What is Docker? Why You Should Use it?

A One Stop Guide to Docker by Ritesh Yadav

Agenda

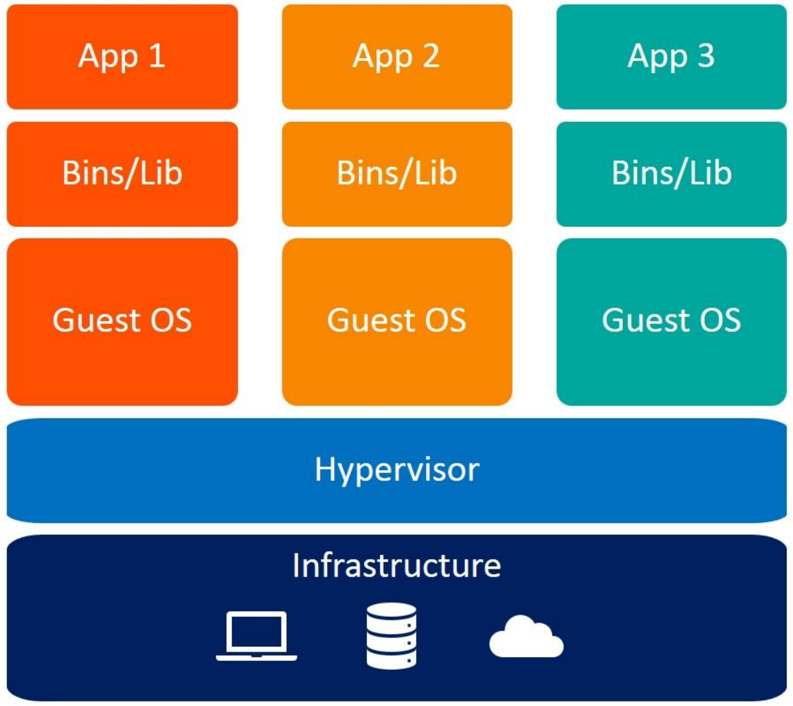
What is Docker?

Docker is a software platform that simplifies the process of building, running, managing and distributing applications. It does this by virtualizing the operating system of the computer on which it is installed and running.

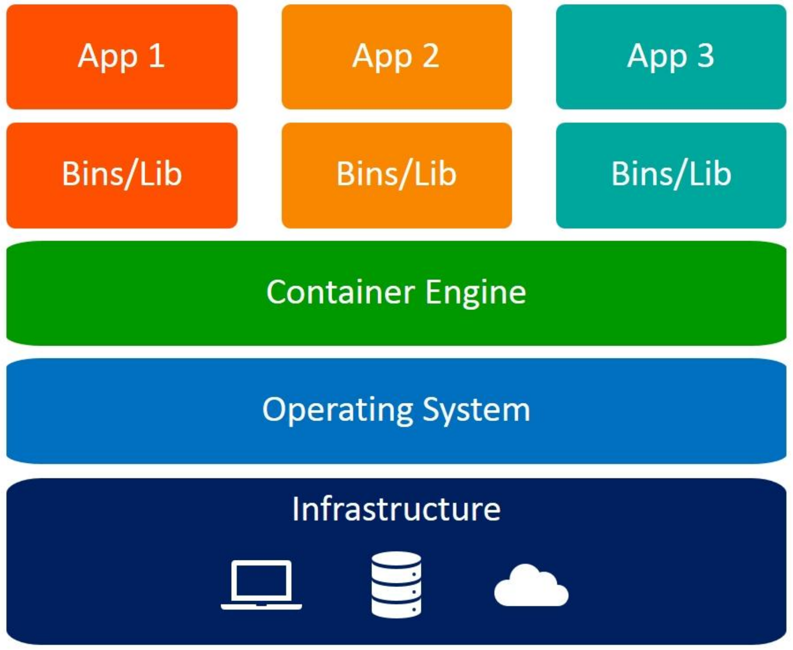
*Docker is a tool designed to make it easier to create, deploy, and run applications by using **containers**.*

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Virtual Machine Vs Docker



Virtual Machines



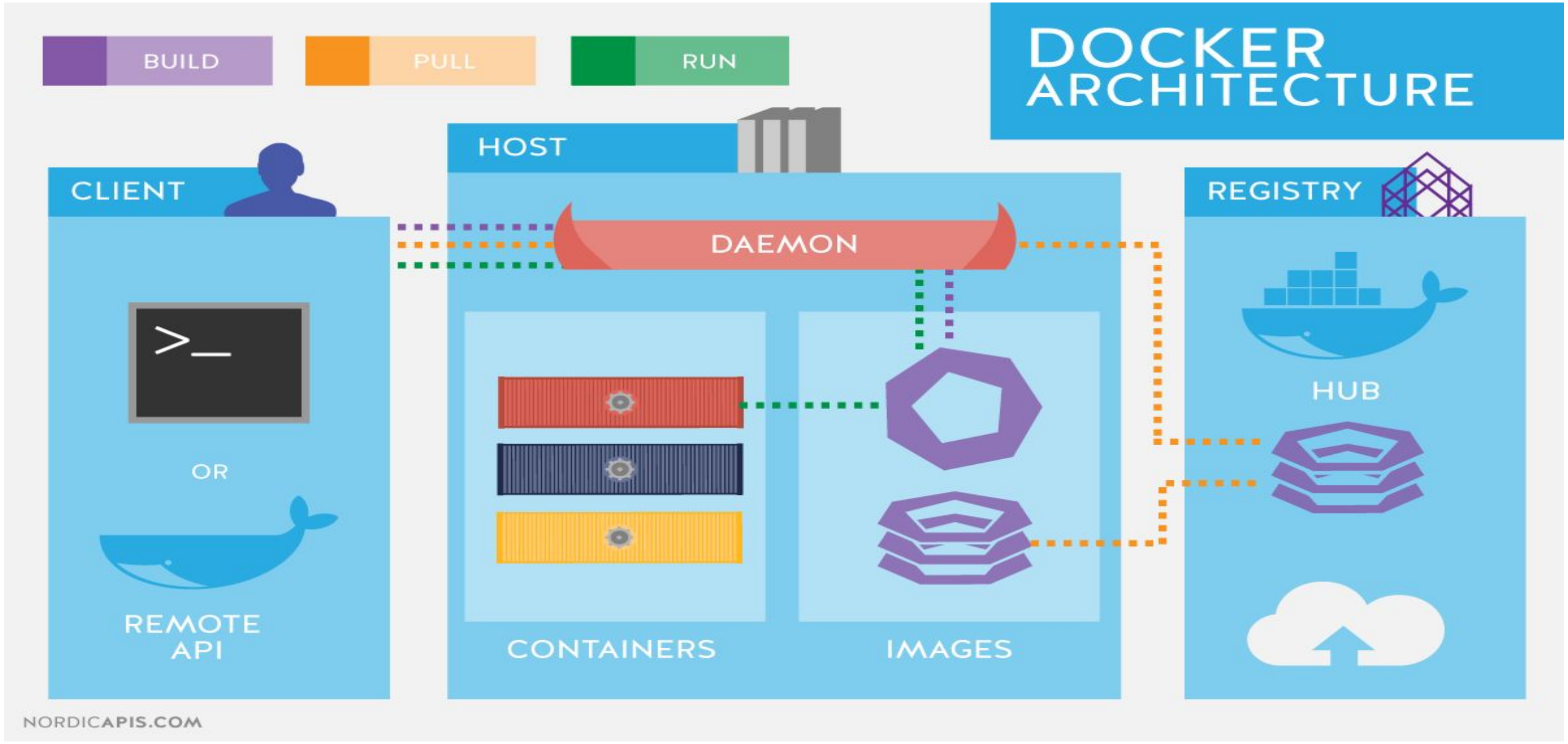
Containers

Docker is container based technology and containers are just user space of the operating system. At the low level, a container is just a set of processes that are isolated from the rest of the system, running from a distinct image that provides all files necessary to support the processes. It is built for running applications. In Docker, the containers running share the host OS kernel.

A **Virtual Machine**, on the other hand, is not based on container technology. They are made up of user space plus kernel space of an operating system. Under VMs, server hardware is virtualized. Each VM has Operating system (OS) & apps. It shares hardware resource from the host.

VMs & Docker – each comes with benefits and demerits. Under a VM environment, each workload needs a complete OS. But with a container environment, multiple workloads can run with 1 OS. The bigger the OS footprint, the more environment benefits from containers. With this, it brings further benefits like Reduced IT management resources, reduced size of snapshots, quicker spinning up apps, reduced & simplified security updates, less code to transfer, migrate and upload workloads.

Docker Architecture



Reasons to use Docker for the Development Environment

1. Runs on my machine = runs anywhere

If you have correctly dockerized your app and it runs without problems on your machine, 99% of the times it will run smoothly anywhere. By anywhere it means on your friend's machine, on the staging environment and production too. Given all of them have docker installed and configured correctly your app will run. Using docker also makes the application code cloud provider agnostic. Your application can potentially run on AWS or GCP or Azure without issues.

2. New team member can be productive from day 1

Think of this, a new team member joins then s/he spends more than a day to set up the machine with the right OS. Setup the language(s) used in the company add database(s) on top of it. 2-3 days is wasted on just getting the environment setup correctly. Enter docker + docker-compose, the new joiner sets up the OS. Installs docker then runs 3-5 commands, grabs some coffee and magic: your apps(s) are running. The new joiner can contribute with the working code on day 1. Think of all the cost a company can save with this approach. A streamlined docker implementation makes it a reality.

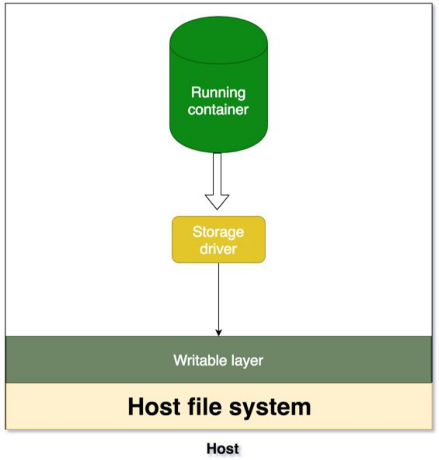
How To Install Docker?

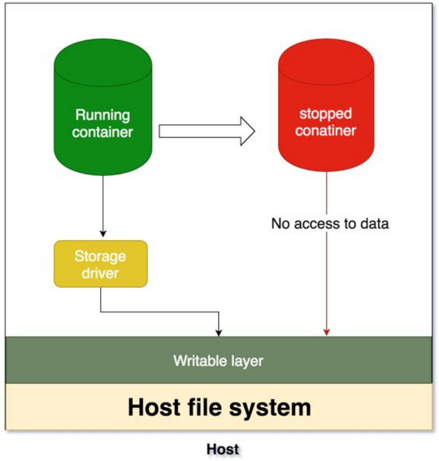
https://docs.docker.com/get-docker/

Docker Volumes!

OK, so now let say you want to store some content in your docker container, so let see how you can can do that as there are two approaches one which is deprecated and one which is working like charm.

1st approach is to use the local storage drivers, to store the data, but it has a lot of problems as whenever the docker container is deleted or stopped the file storage is deleted.Also let say another container wants to use the same storage as a distributed storage, it is very difficult for another container to use it.





- Data is no longer persisted and difficult to access if container stops as shown in the following diagram
- As we can see writable layer is tightly coupled with host filesystem and difficult to move the data.
- We have an extra layer of abstraction with a storage driver which reduces the performance.

A Demo!

// pull the nginx image
docker pull nginx
// run the container
docker run -it --name=webApp -d -p 80:80 nginx

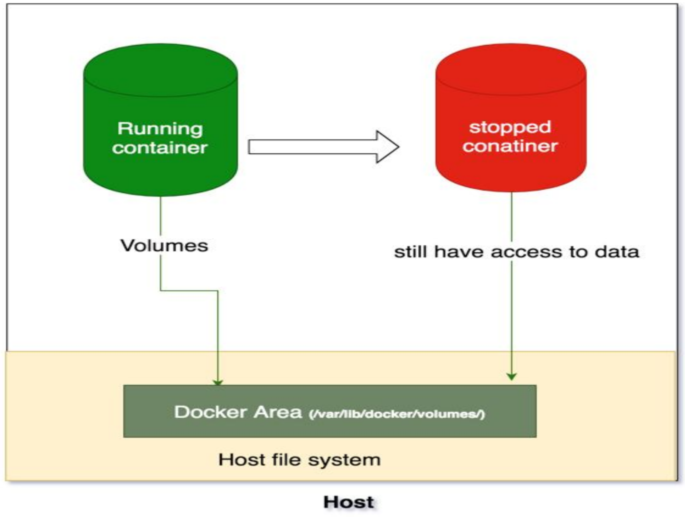
// list the running containers
docker ps
// exec command
docker exec -it webApp bash
// cd to welcome page and edit it
cd /usr/share/nginx/html
echo "I changed this file while running the container" > index.html

Now just stop and run the another container and try to use the same storage for it. What Happen? You can't use that storage as a shared one.

To resolve this problem we have Docker Volumes a second approach.

How Volumes can solve above issues

Volumes are saved in the host filesystem (**/var/lib/docker/volumes/**) which is owned and maintained by docker. Any other non docker process can't access it. But, As depicted in the below other docker processes/containers can still access the data even container is stopped since it is isolated from the container file system.


Running container
stopped conatiner
Volumes
still have access to data
Docker Area (/var/lib/docker/volumes/)
Host file system
Host

How to create a Volume

We can create a volume with the below command or while container/service creation and it is created in the directory of the docker host and When you mount the volume into a container, this directory is what is mounted into the container. we should notice the difference between creation and mounting.

```
$ docker volume create <volumeName>
```

How to remove a Volume

```
$ docker volume prune
```

```
//create a volume
docker volume create new_vol
//list volumes
docker volume ls
//inspect volumes
docker volume inspect new_vol
//removing volumes
docker volume rm new_vol
```

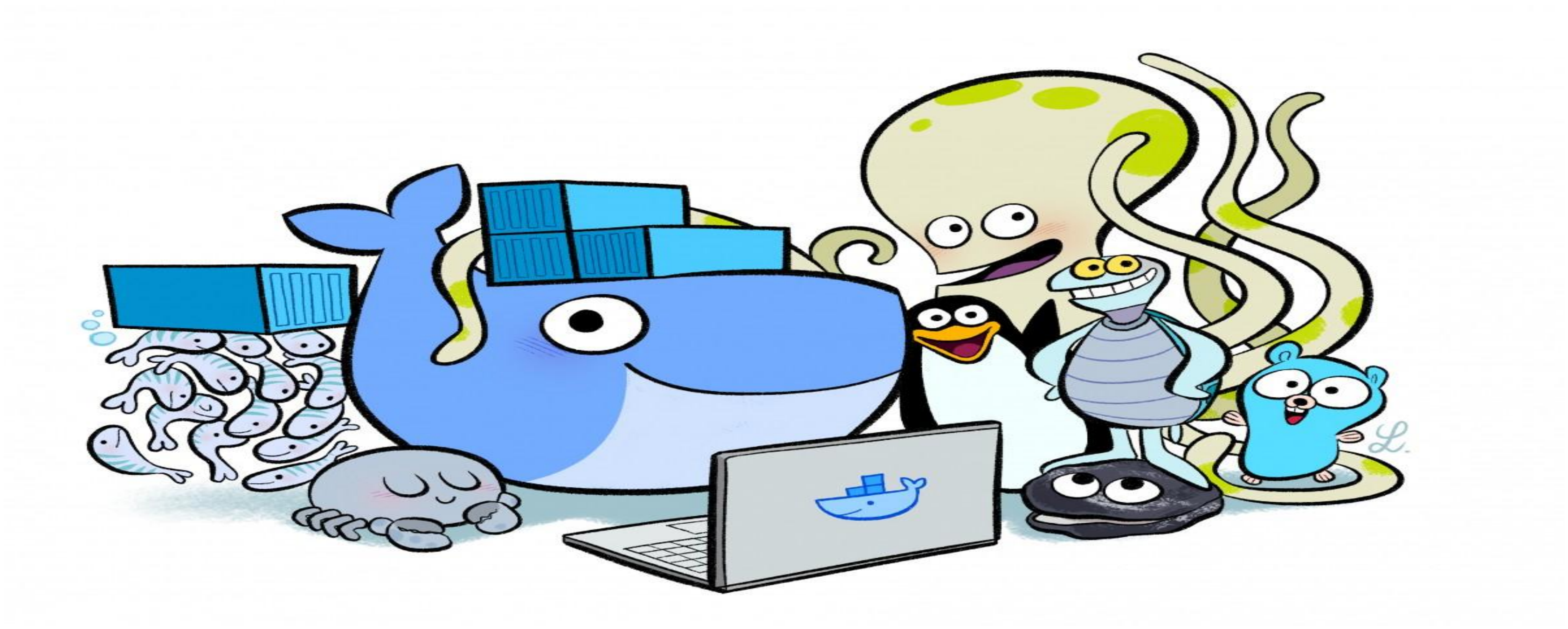
Mount Volumes to Containers

```
docker run -d --name=webApp1 --mount source=new_vol,destination=/usr/share/nginx/html -p 80:80 nginx

cd /var/lib/docker/volumes/

cd new_vol/
```

Docker Networking

https://www.aquasec.com/cloud-native-academy/docker-container/docker-networking/

Docker Commands, Networking, Volumes!



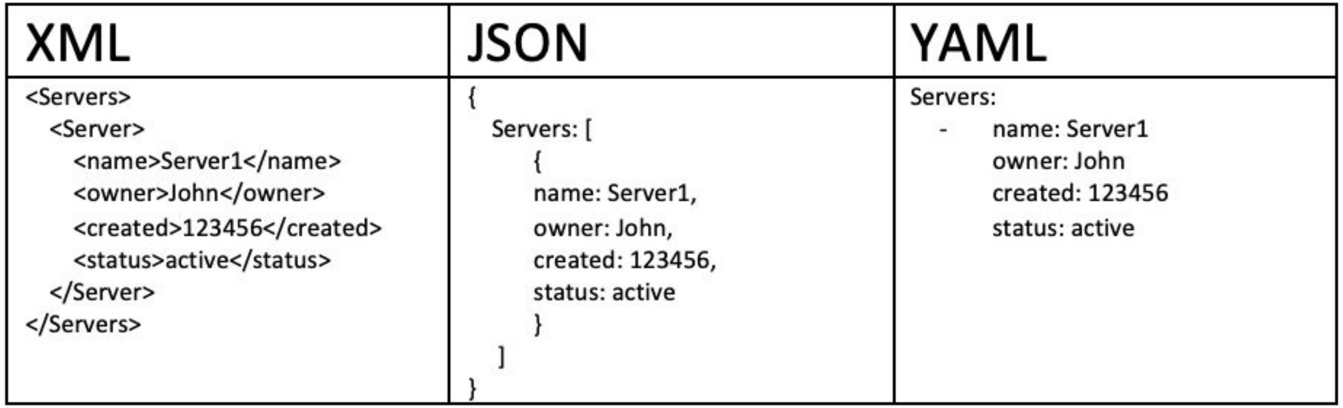
Checkout Day 4 , Day 5, Day 6 of this Live Community Session

Cheat Sheet 1: https://dockerlabs.collabnix.com/docker/cheatsheet/
Cheat Sheet 2: https://hackmd.io/@WjOdIG6eQkKK8RhnwE2Dyw/BJw_Rn2mF

YAML Basics!

YAML is a data serialization language that allows you to store complex data in a compact and readable format. It’s important for DevOps and virtualization because it’s essential in making efficient data management systems and automation.

The YAML acronym was shorthand for Yet Another Markup Language.



XML

```
<Servers>
   <Server>
      <name>Server1</name>
      <owner>John</owner>
      <created>123456</created>
      <status>active</status>
   </Server>
</Servers>
```

JSON

```
{
   Servers: [
        {
        name: Server1,
        owner: John,
        created: 123456,
        status: active
        }
    ]
}
```

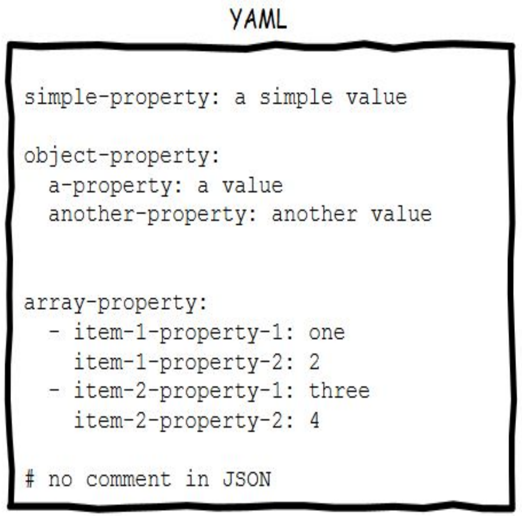
YAML

```
Servers:
   -     name: Server1
         owner: John
         created: 123456
         status: active
```

@IBM Developers

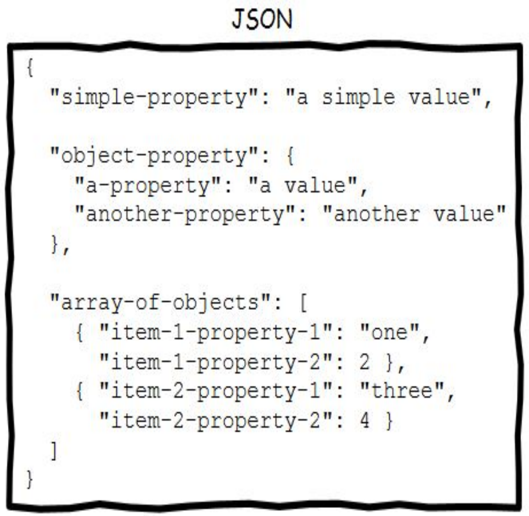
It is similar to XML and JSON files but uses a more **minimalist syntax** even while maintaining similar capabilities. YAML is commonly used to create configuration files in Infrastructure as Code (IoC) programs or to manage containers in the DevOps development pipeline.

More recently, YAML has been used to create automation protocols that can execute a series of commands listed in a YAML file. This means your systems can be more independent and responsive without additional developer attention.



YAML

```yaml
simple-property: a simple value

object-property:
  a-property: a value
  another-property: another value


array-property:
  - item-1-property-1: one
    item-1-property-2: 2
  - item-2-property-1: three
    item-2-property-2: 4

# no comment in JSON
```

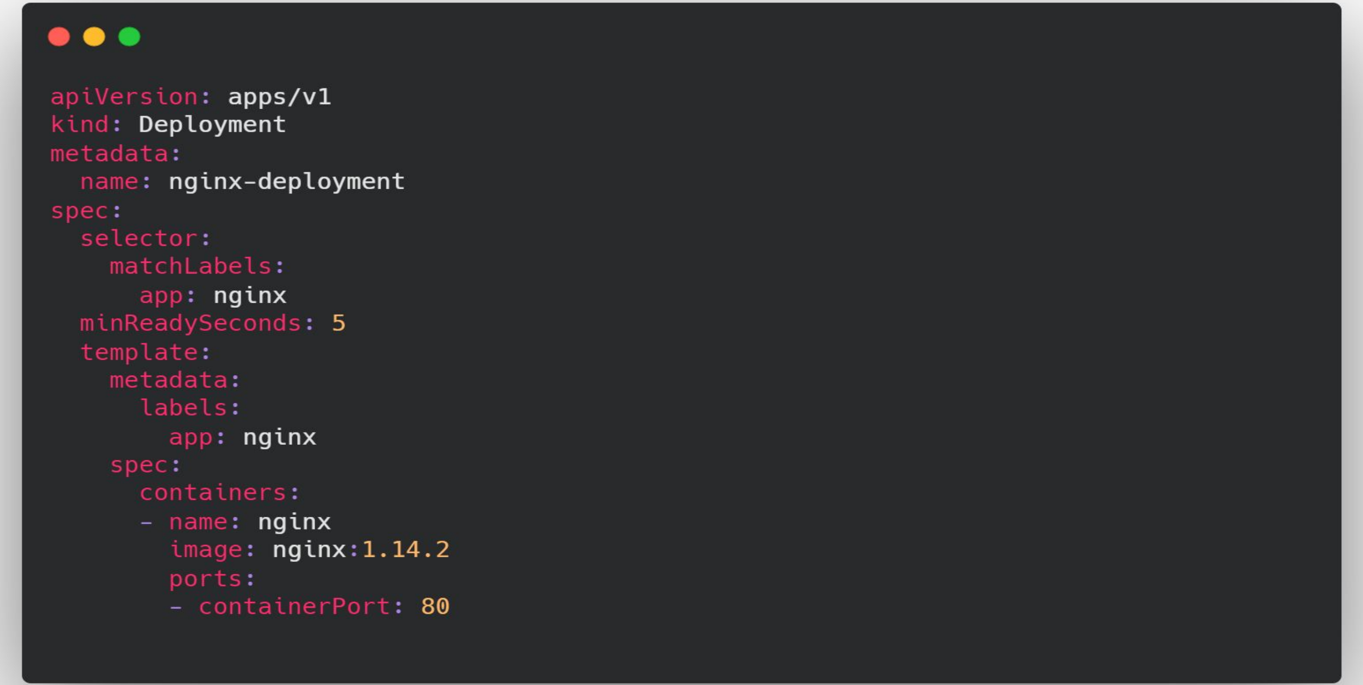


JSON

```json
{
  "simple-property": "a simple value",

  "object-property": {
    "a-property": "a value",
    "another-property": "another value"
  },

  "array-of-objects": [
    { "item-1-property-1": "one",
      "item-1-property-2": 2 },
    { "item-2-property-1": "three",
      "item-2-property-2": 4 }
  ]
}
```

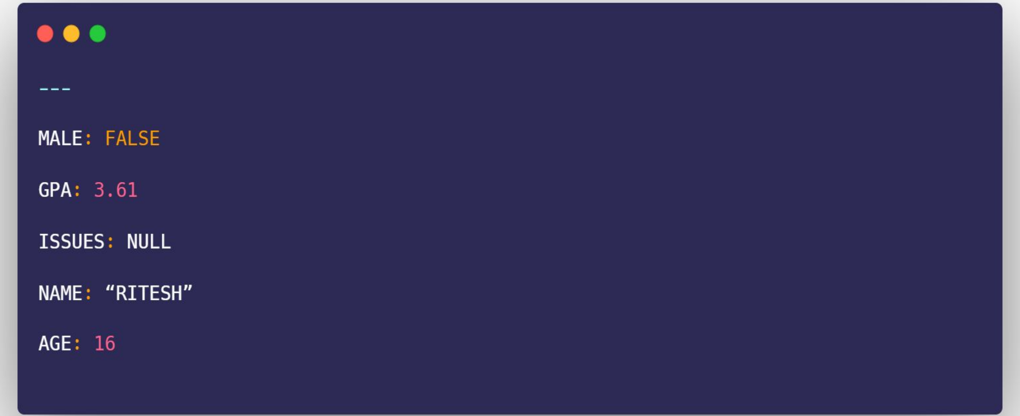
YAML Syntax!

Warning: Just Read! [Sample YAML]



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  minReadySeconds: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Datatype and K&V in YAML



```
---
MALE: FALSE
GPA: 3.61
ISSUES: NULL
NAME: “RITESH”
AGE: 16
```

Numeric Data type

There are three types of numeric data type:

- Integer
- Floating point numbers
- Booleans.

String Data type

String Datatype is a common thing in every programming lang and it is pretty much same here also, eg: **str : “Hello,world”**

YAML has three types of data types:

1. Scalar
2. List
3. Dictionary

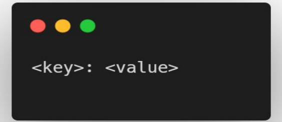
Scalar data type

Scalar is a simple data type. In YAML

scalar means a simple value for a key. The value of the scalar can be integer, float, Boolean, and string. Scalar data types are classified into two data types:

a. Numeric Data type
b. String

Key & Value Pair



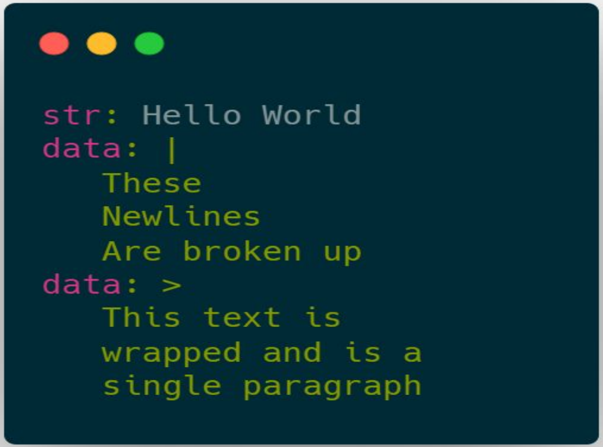
```
<key>: <value>
```

YAML file are a form of key-value pair where the key represents the pair’s name and the value represents the data linked to that name.

Strings in YAML

Strings are a collection of characters that represent a sentence or phrase. You either use `|` to print each string as a new line or `>` to print it as a paragraph.

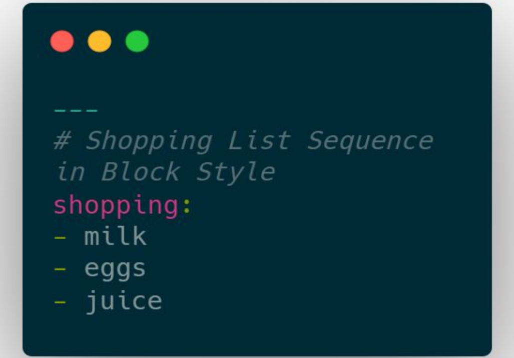
Strings in YAML do not need to be in double-quotes.



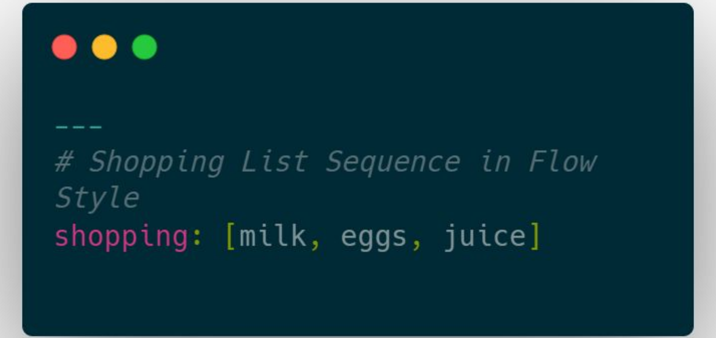
```yaml
str: Hello World
data: |
   These
   Newlines
   Are broken up
data: >
   This text is
   wrapped and is a
   single paragraph
```

Sequences in YAML

Sequences are data structures similar to a list or array that hold multiple values under the same key. They're defined using a block or inline flow style.



```yaml
---
# Shopping List Sequence
in Block Style
shopping:
- milk
- eggs
- juice
```



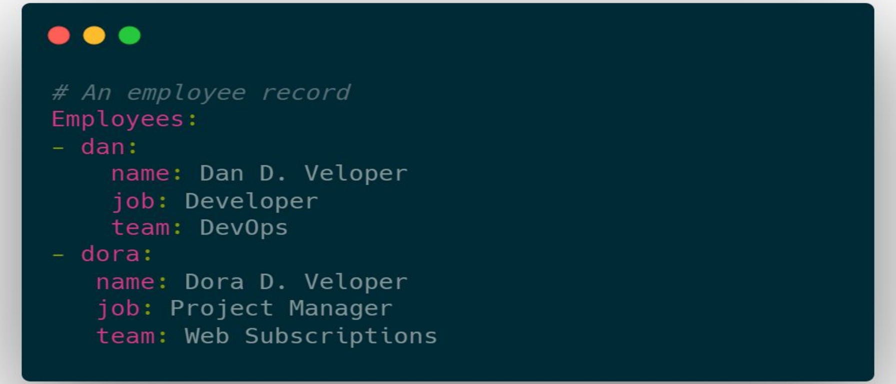
```yaml
---
# Shopping List Sequence in Flow
Style
shopping: [milk, eggs, juice]
```

@educative

Dictionaries in YAML

Dictionaries are collections of key-value pairs all nested under the same subgroup. They're helpful to divide data into logical categories for later use.

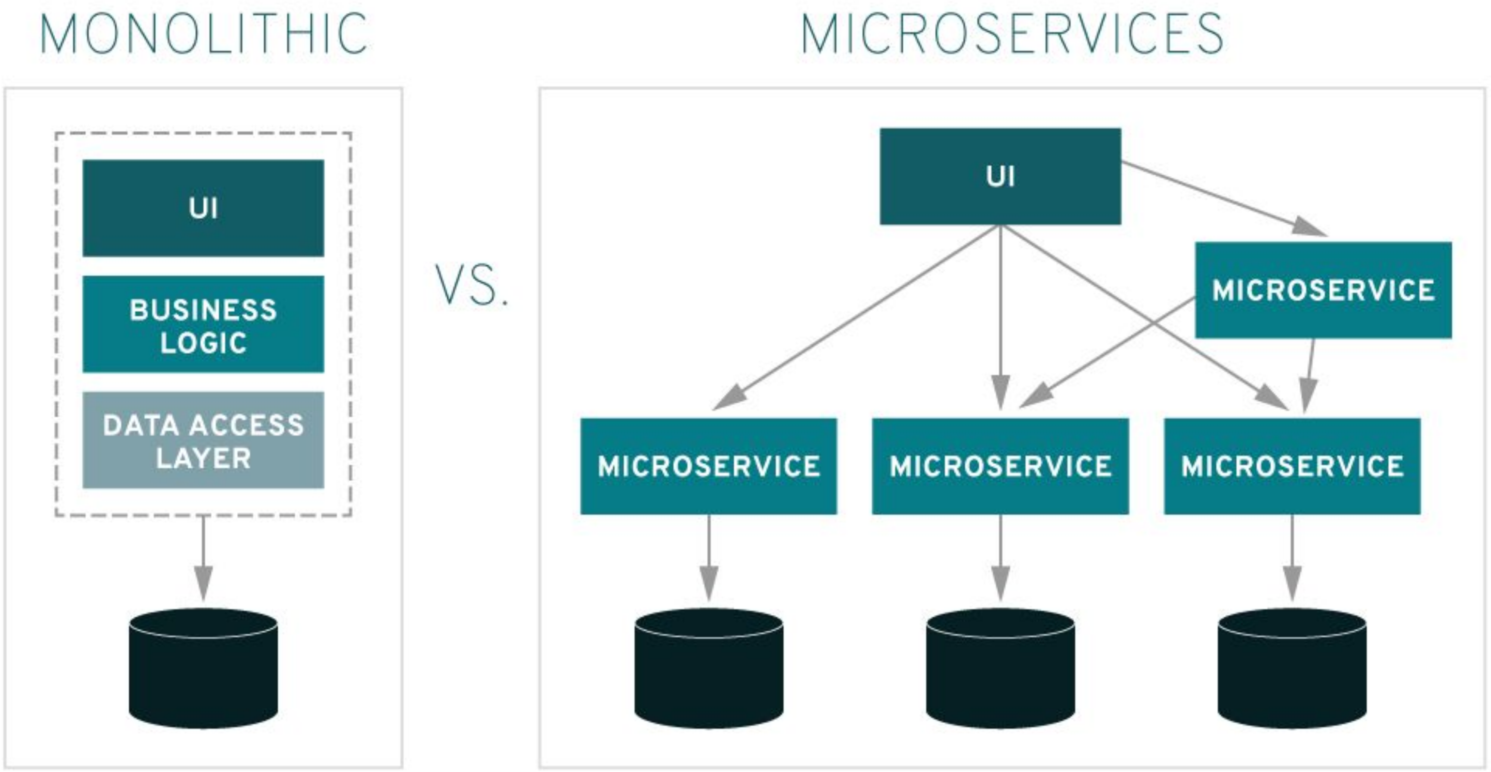
Dictionaries are defined like mappings in that you enter the dictionary name, a colon, and a space followed by 1 or more indented key-value pairs.



```yaml
# An employee record
Employees:
- dan:
    name: Dan D. Veloper
    job: Developer
    team: DevOps
- dora:
   name: Dora D. Veloper
   job: Project Manager
   team: Web Subscriptions
```

@educative

Microservices and Monolithic Arch



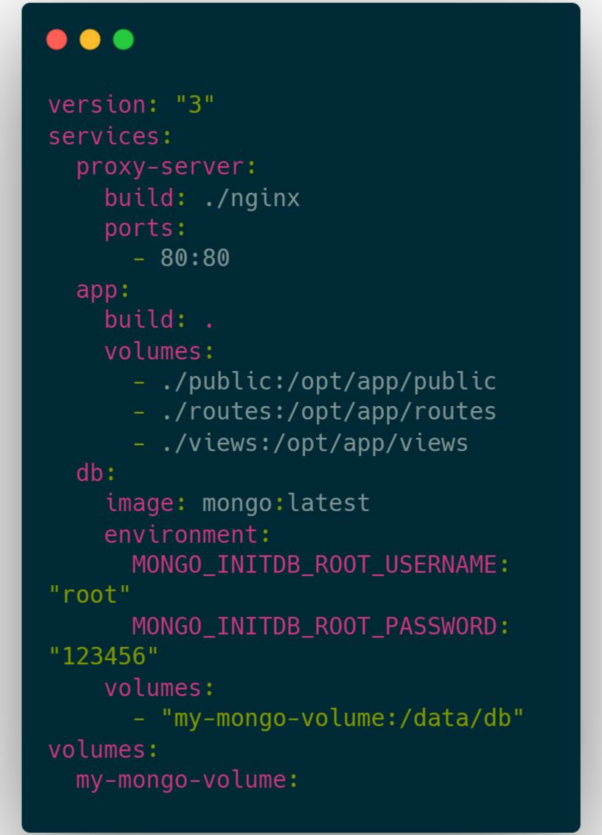
Docker Compose

Docker Compose is used for running multiple containers as a single service. Each of the containers here run in isolation but can interact with each other when required. Docker Compose files are very easy to write in a scripting language called YAML.

Docker Compose works by applying many rules declared within **a single *docker-compose.yml* configuration file**.

Benefits of Docker Compose

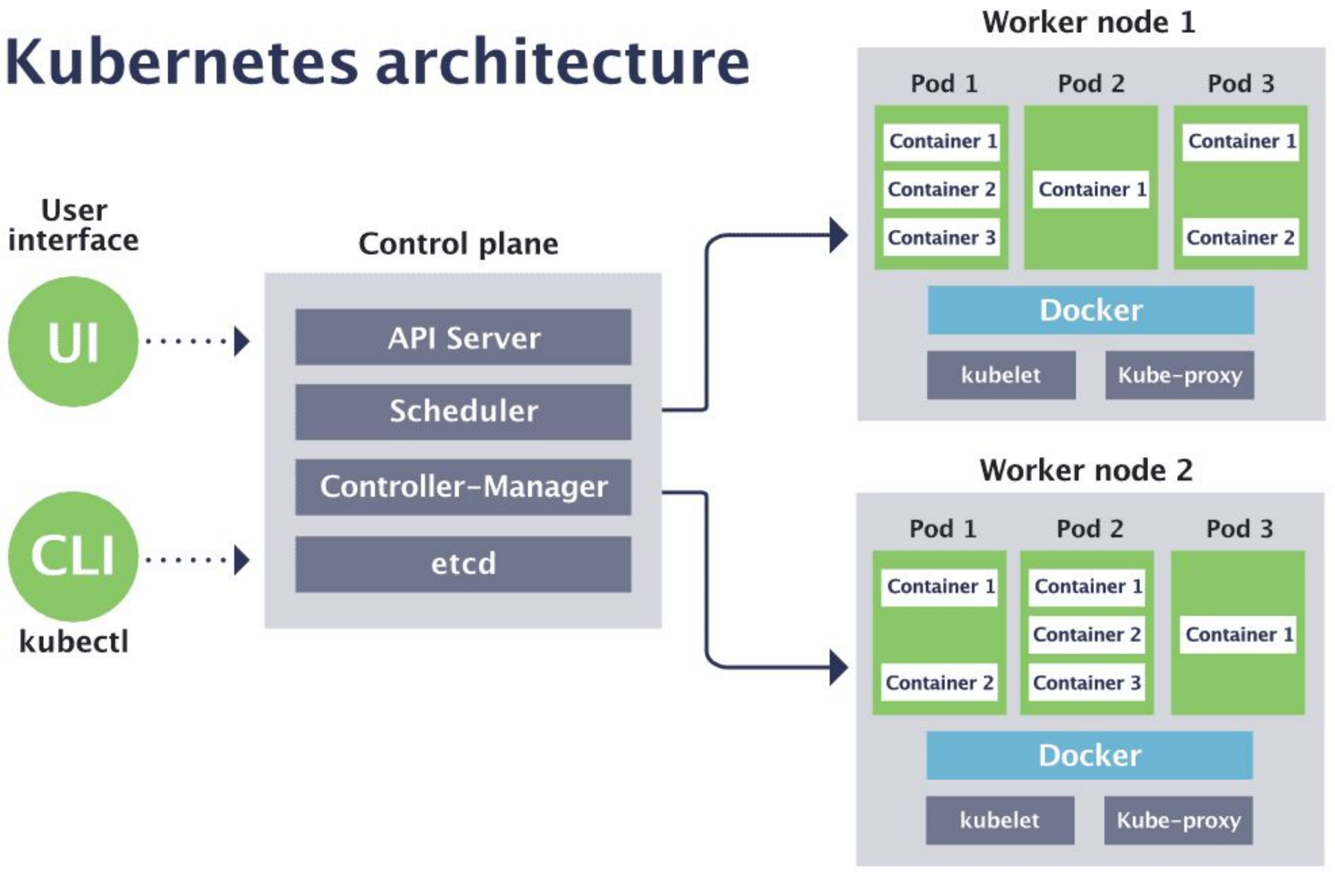
- **Single host deployment** - This means you can run everything on a single piece of hardware
- **Quick and easy configuration** - Due to YAML scripts
- **High productivity** - Docker Compose reduces the time it takes to perform tasks
- **Security** - All the containers are isolated from each other, reducing the threat landscape



```yaml
version: "3"
services:
  proxy-server:
    build: ./nginx
    ports:
      - 80:80
  app:
    build: .
    volumes:
      - ./public:/opt/app/public
      - ./routes:/opt/app/routes
      - ./views:/opt/app/views
  db:
    image: mongo:latest
    environment:
      MONGO_INITDB_ROOT_USERNAME:
"root"
      MONGO_INITDB_ROOT_PASSWORD:
"123456"
    volumes:
      - "my-mongo-volume:/data/db"
volumes:
  my-mongo-volume:
```

What is Kubernetes?

Kubernetes architecture


User interface
UI
CLI
kubectl
Control plane
API Server
Scheduler
Controller-Manager
etcd
Worker node 1
Pod 1
Container 1
Container 2
Container 3
Pod 2
Container 1
Pod 3
Container 1
Container 2
Docker
kubelet
Kube-proxy
Worker node 2
Pod 1
Container 1
Container 2
Pod 2
Container 1
Container 2
Container 3
Pod 3
Container 1
Docker
kubelet
Kube-proxy