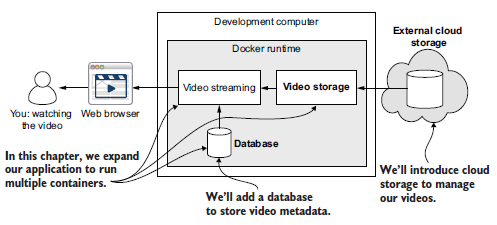
# Data\_Management\_for\_Microservices

This guide provides a step-by-step approach to implementing a microservices-based video streaming application using Docker Compose, AWS S3, and MongoDB.

We will:

* Add file storage for our application to store its videos.
* Add a database to record the metadata for each video

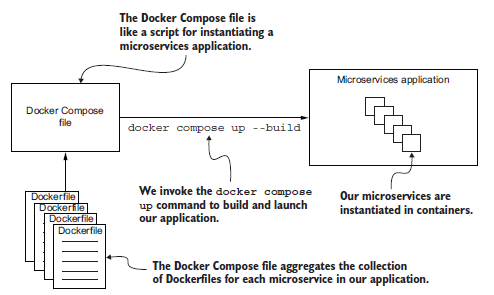
We will have three containers:



With more than one microservice we will start using Docker Compose

* A convenient way to build, run, and manage multiple microservices in development. (to avoid having to open multiple terminals (one for each microservice) and then running each microservice separately using Node.js)

The ***Docker Compose*** *file* is a script that specifies how to compose an application from multiple Docker containers.

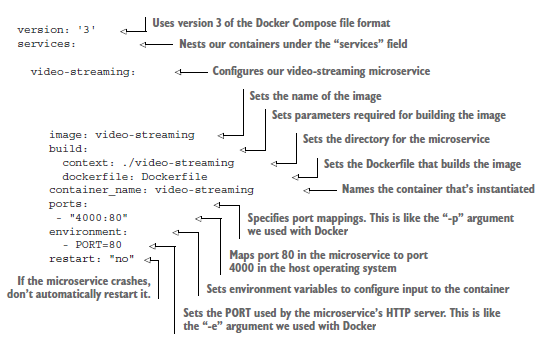


## Step 1: Using Docker Compose

The folder ‘using\_docker\_compose’ is an example of docker compose applied to our previous application, we have:

* Moved the Dockerfile and code for our microservice into a new subdirectory ‘video-streaming’ (to match the name of the microservice as each microservice is moved into its own subdirectory)
* Created our Docker Compose file ‘docker-compose.yaml’ in the root directory of

our microservices application:



version: '3'

services:

video-streaming:

image: video-streaming

build:

context: ./video-streaming

dockerfile: Dockerfile

container\_name: video-streaming

ports:

- "4000:80"

environment:

- PORT=80

restart: "no"

from the ‘using\_docker\_compose’ directory execute:

docker compose up –build

* The --build argument isn’t necessary because the first time we invoke the up command, it builds our images
* At other times (without the --build argument), the up command just starts our container from the image that was previously built (this can be a quick way to restart if we don’t want to rebuild)

Point your browser to ‘http://localhost:4000/video’ to watch the video

The ps command shows a list of our running containers.

docker compose ps

(only relate to the images and containers specified in our Docker Compose files)

To stop the application (doesn’t remove the containers):

docker compose stop

(or Ctrl-C at the terminal where you invoked the up command)

To stop and remove containers

docker compose down

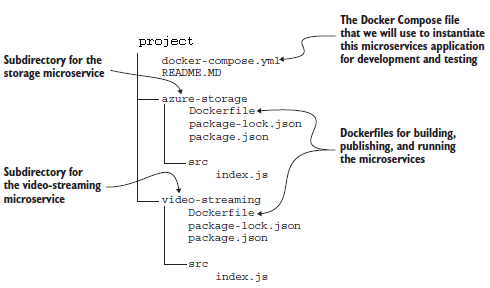
To also delete the volumes:

docker compose down --volumes

to easily reboot our application (updated code or dependencies)

docker compose down --volumes && docker compose up --build

## Step 2: Creating an Azure Storage



To directly run the project in navigate to ‘azure-storage’ and then install the dependencies:

npm install

set the environment variables

set PORT=3000

set STORAGE\_ACCOUNT\_NAME=<the name of your storage account>

set STORAGE\_ACCESS\_KEY=<the access key for your storage account>npm

Run the microservice in development mode:

npm run start:dev

## Step 1: Set Up MongoDB and Microservices

### Define docker-compose.yml

version: '3'  
services:  
  
 db:  
 image: mongo:7.0.0  
 container\_name: db  
 ports:  
 - "4000:27017"  
 restart: always  
  
 aws-storage:  
 build: ./aws-storage  
 ports:  
 - "4001:80"  
 environment:  
 - PORT=80  
 - STORAGE\_ACCOUNT\_NAME=${STORAGE\_ACCOUNT\_NAME}  
 - STORAGE\_ACCESS\_KEY=${STORAGE\_ACCESS\_KEY}  
  
 video-streaming:  
 build: ./video-streaming  
 ports:  
 - "4002:80"  
 environment:  
 - PORT=80  
 - DBHOST=mongodb://db:27017  
 - DBNAME=video-streaming  
 - VIDEO\_STORAGE\_HOST=video-storage  
 - VIDEO\_STORAGE\_PORT=80  
 depends\_on:  
 - db

## Step 2: Set Up the Video Streaming Microservice

### 2.1 Install Dependencies

npm install express mongodb http

### 2.2 Create index.js

const express = require("express");  
const http = require("http");  
const mongodb = require("mongodb");  
  
const PORT = process.env.PORT || 80;  
const VIDEO\_STORAGE\_HOST = process.env.VIDEO\_STORAGE\_HOST;  
const VIDEO\_STORAGE\_PORT = process.env.VIDEO\_STORAGE\_PORT;  
const DBHOST = process.env.DBHOST;  
const DBNAME = process.env.DBNAME;  
  
const client = new mongodb.MongoClient(DBHOST);  
  
async function getVideoPath(id) {  
 await client.connect();  
 const db = client.db(DBNAME);  
 const video = await db.collection("videos").findOne({ \_id: id });  
 return video ? video.videoPath : null;  
}  
  
const app = express();  
app.get("/video/:id", async (req, res) => {  
 const videoPath = await getVideoPath(req.params.id);  
 if (!videoPath) return res.status(404).send("Video not found");  
  
 http.get(`http://${VIDEO\_STORAGE\_HOST}:${VIDEO\_STORAGE\_PORT}/video/${videoPath}`, (storageRes) => {  
 res.writeHead(storageRes.statusCode, storageRes.headers);  
 storageRes.pipe(res);  
 });  
});  
app.listen(PORT, () => console.log(`Video Streaming running on ${PORT}`));

## Step 3: Set Up the AWS Storage Microservice

### 3.1 Install Dependencies

npm install express @azure/storage-blob

### 3.2 Create index.js

const express = require("express");  
const { BlobServiceClient } = require("@azure/storage-blob");  
  
const PORT = process.env.PORT || 80;  
const STORAGE\_ACCOUNT\_NAME = process.env.STORAGE\_ACCOUNT\_NAME;  
const STORAGE\_ACCESS\_KEY = process.env.STORAGE\_ACCESS\_KEY;  
  
const app = express();  
app.get("/video/:filename", async (req, res) => {  
 const blobServiceClient = BlobServiceClient.fromConnectionString(STORAGE\_ACCOUNT\_NAME);  
 const containerClient = blobServiceClient.getContainerClient("videos");  
 const blobClient = containerClient.getBlobClient(req.params.filename);  
 const downloadBlockBlobResponse = await blobClient.download();  
  
 res.setHeader("Content-Type", "video/mp4");  
 downloadBlockBlobResponse.readableStreamBody.pipe(res);  
});  
app.listen(PORT, () => console.log(`Azure Storage running on ${PORT}`));

## Step 4: Initialize MongoDB with Sample Data

### 4.1 Insert Video Metadata

mongo --host localhost --port 4000 --eval '  
db = db.getSiblingDB("video-streaming");  
db.videos.insert({ \_id: "123", videoPath: "sample.mp4" });  
'

## Step 5: Run the Application

docker compose up --build

## Conclusion

This guide demonstrates how to build a microservices-based video streaming application that integrates cloud storage and a database using Docker Compose.