# DOC1 Course Assignment

## Contents



## The VIA Tabloid Application - VIATAB

The main focus of the course assignment is on building a VIA tabloid application that adds sensational stories, read, update and deletes stories, from the different departments here in Horsens following DevOps practices.

In order to make the design and technology decisions in the course more concrete, it is expected that you will implement an example application called the VIA Tabloid application. This application will be implemented incrementally throughout the course.

The VIA-Tabloid application is a web application that displays captions of sensational stories from the different departments (at least 3) in VIA. The department where the respective stories/ideas is displayed could be placed on the page using different components or on different tabs.

The application should have at least three components:

1. Frontend (React, TypeScript)
2. Backend (Java Spring Boot or C# .NET)
3. Database (Postgres or MySQL, MongoDB)

A working frontend is required for this app, and should implement the features for your tabloid app; add story item, delete item, etc. It does not need to be especially pretty but there needs to be a frontend that connects with a backend, server, etc. You can use React or any frontend framework you like, but again it must have a real connection to your backend. We will not be particular on how the frontend looks, rather the use of DevOps throughout the assignment.

## Part 1 – Docker Introduction (Session 5 → Session 6)

Like all programming problems, learning a new technology is not an exercise in reading but rather an exercise in thinking and coding. This Part 1 is designed to give you an opportunity to try hands-on experience in some fundamental skills involved in Docker containerization.

**WHAT YOU NEED TO DO:**

You will practice creating Docker images and containers

1. Get React frontend working in a Docker Container
2. Get Spring boot backend working in a Docker Container
3. Get PostgreSQL working in a Docker Container

## Part 2 – Running the whole App with Docker Compose (Session 6 → Session 7)

With docker-compose, we can describe our build and run instructions in a file and do it for more than one container at a time.

**WHAT YOU NEED TO DO:**

You will create docker-compose yaml file that will orchestrate the backend and the database using container images (The frontend will be added later in the course).

1. Create the Compose file for Docker Compose. This is done by creating a file called `docker-compose.yml` at the root of the project.
2. Build and run the app stack. From your project directory, start up your application by running `docker compose up`.
3. Verify that the application stack is running as expected (Test as well as looking on the Docker Dashboard).
4. Stop the application, either by running `docker compose down` from within your project directory in the second terminal, or by hitting `CTRL+C` in the original terminal where you started the app.

## Part 3 – Deploy to Kubernetes (Session 9 → Session 10)

Kubernetes is a better option for large production-grade deployments because of its ability to manage and deploy large numbers of containers on multiple hosts.

In this exercise, we'll create and run a Kubernetes cluster locally with Minikube and deploy the applications and database to Kubernetes.

**WHAT YOU NEED TO DO:**

You will create Kubernetes resources including deployment and service resource files.

1. Create a deployment resource (**deployment.yaml**) - telling Kubernetes what container you want, how they should be running and how many of them should be running.
2. Create a service resource (**service.yaml**) – which will take care of serving the application to connections from outside of the cluster.
3. Create any other resource that might need. For example, PV, PVC, ConfigMap, etc
4. Apply the resource files and verify your deployment.

```
$ kubectl apply -f pathtoyourfiles/deployment.yaml
$ kubectl apply -f pathtoyourfiles/service.yaml
```

5. Use minikube commands and the *minikube dashboard* to see more details.

## Part 4 – Create CI/CD Pipeline (Session 10 → Session 11)

In this exercise, we'll create Continuous Integration (CI) and Continuous Deployment (CD) workflow using Github Actions. This will automatically run the tests, build images and deploy the applications to Kubernetes cluster using Minikube, either on a pull request or a push to the main/master branch. Feel free to use your preferred strategy.

Before you continue, modify your Kubernetes deployment resource file with **imagePullPolicy: Never** in order to use the existing image on the node.

**WHAT YOU NEED TO DO:**

You will create the CI/CD workflow for the project either in separate yaml files or merged to one file.

1. Create a Continuous Integration workflow (eg:, **ci-workflow.yaml**) in the project root as follows: .github/workflow/ci-workflow.yaml.
2. Add the actions to run tests, build and publish image.
3. Commit your changes and ensure that you get to a "green state" 😊
4. Create a Continuous Deployment workflow (eg:, **cd-workflow.yaml**) as before or merge the two files into one ci-cd-workflow.yaml.
5. Add jobs/actions to automatically deploy to Minikube.
6. Commit your changes and observe the workflow in the GitHub repository under "Actions".