadence that all
* Team must work together to prioritize every release, increase friction
* Common database for all your modules , often the bottleneck for performance and implement change.
* Scaling VM’S and Infrastructures.

Microservices

* App is Decomposed into multiple independent smaller services that focuses in specific business functionality
* Every microservices is responsible for processing that data or external state in their respective databases.
* Individual services can now decide. If they want to use SQL database or any database of choice based on business need.
* Microservices do not shared databases.
* Team can now independently deploy applications multiple times a day
* Without breaking each other
* Communicate with each other by using well-defined API’s
* Its also support **Polygot** (**f writing code in multiple languages to capture additional functionality and efficiency not available in a single language)** programming
* Dot-net application + Java applications

How are microservices packaged and deployed?

* Docker containers and Kubernetes
* Significant benefit with Cloud Native Technologies
* Help team shift features to production fosters.

What role do containers play?

* Application/services it s dependencies and its configurations of packaged together as a container image.
* Gained popularity since Docker’s open-source implementation.
* **Docker** – Open source project for automating applications as portable self-sufficient containers that can run on the cloud or on-premise.
* Using docker developers creates an app or service and packages it and its dependencies into a container image and image is a static
* An image is static representation of the app or service and its configurations and dependencies.
* It’s the image that when run become the container
* Container its memory its tense of an image in a typical scenario using docker

[07:22](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=442s) - Docker Architecture

* You package your app/services as an image and store in somewhere
* Just need the container registry
* When you want to run your app you pull the image from container registry
* Deploy to the hosting operating system and run them as container instance.
* For scalability you can scale out quickly be creating new containers
* Crucial benefits for cloud native.

### Container Image is Immutable ( once image is created, image cannot be changed)

* Only way is to create new image and replace it.

[08:01](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=481s) - Containers vs Virtual Machines

* Traditional hardware,OS, hypervisor,
* Most often low utilization of resources
* Your paying underutilized resources
* Containers are lightweight
* You can spin up a container within seconds

[08:49](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=529s) - What is an Orchestrator/Kubernetes?

* Your application is decomposed into smaller services running in containers
* In microservices based approach owns its model and data to be autonomous from the development and deployment view.
* Therefore you need orchestrator to have a production ready and scalable multiple container application
* Kubernetes has become the defacto standard for orchestrator
* Can run in local or in the cloud

[09:33](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=573s) - **Demo**: Build a Microservice endpoint in .NET

ASP.NET Core Microservices learn path [https://aka.ms/aspnet-microservices](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa0ZOZ01sWFRlQW9RMVZaZjJMcTF2cFJNWFJYUXxBQ3Jtc0tuQkhuTlgtZjhpZFhBbGhTb0FaRlBIRlBqYjVkeXlldjNpUnR4ZUVvaE02LVpES29zWlNkMG5GaFdFQ2dxQVpiMkpjM05LVVhpZjZmc0RnSWxsSTE1cWtTdEtObTIyUEdmUW1PQ3lmMnlNZUxnbjQyQQ&q=https%3A%2F%2Faka.ms%2Faspnet-microservices&v=DFDbh1c9zyE)

<https://docs.microsoft.com/en-us/learn/paths/create-microservices-with-dotnet/?WT.mc_id=friends-0000-NANIL>

# What are microservices?

200 XP

* 7 minutes

The cloud drives today's application development and IT system management. Modern cloud applications need to be fast, agile, massively scalable, and reliable.

Containers help applications achieve all of those requirements. But putting an application into a container without following a design pattern is like getting into a vehicle and hoping to find way to a new city without a map (or GPS-enabled phone). You may end up at your destination, but the route probably wasn't the most efficient.

This is where a microservice architecture comes in. Microservices enable an approach to software development and deployment that is perfectly suited to the agility, scale, and reliability requirements of modern cloud applications.

## What is a microservice architecture?

As the name implies, a microservices architecture is an approach in which a large application is split up into a set of smaller services. Each service runs in its own process and communicates with other processes using protocols such as HTTP/HTTPS, WebSockets, or AMQP. Each microservice implements a specific end-to-end domain or business capability within a certain context boundary, and each must be developed autonomously and be deployable independently. Finally, each microservice should own its related domain data model and domain logic, and could be based on different data storage technologies (SQL, NoSQL) and different programming languages.

Some key characteristics of microservices are:

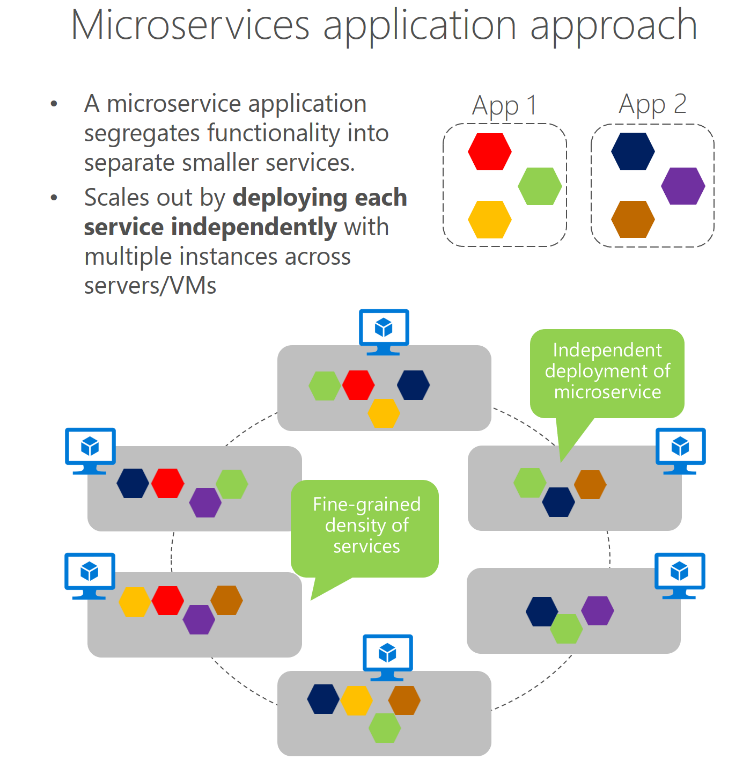
* Microservices are small, independent, and loosely coupled.
* Each microservice has a separate codebase, which can be managed by a small development team.
* Microservices are deployed independently. A team can update an existing microservice without rebuilding and redeploying the entire application.
* Microservices are responsible for persisting their data or external state in their respective databases. Unlike the monolithic architecture, microservices do not share databases.
* Microservices communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
* Supports polyglot programming. For example, microservices don't need to share the same technology stack, libraries, or frameworks.

## Why develop in a microservice architecture?

The benefits of microservices are that each one typically encapsulates simpler customer-requirement functionality, which you can scale out or in, test, deploy, and manage independently. One important benefit of a microservices approach is that teams are driven more by customer scenarios than by technology. Smaller teams develop a microservice based on a customer scenario and use any technologies they want to use.

Microservices provide long-term agility. Microservices enable better maintainability in complex, large, and highly-scalable systems by letting you create applications based on many independently deployable services that each have granular and autonomous lifecycles.

As an additional benefit, microservices can scale out independently. Instead of having a single monolithic application that you must scale out as a unit, you can instead scale out specific microservices. That way, you can scale just the functional area that needs more processing power or network bandwidth to support demand, rather than scaling out other areas of the application that don't need to be scaled. That means cost savings because you need less hardware.

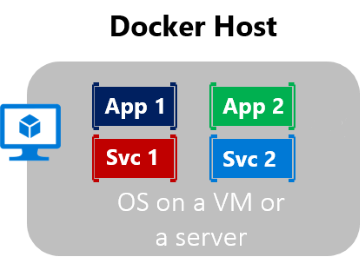


**What role do containers play?**

Containerization is an approach to software development in which an application or service, its dependencies, and its configuration (abstracted as deployment manifest files) are packaged together as a container image. You can test the containerized application as a unit, and deploy them as a container image instance to the host operating system (OS).

Just as shipping containers allow goods to be transported by ship, train, or truck regardless of the cargo inside, software containers act as a standard unit of software deployment that can contain different code and dependencies. Containerizing software this way lets developers and IT professionals deploy them across environments with little or no modification.

If that sounds like containerizing an application would be a great way to implement the microservice architecture pattern, it is. The benefits of containers almost line up exactly to the benefits of microservices one-to-one.



### Docker

[Docker](https://www.docker.com/) is an [open-source project](https://github.com/docker/docker) for automating the deployment of applications as portable, self-sufficient containers that can run in the cloud or on-premises. [Docker](https://www.docker.com/) is also a company that promotes and evolves this technology, working in collaboration with cloud, Linux, and Windows vendors, including Microsoft.

Docker containers can run anywhere on Azure: on-premises in the customer's datacenter, in an external service provider, or in the cloud. Docker image containers can run natively on Linux and Windows.

### What is an image?

When using Docker, a developer creates an app or service and packages it and its dependencies into a container image. An image is a static representation of the app or service and its configuration and dependencies.

It's this image that, when run, becomes our container. The container is the in-memory instance of an image.

A container image is immutable. Once you've built an image, the image can't be changed. Since you can't change an image, if you need to make changes, you will create a new image. This feature is our guarantee that the image we use in production is the same image used in development and QA.

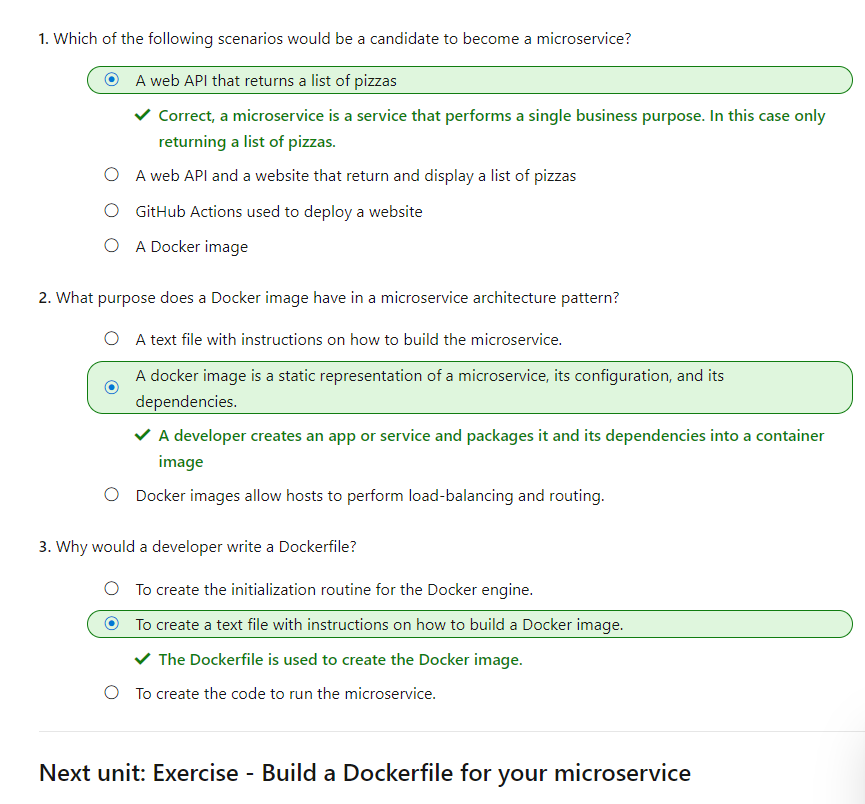
### What is a Dockerfile?

A Dockerfile is a text file that contains instructions on how to build a Docker image. Dockerfiles are written in a minimal scripting language designed for building and configuring images. They also document the operations required to build an image starting with a base image.

To create a Docker image containing your application, you'll typically begin by identifying a base image to which you add additional files and configuration. The process of identifying a suitable base image usually starts with a search on Docker Hub for a ready-made image that already contains an application framework and all the utilities and tools of a Linux distribution like Ubuntu or Alpine. For example, if you have an ASP.NET Core application that you want to package into a container, Microsoft publishes an image called mcr.microsoft.com/dotnet/core/aspnet that already contains the ASP.NET Core runtime.

You can customize an image by starting a container with the base image and making changes to it. Changes usually involve activities like copying files into the container from the local filesystem, and running various tools and utilities to compile code.

In other words, a Docker file is a set of instructions that builds up a Docker image with the exact software you need in it to run your application - including your application itself.



<https://docs.microsoft.com/en-us/learn/modules/dotnet-microservices/3-exercise-build-docker-image>

git clone <https://github.com/MicrosoftDocs/mslearn-dotnetmicroservices>

git bash

Lito@JMPASCUADESKTOP MINGW64 ~/git/NET

C:\Users\Ryzen\git\NET

D:\git\.NET\Microservices\mslearn-dotnetmicroservices\mslearn-dotnetmicroservices

Dotnet build

Try : dotnet run

Build : docker build -t pizzabackend .

Run : docker run -it --rm -p 5200:80 --name pizzabackendcontainer pizzabackend

docker run -it --rm -p 5201:80 pizzabackend

<http://localhost:5201/PizzaInfo>

D:\git\.NET\Microservices\mslearn-dotnetmicroservices\mslearn-dotnetmicroservices

Go to the backend

Dotnet build

Dotnet run

<https://localhost:5901/swagger/index.html>

Lito@JMPASCUADESKTOP MINGW64 ~/git/NET

C:\Users\Ryzen\git\NET

**What is docker file ?**

* A docker file is a text file that has instruction for Docker to build your image
* Its recommended that your create an image of an existing popular image

This will perform the following steps sequentially when invoked:

* Pull the mcr.microsoft.com/dotnet/aspnet:6.0 image
* Set the working directory within the image to /app
* Exposes port 80 and 443
* Copy everything from the /app directory of the **build** image created above into the app directory of this image
* Sets the entrypoint of this image to dotnet and passes backend.dll as an argument

Cd backend

docker build -t pizzabackend .

D:\git\.NET\Microservices\mslearn-dotnetmicroservices\mslearn-dotnetmicroservices\backend

To test

Docker run hello-world

Docker pull hello-world

See the list of docker images

Docker Images

<https://docs.microsoft.com/en-us/learn/modules/dotnet-microservices/4-orchestrators>

<https://docs.microsoft.com/en-us/learn/modules/dotnet-microservices/3-exercise-build-docker-image>

git clone <https://github.com/MicrosoftDocs/mslearn-dotnetmicroservices>

git bash

Lito@JMPASCUADESKTOP MINGW64 ~/git/NET

C:\Users\Ryzen\git\NET

D:\git\.NET\Microservices\mslearn-dotnetmicroservices\mslearn-dotnetmicroservices

Dotnet build

Try : dotnet run

Build : docker build -t pizzabackend .

Run : docker run -it --rm -p 5200:80 --name pizzabackendcontainer pizzabackend

>docker run -t --rm -p 5201:80 pizzabackend

.net

sudo ln -s "$HOME/dotnet/dotnet" "/usr/bin/dotnet"

Installing docker in ubuntu

<https://docs.docker.com/desktop/install/linux-install/>

Start SSH

sudo service ssh status

sudo service ssh start

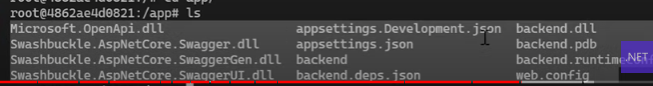
--docker

<https://admcpr.com/how-to-upgrade-wsl-1-to-wsl-2/>

wsl -l -v

wsl --set-version Ubuntu 2

10am – 6am



[17:51](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=1071s) - What are Docker multi-stage builds?

* Size of the image 600MB included .net SDK and build inside it
* SDK + source code
* Final can be compact runtime + package dlls only
* Docker provides of your final image drastically in multi-stage build
* When testing application requires other test tools and final image can be excluded.

# Pull the mcr.microsoft.com/dotnet/aspnet:6.0 image

# Set the working directory within the image to /app

# Exposes port 80 and 443

# Copy everything from the /app directory of the build image created above into the app directory of this image

# Sets the entrypoint of this image to dotnet and passes backend.dll as an argument

FROM mcr.microsoft.com/dotnet/aspnet:6.0

WORKDIR /app

EXPOSE 80

EXPOSE 443

COPY --from=build /app .

ENTRYPOINT ["dotnet", "backend.dll"]

[19:58](https://www.youtube.com/watch?v=DFDbh1c9zyE&t=1198s) - Run multiple containers locally using Docker-Compose

* Using Docker –compose to run multiple containers
* You need a yaml file to configure the services

<https://docs.microsoft.com/en-us/learn/modules/dotnet-microservices/4-orchestrators>

# Microservices orchestration

200 XP

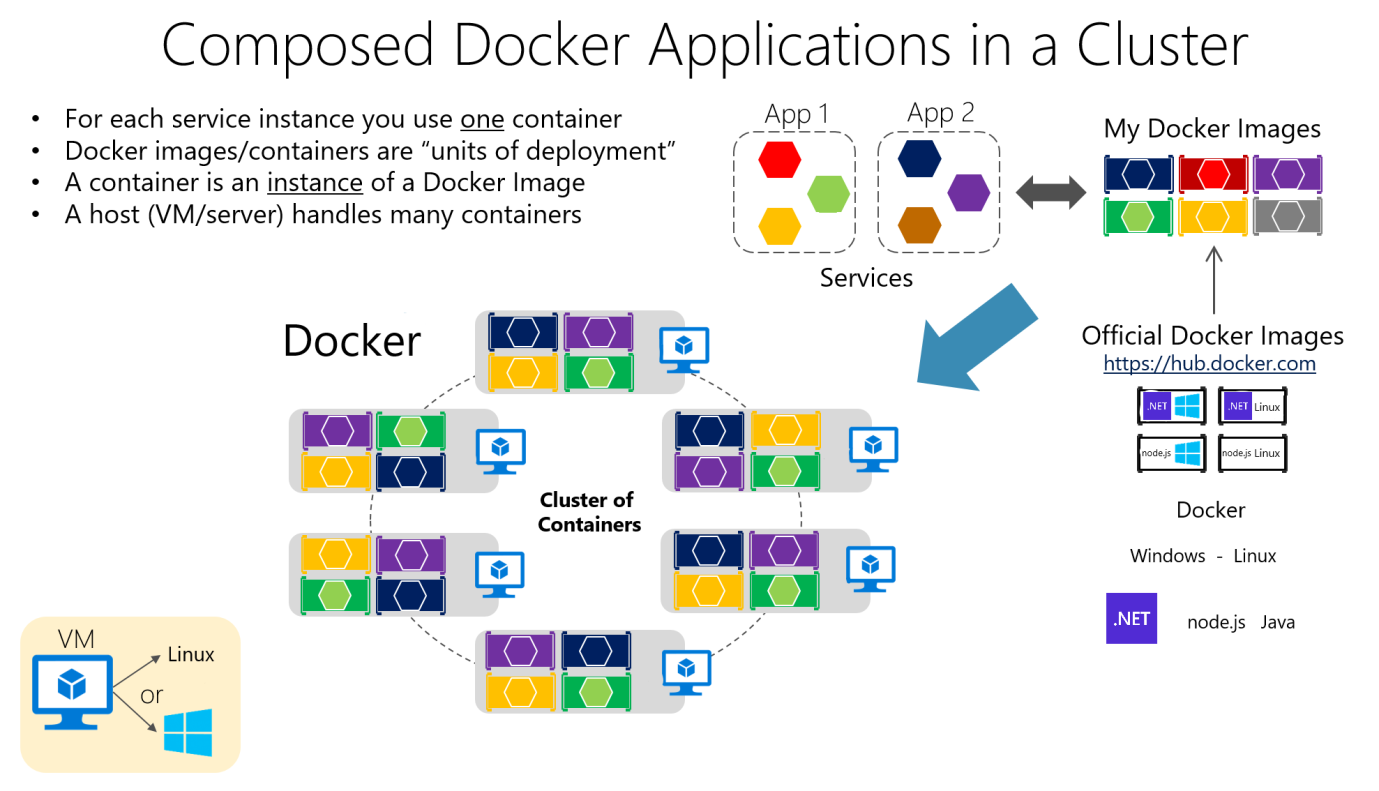
Contoso Pizza loves the results the microservice architecture has been giving them so far. The overall web application calls individual microservices to provide and manipulate data.

As more and more services get added, however, the overall system becomes more complex to scale out and manage. Orchestrators can help.

## What is an orchestrator?

Using orchestrators for production-ready applications is essential if your application is based on microservices or simply split across multiple containers. As introduced previously, in a microservice-based approach, each microservice owns its model and data so it will be autonomous from a development and deployment point of view. These kinds of systems are complex to scale out and manage; therefore, you absolutely need an orchestrator if you want to have a production-ready and scalable multi-container application.

The image below illustrates deployment into a cluster of an application composed of multiple microservices (clusters).



* Docker images/Containers – are unit of deployment
* Container is an instance of a Docker Image

<https://docs.microsoft.com/en-us/learn/modules/dotnet-microservices/5-exercise-create-docker-compose-file>

You use one container for each service instance.

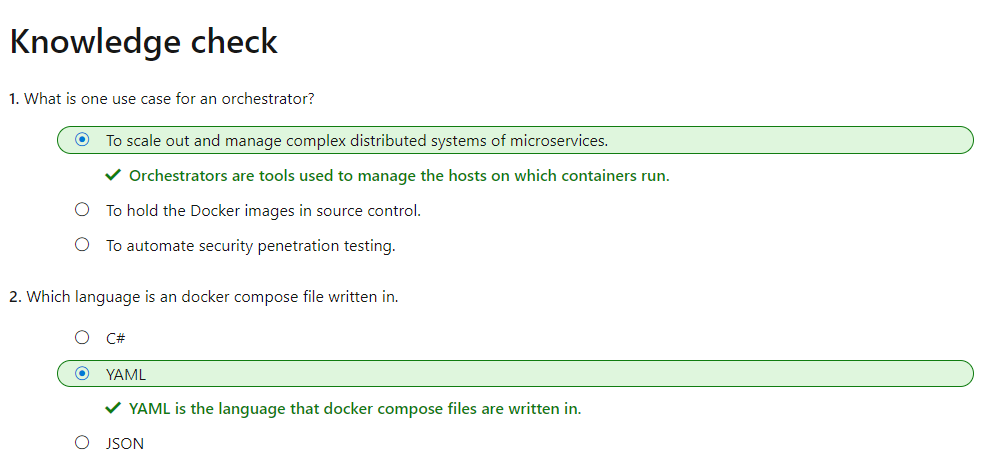
* Docker containers are "units of deployment," and a container is an instance of a Docker image. A host handles many containers. It looks like a logical approach. An orchestrator helps deploy the containers and manage the hosts.
* **Why would you use an orchestrator?**
* Help composing applications consisting of may microservices into one deployable unit.
* Units is moved or deployed to host/vms
* Orchestrator managing the host
* It can automatically start the container, scale them out with multiple instances per image,
* Suspend them and shud them down when needed.
* Can also control how containers access resources like network and data

**Docker Compose**

* Full-featured orchestrator might be to much for the simple website
* We need a tool that lets them build multiple docker images together as a single unit and then deploy that unit
* use YAML file to configure an applications services.
* Docker compose tool provide a way to build the individual services as means to start them

**Note**

The Docker compose tool gets installed by default with the Docker for desktop which makes it easier to run all containers along with their dependencies locally. For production scenarios, we recommend using Orchestrators that give finer control over automatic deployment, scaling, and management of containers.



Docker-compose.yml

version: '3.4'

# lito source .NET

# https://docs.microsoft.com/en-us/learn/modules/dotnet-microservices/5-exercise-create-docker-compose-file

# First, it creates the frontend website, naming it pizzafrontend.

# The code tells Docker to build it, pointing to the Dockerfile found in the frontend folder.

# Then the code sets an environment variable for the website: backendUrl=http://backend.

# Finally, this code opens a port and declares it depends on the backend service.

services:

  frontend:

    image: pizzafrontend

    build:

      context: frontend

      dockerfile: Dockerfile

    environment:

      - backendUrl=http://backend

    ports:

      - "5902:80"

    depends\_on:

      - backend

# The backend service gets created next. It's named pizzabackend.

# It's built from the same Dockerfile you created in the previous exercise. The last command specifies which port to open.

  backend:

    image: pizzabackend

    build:

      context: backend

      dockerfile: Dockerfile

    ports:

      - "5000:80"

docker-compose build

docker-compose up

D:\git\.NET\Microservices\mslearn-dotnetmicroservices\mslearn-dotnetmicroservices>docker-compose up

[+] Running 3/2

- Network mslearn-dotnetmicroservices\_default Created 0.0s

- Container mslearn-dotnetmicroservices-backend-1 Created 0.1s

- Container mslearn-dotnetmicroservices-frontend-1 Created 0.0s

Attaching to mslearn-dotnetmicroservices-backend-1, mslearn-dotnetmicroservices-frontend-1

mslearn-dotnetmicroservices-backend-1 | info: Microsoft.Hosting.Lifetime[14]

mslearn-dotnetmicroservices-backend-1 | Now listening on: http://[::]:80

mslearn-dotnetmicroservices-backend-1 | info: Microsoft.Hosting.Lifetime[0]

mslearn-dotnetmicroservices-backend-1 | Application started. Press Ctrl+C to shut down.

mslearn-dotnetmicroservices-backend-1 | info: Microsoft.Hosting.Lifetime[0]

mslearn-dotnetmicroservices-backend-1 | Hosting environment: Production

mslearn-dotnetmicroservices-backend-1 | info: Microsoft.Hosting.Lifetime[0]

mslearn-dotnetmicroservices-backend-1 | Content root path: /app

mslearn-dotnetmicroservices-frontend-1 | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]

mslearn-dotnetmicroservices-frontend-1 | Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will be unavailable when container is destroyed.

mslearn-dotnetmicroservices-frontend-1 | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]

mslearn-dotnetmicroservices-frontend-1 | No XML encryptor configured. Key {85dfc819-96ca-48ff-bedc-78cf4495a535} may be persisted to storage in unencrypted form.

mslearn-dotnetmicroservices-frontend-1 | info: Microsoft.Hosting.Lifetime[14]

mslearn-dotnetmicroservices-frontend-1 | Now listening on: http://[::]:80

mslearn-dotnetmicroservices-frontend-1 | info: Microsoft.Hosting.Lifetime[0]

mslearn-dotnetmicroservices-frontend-1 | Application started. Press Ctrl+C to shut down.

mslearn-dotnetmicroservices-frontend-1 | info: Microsoft.Hosting.Lifetime[0]

mslearn-dotnetmicroservices-frontend-1 | Hosting environment: Production

mslearn-dotnetmicroservices-frontend-1 | info: Microsoft.Hosting.Lifetime[0]

mslearn-dotnetmicroservices-frontend-1 | Content root path: /app

mslearn-dotnetmicroservices-frontend-1 | info: System.Net.Http.HttpClient.PizzaClient.LogicalHandler[100]

mslearn-dotnetmicroservices-frontend-1 | Start processing HTTP request GET http://backend/pizzainfo

mslearn-dotnetmicroservices-frontend-1 | info: System.Net.Http.HttpClient.PizzaClient.ClientHandler[100]

mslearn-dotnetmicroservices-frontend-1 | Sending HTTP request GET http://backend/pizzainfo

mslearn-dotnetmicroservices-frontend-1 | info: System.Net.Http.HttpClient.PizzaClient.ClientHandler[101]

mslearn-dotnetmicroservices-frontend-1 | Received HTTP response headers after 120.9297ms - 200

mslearn-dotnetmicroservices-frontend-1 | info: System.Net.Http.HttpClient.PizzaClient.LogicalHandler[101]

mslearn-dotnetmicroservices-frontend-1 | End processing HTTP request after 127.5423ms – 200

# Summary

Completed100 XP

* 3 minutes

The cloud has changed how developers build applications.

Some of the changing business needs include:

* A service that's built and operated at scale to reach customers in new geographic regions
* Faster delivery of features and capabilities to respond to customer demands in an agile way
* Improved resource utilization to reduce costs

Microservice architecture helps address those needs by building a single application that is composed of a set of small services. Each microservice implements a specific end-to-end domain or business capability within a certain context boundary, and each must be developed autonomously and be deployable independently.

Containers are a great approach to building microservices. Each application or service, its dependencies, and its configuration (abstracted as deployment manifest files) are packaged together as a container image. **And Docker is a popular way of containerizing applications.**

A Docker file is a text file that contains instructions on how to build a Docker image, and a **Docker compose file is a YAML file that groups together several docker containers for build and deployment.**

Finally, .NET was built to be cloud native first. It runs cross-platform, so your Docker image could be based on a flavor of Linux and your .NET code will still run. In addition, there are already .NET images created for Docker by Microsoft. Plus, .NET is extremely fast, with the ASP.NET's Kestrel web server routinely outperforming other web servers.

# Deploy a .NET microservice to Kubernetes

<https://www.youtube.com/watch?v=Z-2Bl6Z7Ego>

<https://docs.microsoft.com/en-us/learn/modules/dotnet-deploy-microservices-kubernetes/>

Microservice applications are composed of small, independently versioned, and scalable customer-focused services.

Microservices applications deployed in containers make it possible to scale out apps, and respond to increased demand by deploying more container instances, and to scale back if demand is decreasing.

In complex solutions of many microservices

* the process of deploying,
* updating,
* monitoring,
* and removing containers introduces challenges.
* This module explains some of those challenges and shows how Kubernetes can help.

<https://www.youtube.com/watch?v=Z-2Bl6Z7Ego>

Verify the Docker Images by creating containers locally

* Two containers images in the Contoso Pizza Shop project
* Lets use them to create the containers locally before pushing the Images to Docker Hub
* Once the container are created and running we shall be able to browse the website
* Verify the miscroservices are running OK.

Introduction

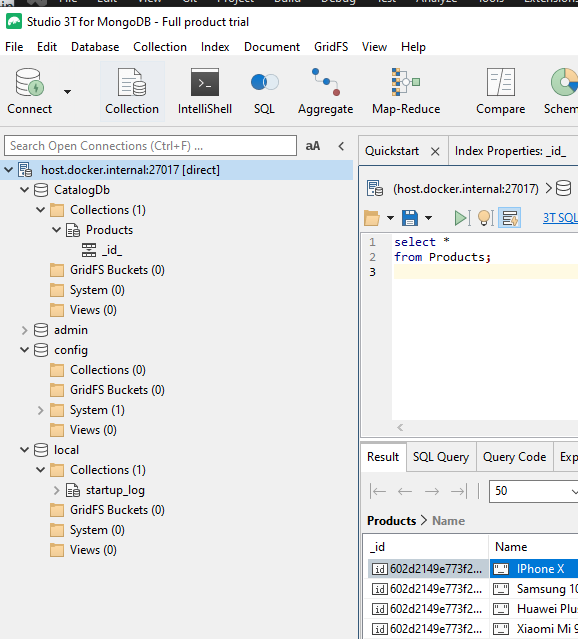
* pizzas are in stock or not has recently been refactored into microservices hosted in Docker containers.

**Kubernetes** is one such orchestrator

* It is an extensible open-source platform for managing and orchestrating containerized workloads.
* In a microservice-based development approach, each microservice owns its model and data so that it will be autonomous from a development
* and deployment point of view from other microservices.
* Hosting microservices inside of a container is a common way to achieve that. These kinds of systems are complex to scale out and manage.
* You need to consider the process of organizing, adding, removing, and updating many containers. **This process is referred to as container management.**
* For example, you may find during specific times of the **day you need to scale up the number of container instances that handle caching.**
* Or you may have **an update to the container instance that checks pizza inventory.**

https://docs.microsoft.com/en-us/training/modules/dotnet-deploy-microservices-kubernetes/2-what-are-orchestrators

**What are orchestrators?**



<https://github.com/aspnetrun/run-aspnetcore-microservices>

D:\git\.NETCore\.NETCore\_microservices\src

#### Catalog microservice which includes;

* ASP.NET Core Web API application
* REST API principles, CRUD operations
* **MongoDB database** connection and containerization
* Repository Pattern Implementation
* Swagger Open API implementation

#### Basket microservice which includes;

* ASP.NET Web API application
* REST API principles, CRUD operations
* **Redis database** connection and containerization
* Consume Discount **Grpc Service** for inter-service sync communication to calculate product final price
* Publish BasketCheckout Queue with using **MassTransit and RabbitMQ**

<https://github.com/aspnetrun/run-aspnetcore-microservices>

---Visual Studio 2019

.Net Core 5 or later

https://github.com/aspnetrun/run-aspnetcore-microservices

Microservices Architecture and Implementation on .NET 5

https://github.com/aspnetrun/run-aspnetcore-microservices

https://github.com/aspnetrun/run-aspnetcore-microservices

(e.g.

well-designed APIs,

high volume data pipelines,

service-meshes,

message queues,

distributed caching,

efficient algorithms).

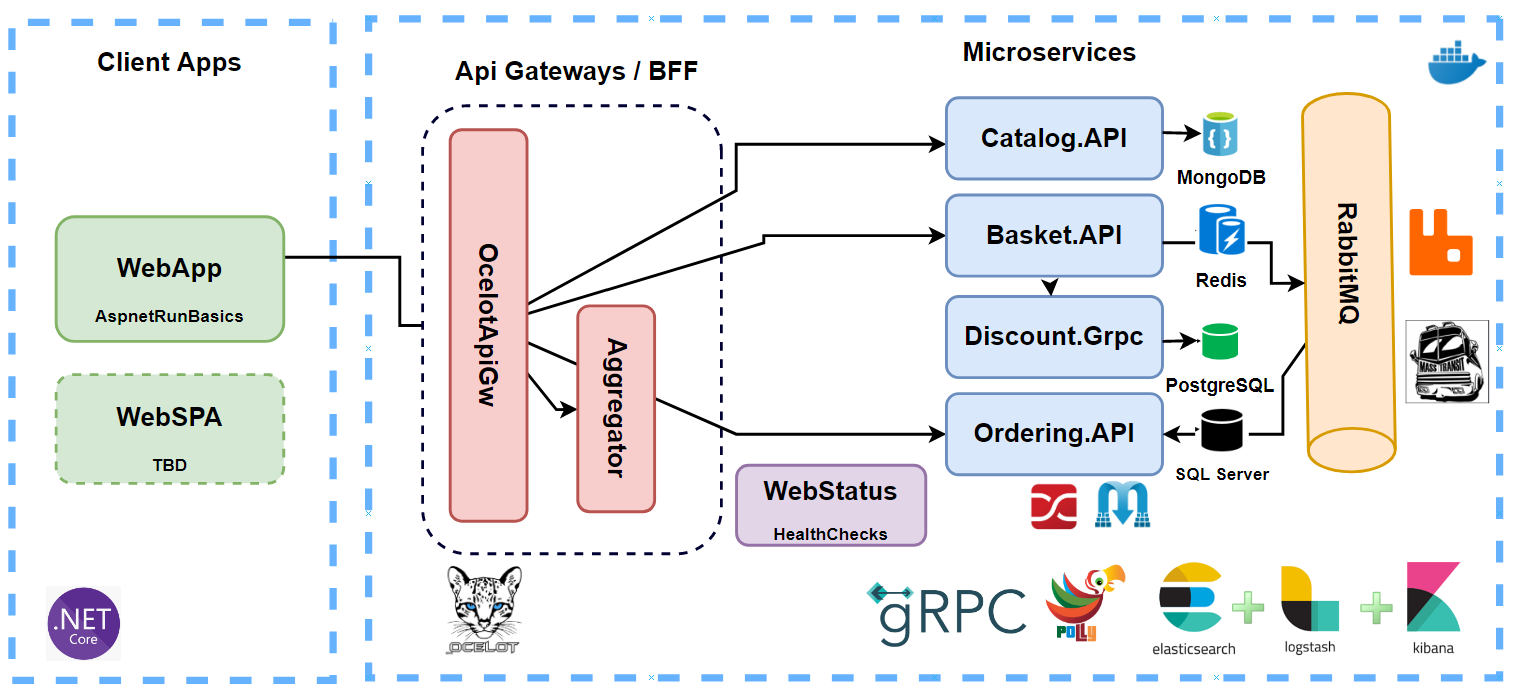
--STUDY this LITO

D:\git\.NET\Microservices\AspCoreMicroservices\run-aspnetcore-microservices\src

D:\git\.NET\Microservices\AspCoreMicroservices\run-aspnetcore-microservices\src

D:\git\.NET\Microservices\AspCoreMicroservices\run-aspnetcore-microservices\src

Overall picture of implementations on microservices with .net tools on real-world e-commerce microservices project;



Microservices – implemented e-commerce modules over Catalog, Basket, Discount and Ordering with NoSQL ( MongoDB, Redis ) and Relational DB ( Postgres, MSSQL) with communicating over the RabbitMQ Event Driven Communications and using Ocelot API Gateway.

Including : Cache Controls DB Redis.

Microservices Observability with Distributed Logging, Health Monitoring, Resilient and Fault Tolerance with using Polly

Microservices Architecture on .NET with applying CQRS,

Clean Architecture and Event-Driven Communication

Microservices Observability,

Resilience, Monitoring on .Net

NETCore\_microservices ( NoSQL(MongoDB,Redis) and Postgres/MSSQL communicating over the RabbitMQ Eventdriven Communications and Ocelot API Gateway

* Cache Control (Redis)

Also using Microservices Observability with Distributed Logging, Health Monitoring, Resilient and Fault Tolerance with using Polly

Microservices Architecture on .NET with applying CQRS,

Clean Architecture and Event-Driven Communication

Microservices Observability,

Resilience, Monitoring on .Net

I have order the implementation of the Miscroservices in sequence.

1. Catalog

ASP.NET Core Web API application

REST API principles, CRUD operations

MongoDB database connection and containerization

Repository Pattern Implementation

Swagger Open API implementation

2. Discount

ASP.NET Grpc Server application

Build a Highly Performant inter-service gRPC Communication with Basket Microservice

Exposing Grpc Services with creating Protobuf messages

Using Dapper for micro-orm implementation to simplify data access and ensure high performance

PostgreSQL database connection and containerization

3. Basket

ASP.NET Web API application

REST API principles, CRUD operations

Redis database connection and containerization

**Consume Discount Grpc Service for inter-service sync communication to calculate product final price**

**Publish BasketCheckout Queue with using MassTransit and RabbitMQ**

Dotnet build

docker-compose -f docker-compose.yml -f docker-compose.override.yml up -d

Basket API -> http://host.docker.internal:8001/swagger/index.html

Rabbit Management Dashboard -> http://host.docker.internal:15672 -- guest/guest