Docker and MongoDB

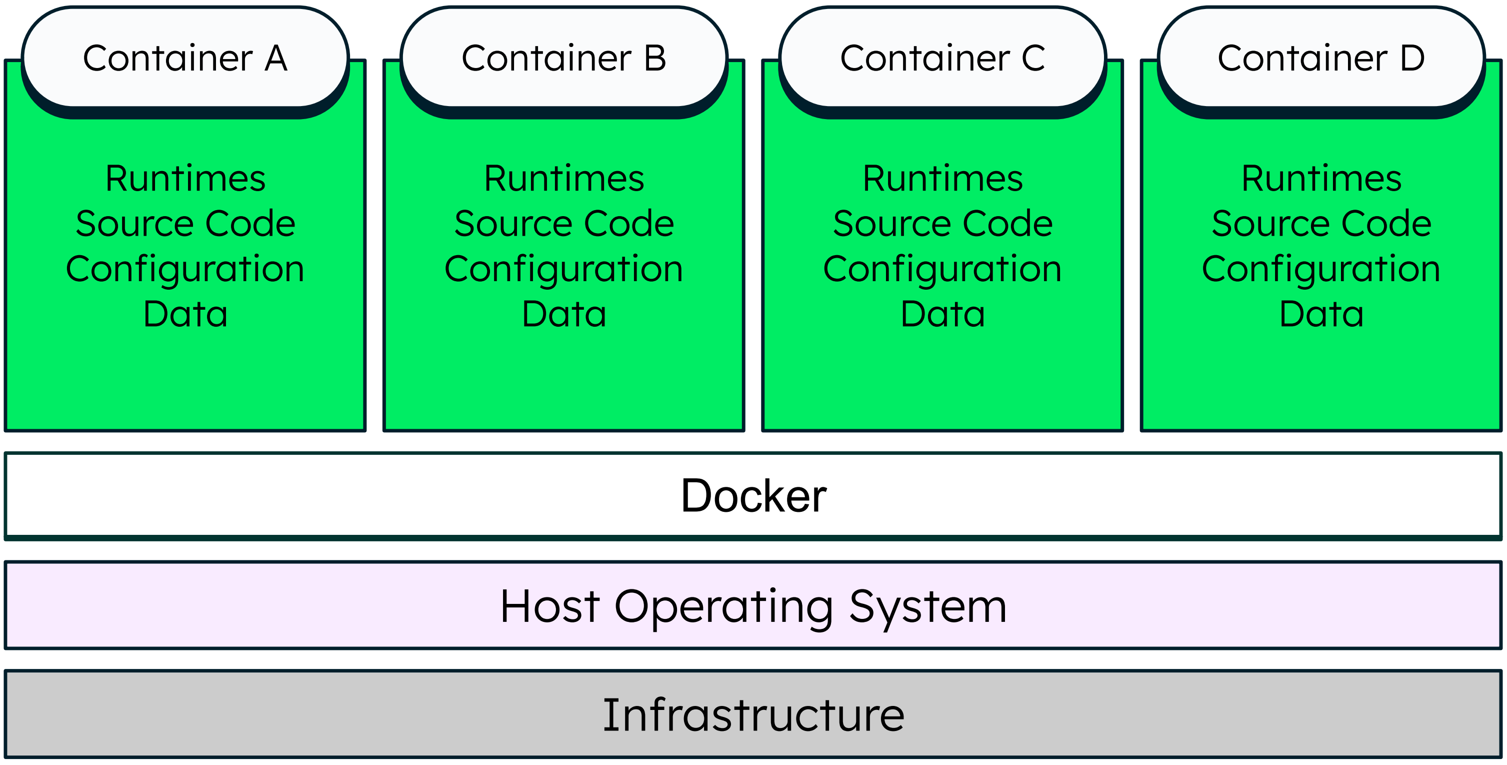
The company Docker has been around since 2013, and they popularized the concept of containers. Ever since, many software developers have adopted containers as part of their day-to-day workflow. These containers can help developers by running an application in an isolated and consistent environment. In this article, you will learn about containers and how you can use them when developing applications using MongoDB.

What Is a Docker Container?

A Docker container is a single unit containing an application and all of its necessary configurations or dependencies. Imagine a big zip file that would include everything needed to run your application on any operating system or hardware. Docker is a tool to run those containers.

The concept of containers dates back to the '70s. It finds its roots with the advent of the Unix operating system. The **chroot** command was introduced as part of the operating system, which changes the root directory of a process and its children. This was the beginning of process isolation.

Fast forward a few years, and you get the state of containers as they are today. Containers are a way to isolate everything needed to run an application in its environment. Docker popularized the concept of containers when they introduced the Docker Engine, which lets containers run on any operating system. Thanks to that engine, a container can run uniformly on Linux, macOS, and Windows.



Containers run as single processes on the Docker engine.

Containers run in their own isolated processes in a given user space. This has the benefit of significantly reducing the resources needed to run compared to virtual machines, which require a full operating system of their own. Therefore, the total cost of running an application as a container is much smaller than running it on a virtual machine.

Container technology is also at the heart of the microservices revolution in software development. Their small footprint makes it easy to run multiple self-contained and isolated applications. Containers also make it possible to have small microservices running together side by side. Orchestration tools such as Kubernetes also make it simple to upgrade unique microservices without any downtime.

Another benefit of using containers is that they are ephemeral in nature. When a container is restarted, it will always be the same, and any change that might have occurred while it was running is now gone. This is great to ensure that the application always runs consistently. It is also an excellent feature for software developers running tests on their system since any data entered would be deleted, and the test run would start fresh each time.

On the other hand, this can be somewhat challenging when preserving data is necessary, as would be the case when running a container for a database. If you need to preserve data, you will need to mount a volume which is accessible from the container. Tools such as Kubernetes will help you to manage these containerse, and the Kubernetes Operators can help you with setting up the persistent volumes.

Can MongoDB Run in a Docker Container?

MongoDB can run in a container. The official images available on Docker Hub include the community edition and enterprise edition of MongoDB and are maintained by

MongoDB. These images can be used in development environments. Using the kubernetes operator with the enterprise image is highly recommended for a production environment to be fully supported.

How to Use MongoDB With Docker

There are many ways to use MongoDB in a containerized environment. In the following sections, you'll learn some methods to run and manage containers with MongoDB.

Prerequisites

In order to run the following instructions, you will need to install Docker Desktop. You can find the installation instructions for your operating system on their website.

Running MongoDB as a Docker Container

You can start a MongoDB container using Docker with the following command. Start by setting the version you want to use in a variable.

```
export MONGODB_VERSION=6.0-ubi8
docker run --name mongodb -d mongodb/mongodb-community-server:$MONGODB_VERSION
```

This command will start a MongoDB server running version 6.0 based on the Red Hat UBI in detached mode (as a background process). As a best practice, it’s recommended to use a tag to specify the MongoDB version to ensure consistency. If you need to access the MongoDB server from another application running locally, you will need to expose a port using the -p argument.

```
docker run --name mongodb -d -p 27017:27017 mongodb/mongodb-community-server:$MONGODB_VERSION
```

Using this method, you will be able to connect to your MongoDB instance on **mongodb://localhost:27017**. You can try it with Compass, MongoDB's GUI to visualize and analyze your data.

Any data created as part of the lifecycle of that container will be destroyed once the container is deleted. You can do that with the Docker **stop** and **rm** commands.

```
docker stop mongodb && docker rm mongodb
```

If you want to persist the data on your local machine, you can mount a volume using the -v argument.

```
docker run --name mongodb -d -p 27017:27017 -v $(pwd)/data:/data/db
mongodb/mongodb-community-server:$MONGODB_VERSION
```

If you stop and start the container again, all the data you entered previously will still be there.

If your application is running inside a container itself, you can run MongoDB as part of the same Docker network as your application using **--network.** With this method, you will connect to MongoDB on **mongodb://mongodb:27017** from the other containerized applications in the network.

```
docker run --name mongodb -d --network mongodb mongodb/mongodb-community-server:$MONGODB_VERSION
```

You can test this with a second container running **mongosh** on the same network.

```
docker run --name mongosh --network mongodb mongodb/mongodb-community-server:$MONGODB_VERSION mongosh mongodb://mongodb --eval "show dbs"
```

To initialize your MongoDB with a root user, you can use the environment variables **MONGO_INITDB_ROOT_USERNAME** and **MONGO_INITDB_ROOT_PASSWORD.** These environment variables will create a user with root permissions with the specified user name and password.

```
docker run --name mongodb -d -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=user -e MONGO_INITDB_ROOT_PASSWORD=pass mongodb/mongodb-community-server:$MONGODB_VERSION
```

Keep in mind that you can combine these flags to suit your needs.

Connecting to MongoDB from Another Docker Container

Often, your application will run inside a container, and you will need to connect to a database running outside that container. The best way to do so is by using environment variables. By using an environment variable, you will set the connection string to a different value depending on where you are running this container.

For example, you might have the same container running both in your production environment and on your development servers. Your production application would most likely connect to MongoDB Atlas or a containerized MongoDB Enterprise server, while your development instance would connect to a local MongoDB instance. A connection string providing the server address will be passed as an environment variable in the **docker run** command.

```
docker run -d --name MYAPP -e
MONGODB_CONNSTRING=mongodb+srv://username:password@clusterURL MYAPP:1.0
```

Using environment variables, you will be able to connect to any MongoDB instance, whether cloud-based or locally, from your application that is running inside a container. You can find out more about doing this by following the full tutorial on MongoDB Developer Hub.

Using MongoDB with Docker Compose

If you have an application and a MongoDB container both running on the same machine, you can use Docker Compose to start and stop them together. Docker

Compose is better suited for development or testing environments where you don't need the full functionalities of MongoDB Enterprise or MongoDB Atlas.

In a **docker-compose.yaml** file, describe all of your containers that are part of the application. One of your containers could be a MongoDB server. As a best practice, you would pass your connection string to your application as an environment variable, as explained in the previous section.

```
version: '3'
services:
  myapplication:
    image: mongodb/mongodb-community-server:6.0-ubi8
    environment:
      - CONN_STR=mongodb://user:pass@mongodb
    command: '/bin/bash -c "sleep 5; mongosh $$CONN_STR --eval \"show
dbs;\""'
    depends_on:
      - mongodb
  mongodb:
    image: mongodb/mongodb-community-server:6.0-ubi8
    environment:
      - MONGO_INITDB_ROOT_USERNAME=user
```

From the directory in which the file resides, run the **docker-compose** command.

```
docker-compose up
```

This command will start both your application and your local MongoDB instance.

Managing MongoDB from a Container

To manage your MongoDB server or to access, import, and export your data, you can use a second MongoDB container from which you will run the necessary CLI tools.

To open up a Mongo Shell session to your MongoDB Atlas server, use mongosh and specify the cluster URL.

```
docker run -it mongodb/mongodb-community-server:$MONGODB_VERSION mongosh
"mongodb://username:password@clusterURL/database"
```

If you want to use the **mongoexport** tool to export an existing collection to a **.json** file, you can use the command from a separate MongoDB container. You will need to mount a volume to be able to access the resulting JSON file.

```
docker run -it -v $(pwd):/tmp mongodb/mongodb-community-
server:$MONGODB_VERSION mongoexport --collection=COLLECTION --
out=/tmp/COLLECTION.json "mongodb://username:password@clusterURL/database"
```

If you need to import data into a collection, you use the **mongoimport** tool, also available from the **mongodb/mongodb-community-server:$MONGODB_VERSION** image. Again, you will need to mount a volume to access a file stored on your local machine from inside the container.

```
docker run -it -v $(pwd):/tmp mongodb/mongodb-community-
server:$MONGODB_VERSION mongoimport --drop --collection=COLLECTION
"mongodb+srv://user:password@clusterURL/database" /tmp/COLLECTION.json
```

Any other tool that is packaged with a MongoDB installation can be accessed in the same fashion.

What Are the Benefits of Using Docker?

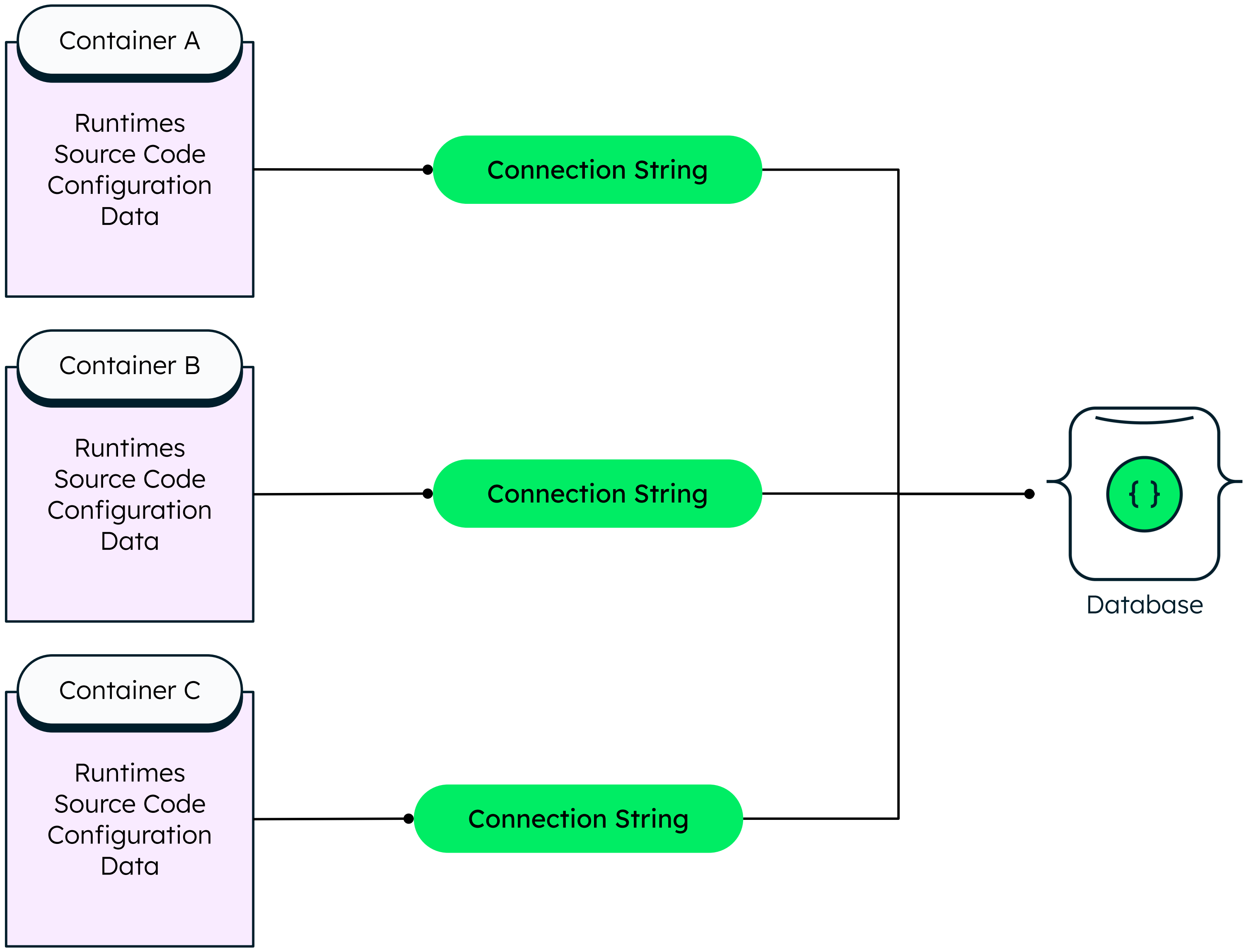
There are many advantages of using containers as part of your day-to-day life as a software developer.

- **Consistency.** By using container technology, you can ensure that everyone on your team uses the exact same runtimes and configurations. You also significantly reduce deployment friction as the production environment will be consistent with your development environment.
- **Lightweight.** Docker containers are quick to start and consume minimal resources compared to virtual machines.
- **Ephemeral.** Any change in the container file system will be destroyed upon termination. This impermanence ensures a fresh environment at every start.

Docker and MongoDB Atlas

The best way to use Docker with MongoDB is to have a containerized application that connects to MongoDB Atlas, the Database-as-a-Service offering by MongoDB. A free tier is available, which is well suited for a development environment. Using this free tier, you can easily share a database with your team to use a shared test data set, or even have a distinct test database for each team member.

The rest of your application can run in containers to ensure consistency across all environments. The application would connect to MongoDB using a connection string that is passed in an environment variable.



Containers connecting to Atlas through a connection string environment variable.

With MongoDB Atlas, it's easy to connect all the various microservices that compose your system to the same shared database. MongoDB Atlas also provides you with all the benefits of running a MongoDB cluster without the hassle of setting up, configuring, and managing a replica set.

Next Steps

In this article, you've learned that containers are isolated processes that will run your application in a consistent way across all operating systems or infrastructures. You've also learned how to start a container to run MongoDB and how to connect to MongoDB from a containerized application.

MongoDB Atlas is easy to use with your containerized application and makes it easy for you and your team to access your data. You can find out more about MongoDB and containers with the following resources.

WHITE PAPER

Enabling Microservices: Containers & Orchestration Explained

Learn more

BLOG

Running MongoDB as a Microservice with Docker and Kubernetes

Learn more

DOCUMENTATION

Running MongoDB as a Microservice with Docker and Kubernetes

Learn more

WEBINAR

Enabling Microservices with Containers, Orchestration, and MongoDB

Learn more

BLOG

Leaf in the Wild: Leading Soccer Streaming Service fuboTV Scales its Business with MongoDB, Docker Containers and Kubernetes

Learn more

FAQ

Should I Run MongoDB In Docker? +

How to Use MongoDB from a Docker Container

How Do I Connect to a MongoDB Docker Container?