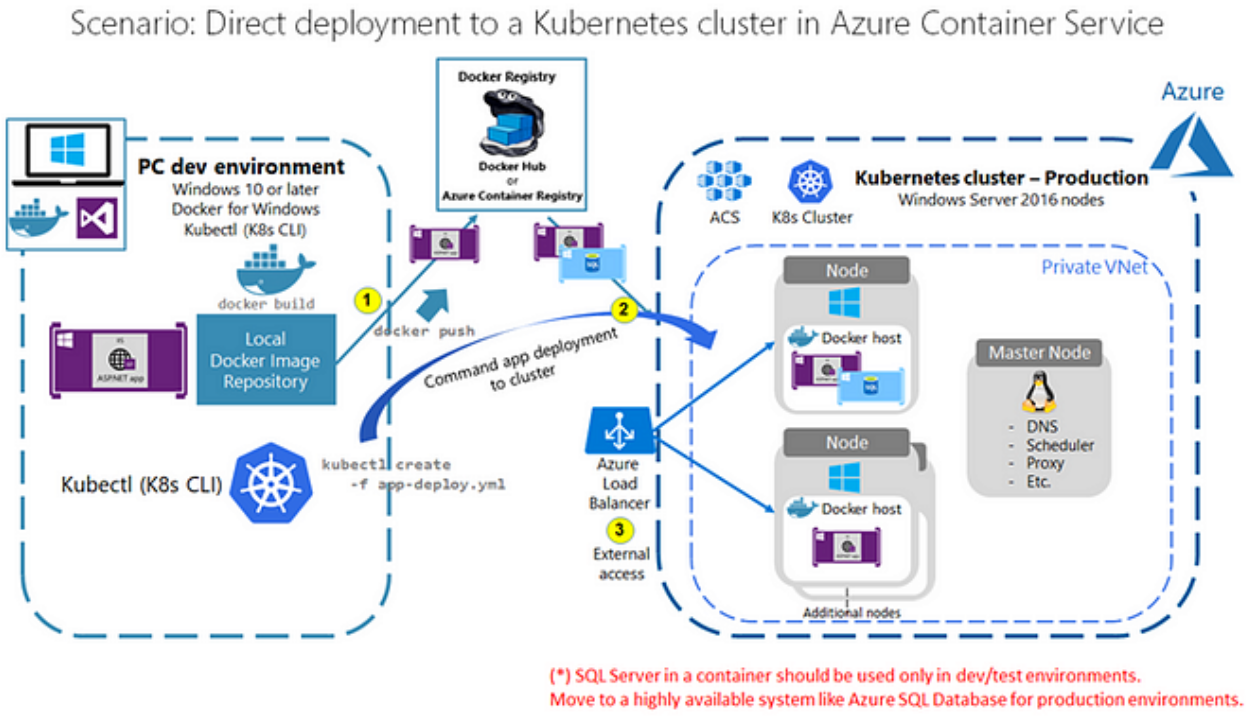
Deploying .Net Microservices to Azure Kubernetes Services(AKS) and Automating with Azure DevOps

In this article, we're going to learn how to Deploying .Net Microservices into **Kubernetes**, and moving deployments to the cloud **Azure Kubernetes Services (AKS)** with using **Azure Container Registry (ACR)** and last section is we will learn how to Automating Deployments with **Azure DevOps** and GitHub.



The image above, you can find the steps of our article structure.
We're going to containerize our microservices on docker environment, and push these images to the **DockerHub** and deploy microservices on **Kubernetes**. As the same setup, we are going to shifting to the cloud for deploying **Azure Kubernetes Services (AKS)** with pushing images to **Azure Container Registry (ACR).**

Also we will cover additional topics that;

- Docker compose microservices
- K8s components
- Zero-downtime deployments
- Using azure resources like ACR, AKS
- Automate whole deployment process with writing custom pipelines with Azure DevOps and so on..

Step by Step Development w/ Course

IT & Software > Other IT & Software > Kubernetes

Deploying .Net Microservices with K8s, AKS and Azure DevOps

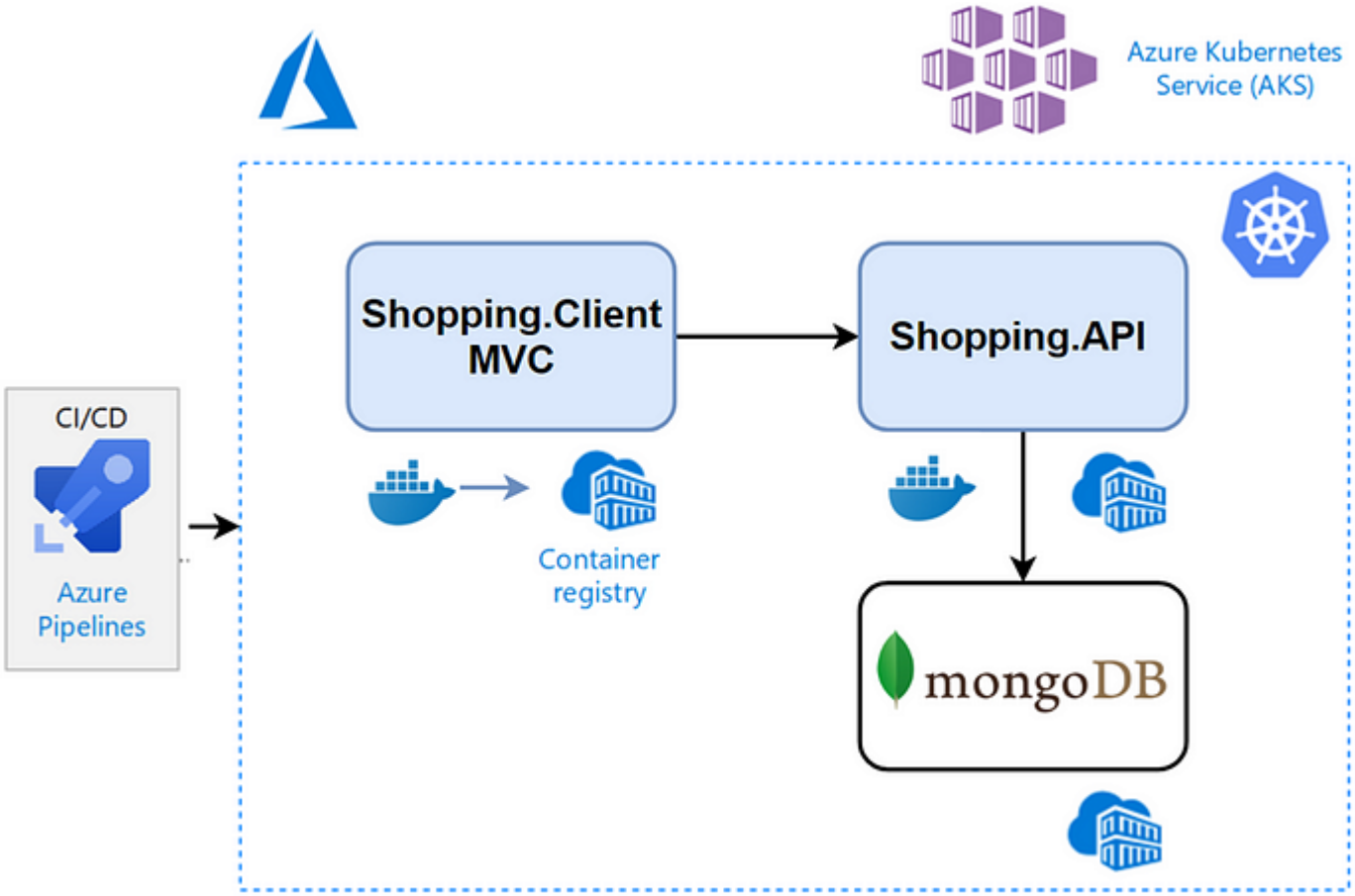
Deploying .Net Microservices to Kubernetes, move cloud Azure Kubernetes Services(AKS), Automating with Azure DevOps

I have just published a new course — Deploying .Net Microservices with K8s, AKS and Azure DevOps.

In this course, we're going to learn how to **Deploying .Net Microservices** into **Kubernetes**, and moving deployments to the cloud **Azure kubernetes services (AKS)** with using **Azure Container Registry(ACR)** and last section is we will learn how to Automating Deployments with **CI/CD pipeline of Azure DevOps** and GitHub.

Overall Picture

See the overall picture. You can see that we will have 3 microservices which we are going to develop and deploy together.



Shopping MVC Client Application

First of all, we are going to develop Shopping MVC Client Application For Consuming Api Resource which will be the Shopping.Client Asp.Net MVC Web Project. But we will start with developing this project as a standalone Web application which includes own data inside it. And we will add container support with **DockerFile**, push docker images to Docker hub and see the deployment options like "Azure Web App for Container" resources for 1 web application.

Shopping API Application

After that we are going to develop Shopping.API Microservice with MongoDb and Compose All Docker Containers.
This API project will have Products data and performs CRUD operations with exposing api methods for consuming from Shopping Client project.
We will containerize API application with creating dockerfile and push images to Azure Container Registry.

Mongo Db

Our API project will manage product records stored in a no-sql mongodb database as described in the picture.
we will pull **mongodb** docker image from docker hub and create connection with our API project.

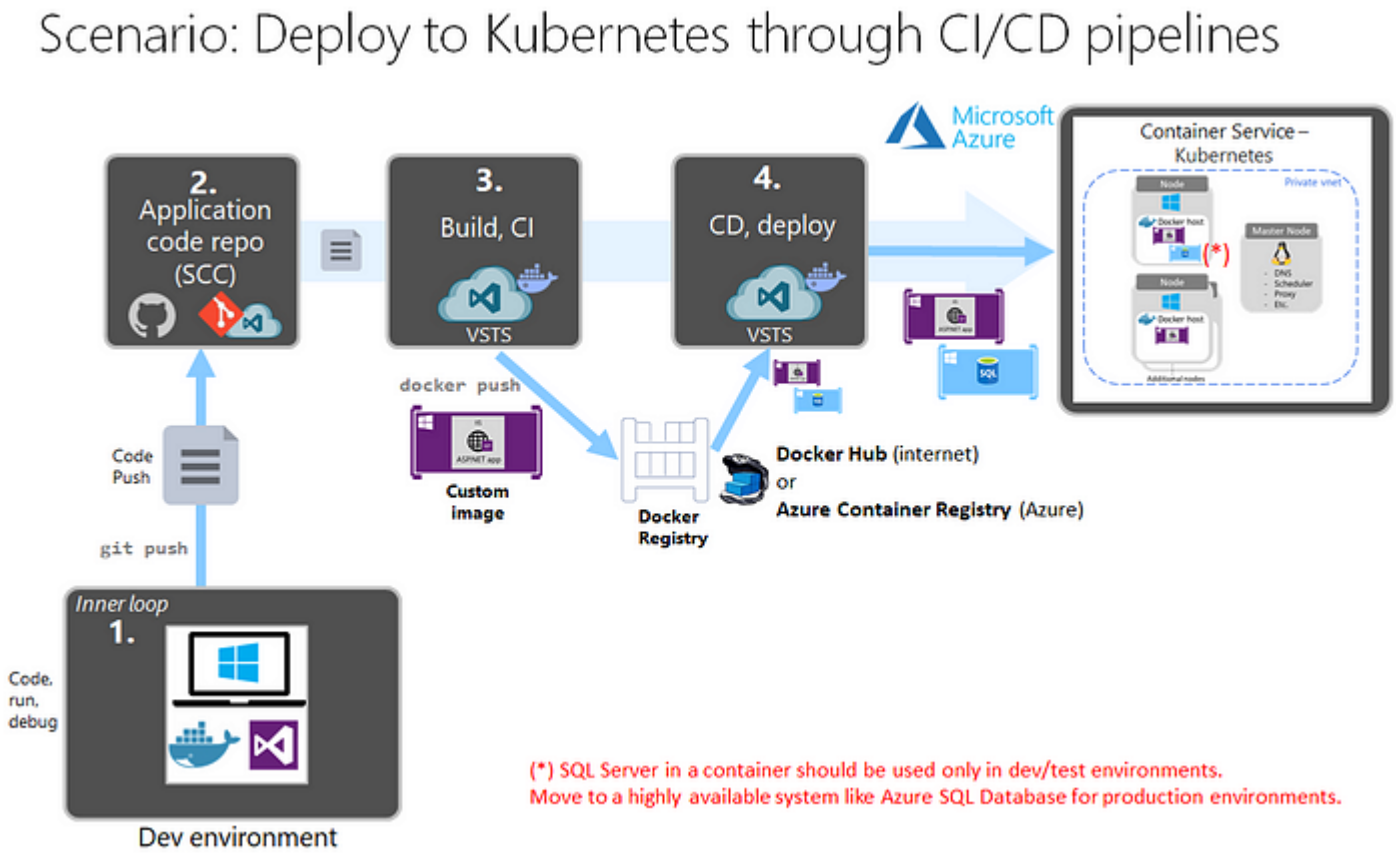
At the end of the section, we will have 3 microservices whichs are **Shopping.Client — Shopping.API — MongoDb** microservices.

As you can see that, we have

- Created docker images,
- Compose docker containers and tested them,
- Deploy these docker container images on local Kubernetes clusters,
- Push our image to ACR,
- Shifting deployment to the cloud Azure Kubernetes Services (AKS),
- Update microservices with **zero-downtime** deployments.

Deploy to Azure Kubernetes Services (AKS) through CI/CD Azure Pipelines

And the last step, we are focusing on automation deployments with creating **CI/CD pipelines** on **Azure Devops** tool. We will develop separate microservices deployment pipeline yamls with using **Azure Pipelines**.
When we push code to **Github**, microservices pipeline triggers, build docker images and push the ACR, deploy to Azure Kubernetes services with zero-downtime deployments.



By the end of this articles, you'll learn how to deploy your multi-container microservices applications with automating all deployment process seperately.

Background

This is the introduction of the series. This will be the series of articles. You can follow the series with below links.

Prerequisites

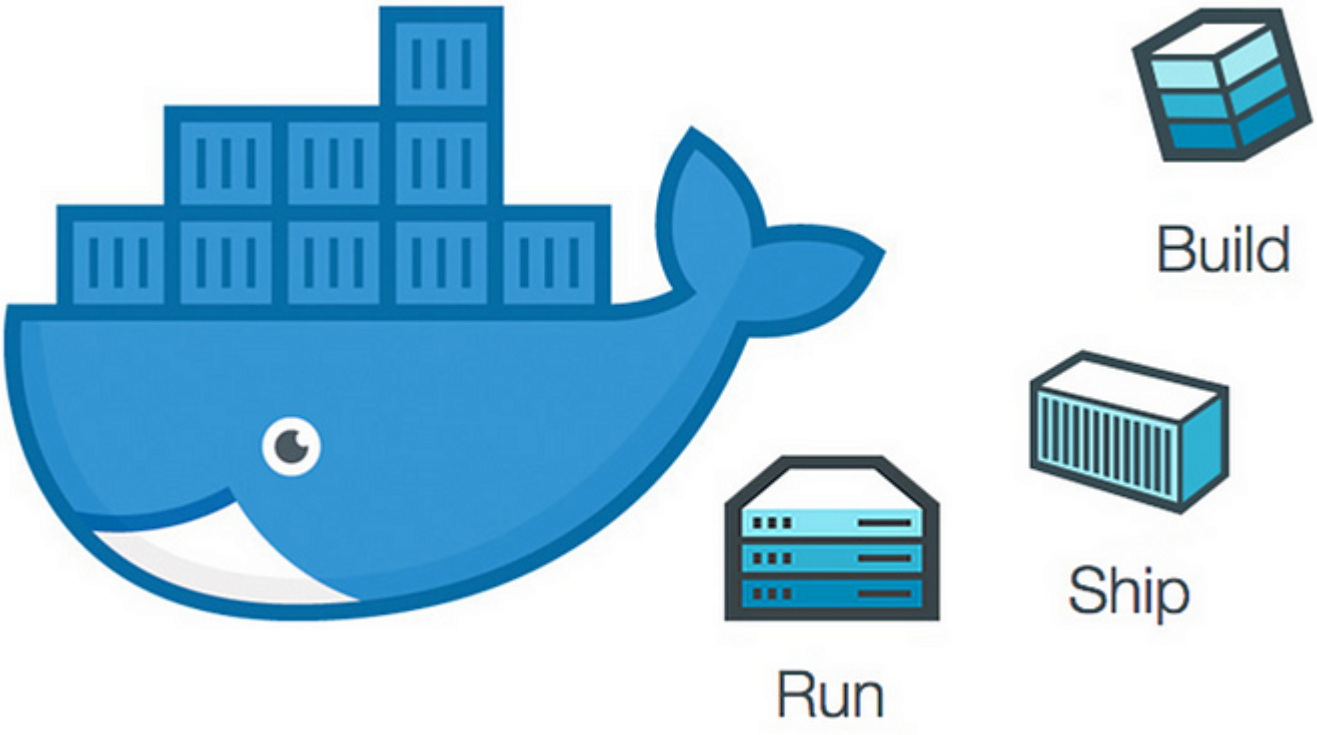
- Visual Studio 2019 and VS Code
- Docker Desktop and Docker Account for pushing images Docker Hub
- Git on Local and Github Account for granting our devops pipelines triggering when push the code.
- Azure Free Subscription for creating all azure resources like ACR, Web app for Containers, AKS and so on..
- Azure Devops Account for ci/cd devops pipelines.

Source Code

Get the Source Code from AspnetRun Microservices Github — Clone or fork this repository, if you like don't forget the star :) If you find or ask anything you can directly open issue on repository.

Basics — What is Docker and Container ?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.



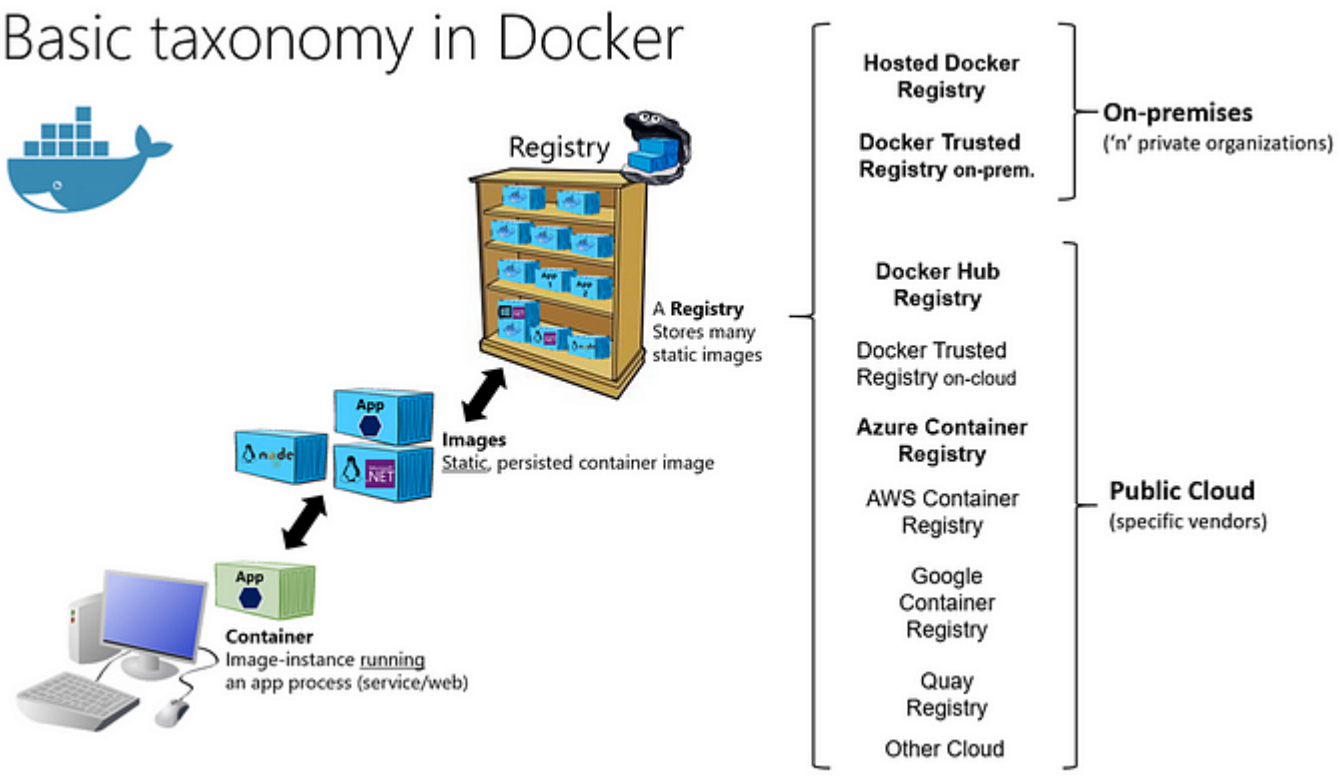
Advantages of Docker's methodologies for **shipping, testing,** and **deploying** code quickly, you can significantly reduce the delay between writing code and running it in production. Docker provides for automating the deployment of applications as portable, self-sufficient containers that can run on the cloud or on-premises. Docker containers can run anywhere, in your local computer to the cloud. Docker image containers can run natively on Linux and Windows.

Docker Container

A container is a standard unit of software that packages up code and **all its dependencies** so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, **executable package** of software that includes everything needed to run an application.

Docker containers, images, and registries

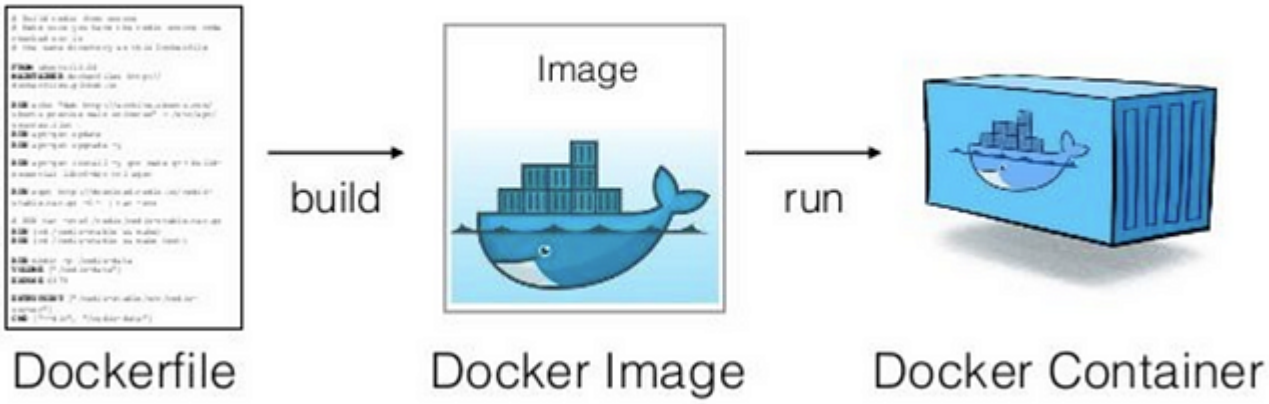
When using Docker, a developer develops an application and packages it with its dependencies into a container image. An image is a static representation of the application with its configuration and dependencies.



In order to run the application, the application's image is instantiated to create a container, which will be running on the Docker host.
Containers can be tested in a development local machines.

As you can see the images above, how docker components related each other.
Developer creates container in local and push the images the Docker Registry.
Or its possible that developer download existing image from registry and create container from image in local environment.

Developers should store images in a registry, which is a library of images and is needed when deploying to production orchestrators. Docker images are stores a public registry via Docker Hub; other vendors provide registries for different collections of images, including Azure Container Registry.
Alternatively, enterprises can have a private registry on-premises for their own Docker images.



If we look at the more specific example of Application Containerization with Docker;

- First, we should write dockerfile for our application,
- Build application with this docker file and creates the docker images.
- And lastly run this images on any machine and creates running docker container from docker image.

We will use all steps with orchestrating whole microservices application with docker and Kubernetes for the next articles ->