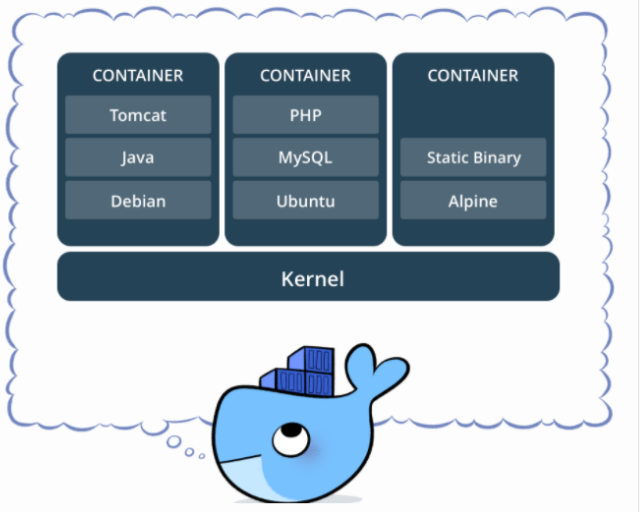
Running multi-containers

Why multi-containers?

In particular, we are going to see how we can run and manage *multi-container docker environments*.

Why multi-container you might ask?> Well, one of the key points of Docker is the way it provides isolation. The idea of bundling a process with its dependencies in a sandbox (called containers) is what makes this so powerful.



Multi Container

- Just like it's a good strategy to decouple your application tiers, it is wise to keep containers for each of the **services** separate. Each tier is likely to have different **resource needs** and those needs might grow at different rates.
- By separating the tiers into different containers, we can **compose each tier** using the most appropriate instance type based on different resource needs. This also plays in very well with the whole `microservices` (software applications as suites of independently deployable services) movement which is one of the main reasons why Docker (or any other container technology) is at the `forefront of modern microservices architectures`.

Docker Compose

> Docker Compose is a tool for defining and running multi-container Docker applications.

Compose is a tool that is used for defining and running multi-container Docker apps in an easy way. It provides a configuration file called `docker-compose.yml` that can be used to bring up an application and the **suite of services** it depends on with just one command. Compose works in all environments: production, staging, development, testing, as well as CI workflows, although Compose is ideal for development and testing environments.

Using Compose is basically a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in **docker-compose.yml** so they can be run together in an isolated environment.
3. Run `docker-compose up` and Compose starts and runs your entire app.

Common Use Cases

Compose can be used in many different ways. Some common use cases are outlined below.

- **Development environments** When you're developing software, the ability to run an application in an *isolated environment* and interact with it is crucial. The Compose command line tool can be used to create the environment and interact with it.

The `Compose file` provides a way to document and configure all of the application's service dependencies (databases, queues, caches, web service APIs, etc). Using the Compose command line tool you can create and start one or more containers for each dependency with a single command (`docker-compose up`).

- **Automated testing environments**

An important part of any Continuous Deployment or Continuous Integration process is the **automated test suite**. Automated end-to-end testing requires an environment in which to run tests. Compose provides a convenient way to create and destroy isolated testing environments for your test suite. By defining the full environment in a Compose file, you can create and destroy these environments in just a few commands:

```
$ docker-compose up -d
$ ./run_tests
$ docker-compose down
```

- **Single Host Deployments**

Compose has traditionally been focused on *development* and *testing* workflows. You can use Compose to deploy to a remote Docker Engine. The Docker Engine may be a single instance provisioned with **Docker Machine**

> Q: What are the docker compose features?
> A: The features of Compose that make it effective are:
>
> - Multiple isolated environments on a single host
> - Preserve volume data when containers are created
> - Only recreate containers that have changed
> - Variables and moving a composition between environments
>
> — Interview Q&A

Docker Compose File

Installation

If you're running Windows or Mac, Docker Compose is already installed as it comes in the Docker Toolbox. Linux users can easily get their hands on Docker Compose by pasting this lines to your terminal: (First line downloads the current stable release of Docker Compose and the second line applies executable permissions to the binary)

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.25.4/docker
    -compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

> **Note:** If the command docker-compose fails after installation, check your path. You can also create a symbolic link to **/usr/bin** or any other directory in your path.

Since Compose is written in Python, you can also simply do `pip install docker-compose`. Test your installation with

```
$ docker-compose --version
docker-compose version 1.25.4, build 1110ad01
```

Docker Compose File (docker-compose.yml)

As you understand from .yml extension its YAML file. The syntax for YAML is quite simple. As an example, we will look one of the docker-compose.yml:

```
version: "3"
services:
  es:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.3.2
    container_name: es
    environment:
      - discovery.type=single-node
    ports:
      - 9200:9200
    volumes:
      - esdata1:/usr/share/elasticsearch/data
  web:
    image: prakhar1989/foodtrucks-web
    command: python app.py
    depends_on:
      - es
    ports:
      - 5000:5000
    volumes:
      - ./flask-app:/opt/flask-app
volumes:
    esdata1:
      driver: local
```

Let's breakdown what the file above means. At the parent level, the names of services are defined - `es` and `web`. For each service that Docker needs to run, we can add additional parameters out of which image is required.

Via other parameters such as `command` and `ports` we can reach more information about the container. The `volumes` parameter specifies a mount point in web container where the code will reside. This is purely optional and is useful if you need access to logs etc. There is also `volumes` for the es container so that the data we load persists between restarts. Also `depends_on` is specified , which tells docker to start the es container before web.

> **Note:** You must be inside the directory with the docker-compose.yml file in order to execute most Compose commands.