Docker is a popular platform for containerization of applications. A container is a lightweight and portable executable package that contains everything needed to run an application, including code, runtime, system tools, libraries, and settings.

To create a data container with Docker, you can use the following steps:

Create a new container: Use the docker create command to create a new container, which will serve as the data container. You can specify the image you want to use, as well as any additional settings or volumes you need.
docker create --name my-data-container -v /path/to/data:/data some/image

This command creates a new container named my-data-container based on the some/image image and creates a volume at /data inside the container, which maps to the local directory /path/to/data on the host machine.

Populate the data container: You can copy files or data into the data container using the docker cp command:

docker cp /local/path/to/data my-data-container:/data

This command copies the files from /local/path/to/data on the host machine to the /data volume inside the my-data-container container.

Use the data container: You can use the data container in other containers by specifying the --volumes-from option:

docker run --volumes-from my-data-container some/other-image

This command runs a new container based on the some/other-image image and mounts the /data volume from the my-data-container container.

By using a data container in this way, you can ensure that your application data is separate from the application itself and can be easily shared and reused across different containers.

A data container in Docker is a container that is specifically created to store and manage persistent data that needs to be shared across different containers. The use of data containers provides a way to decouple application code from application data, making it easier to manage and share data among different containers.

The key benefits of using a data container are:

Data sharing: With a data container, multiple containers can share the same data volume, which makes it easier to manage data across different containers.

Data persistence: A data container ensures that data remains persistent even if the application container is stopped, started, or deleted. This is particularly useful for storing configuration files, logs, and other important data that needs to be preserved even if the container is removed.

Data isolation: By separating the data from the application container, a data container can provide an additional layer of isolation, making it easier to manage data access and permissions.

Easy backup and restore: Since the data is stored in a separate container, it is easier to backup and restore data without having to backup or restore the entire application container.

Overall, data containers provide a simple and efficient way to manage and share data among different containers, making it easier to build and deploy complex applications in a microservices architecture.

There are two ways of approaching stateful Containers, that is containers are store and persistent data for future use. This could be the container creating and storing data, for example, a database. Alternatively, it

could be data requiring additional for instance the configuration or SSL certifications. This approach can also be used to backup data or debug containers.

One approach we've discussed is using the -v <host-dir>:<container-dir> option to map directories. The other approach is to use Data Containers. This scenario will introduce the advantages of using Data Containers.

Step 1 - Create Container
Data Containers are containers whose sole responsibility is to be a place to store/manage data.

Like other containers they are managed by the host system. However, they don't run when you perform a docker ps command.

To create a Data Container we first create a container with a well-known name for future reference. We use busybox as the base as it's small and lightweight in case we want to explore and move the container to another host.

When creating the container, we also provide a -v option to define where other containers will be reading/saving data.

Task
Create a Data Container for storing configuration files using

```
$ docker create -v /config --name dataContainer busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
4b35f584bb4f: Pull complete
Digest: sha256:b5d6fe0712636ceb7430189de28819e195e8966372edfc2d9409d79402a0dc16
Status: Downloaded newer image for busybox:latest
de8ab1e20638be1032b7949435ee471c119e73aa53ad4b701b5558e6eb5c3359
```

Above command , creates a new Docker container named dataContainer based on the busybox image, and creates a new volume at /config inside the container.

The --name option is used to specify the name of the container, which in this case is dataContainer. The -v option is used to create a new volume at the specified path, in this case /config.

Once created, the dataContainer container can be used as a data volume for other containers by specifying the --volumes-from option when starting a new container.

Step 2 - Copy Files
With the container in place, we can now copy files from our local client directory into the container.

To copy files into a container you use the command docker cp. The following command will copy the config.conf file into our dataContainer and the directory config.

```
$ docker cp config.conf dataContainer:/config/
$ docker cp config.conf dataContainer:/config/
$
```

The command docker cp config.conf dataContainer:/config/ copies the config.conf file from the host machine to the /config directory inside the dataContainer container.

The docker cp command is used to copy files and directories between a container and the host machine or between two containers. In this case, the source file is config.conf located on the host machine and the destination directory is /config inside the dataContainer container.

After running this command, the config.conf file will be available in the /config directory inside the dataContainer container, and can be accessed by other containers that use this container's volumes.

Step 3 - Mount Volumes From

Now our Data Container has our config, we can reference the container when we launch dependent containers requiring the configuration file.

Using the --volumes-from <container> option we can use the mount volumes from other containers inside the container being launched. In this case, we'll launch an Ubuntu container which has reference to our Data Container. When we list the config directory, it will show the files from the attached container.

```
$ docker run --volumes-from dataContainer ubuntu ls /config
config.conf
```

The command docker run --volumes-from dataContainer ubuntu ls /config runs a new Ubuntu container and mounts the volumes from the dataContainer container using the --volumes-from option. It then lists the contents of the /config directory inside the dataContainer container.

This command is useful for checking that the dataContainer container has the correct data volume mounted and that the files and directories are accessible from other containers that use this container's volumes.

Step 4 - Export / Import Containers

If we wanted to move the Data Container to another machine then we can export it to a .tar file.

```
$ docker export dataContainer > dataContainer.tar
$
```

The command docker import dataContainer.tar will import the Data Container back into Docker.

```
$ docker import dataContainer.tar
sha256:77ed91d04748108335328367df12aba35cdb2c0174cd84696b56c1767d7319d2
```

This scenario has explained how to use Data Containers. Data Containers are a great way of managing configuration data which can be used by other containers. The volumes-from property allows us to gain access to other containers volumes which we can utilise for debugging or backup purposes.