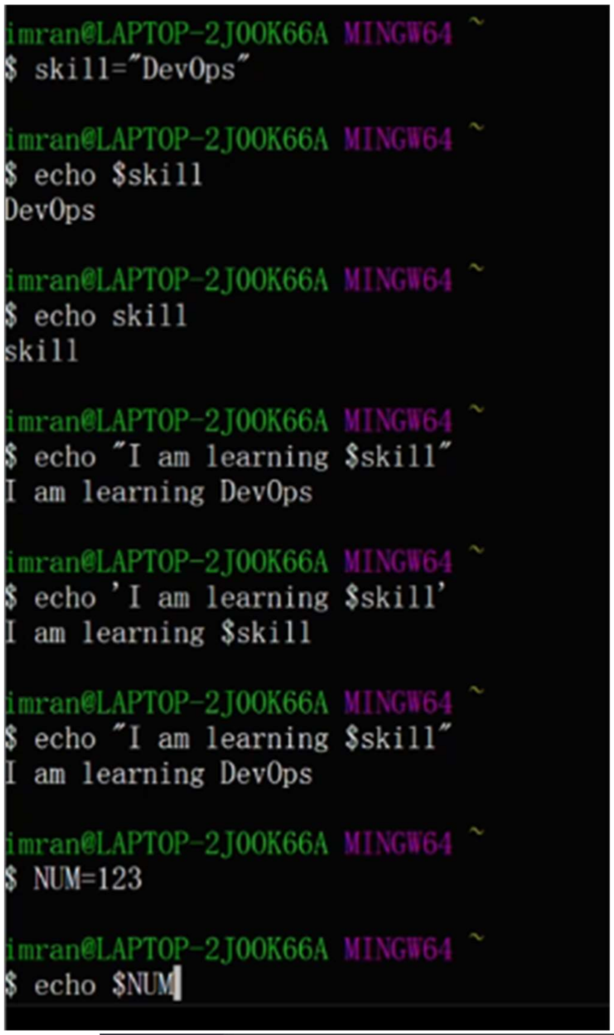
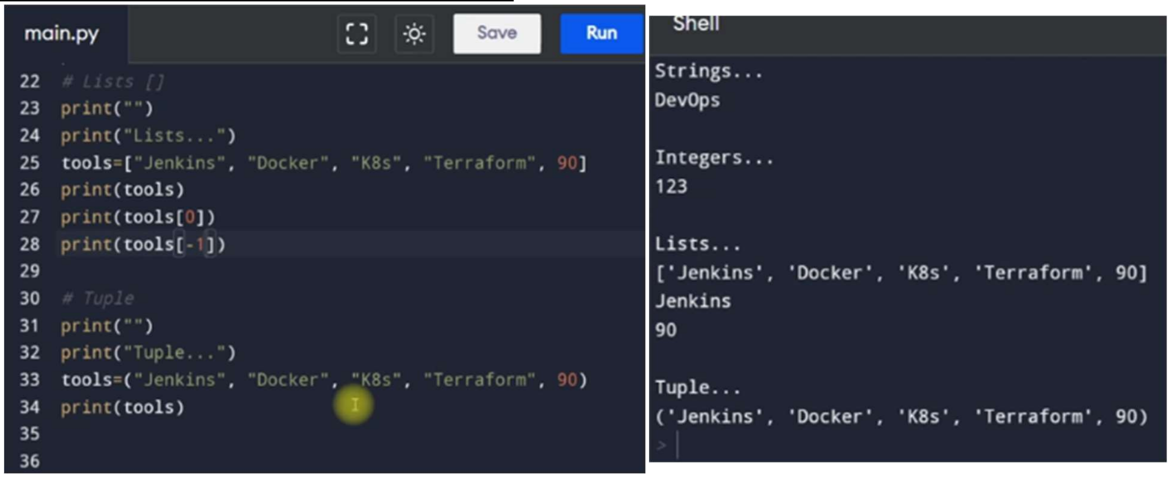
VARIABLES & PYTHON DS



```
imran@LAPTOP-2J00K66A MINGW64 ~
$ skill="DevOps"

imran@LAPTOP-2J00K66A MINGW64 ~
$ echo $skill
DevOps

imran@LAPTOP-2J00K66A MINGW64 ~
$ echo skill
skill

imran@LAPTOP-2J00K66A MINGW64 ~
$ echo "I am learning $skill"
I am learning DevOps

imran@LAPTOP-2J00K66A MINGW64 ~
$ echo 'I am learning $skill'
I am learning $skill

imran@LAPTOP-2J00K66A MINGW64 ~
$ echo "I am learning $skill"
I am learning DevOps

imran@LAPTOP-2J00K66A MINGW64 ~
$ NUM=123

imran@LAPTOP-2J00K66A MINGW64 ~
$ echo $NUM
```

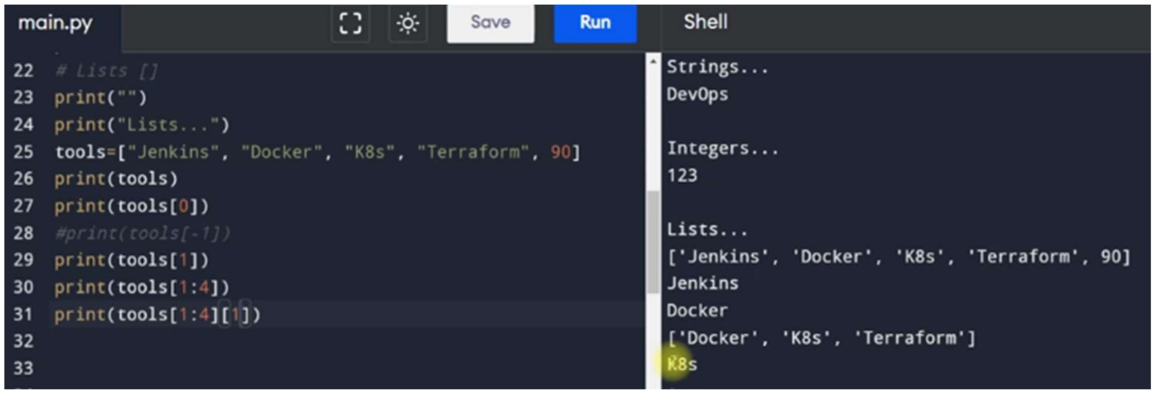
Here, we can see the basic bash script variables and printing them on the bash terminal, if we want to print a variable, we must prefix the variable name with the $ sign, we definitely use double quotes, when we are trying to print something, which also haves the variable to be printed in it. If we use a single quote, the variable won't be interpreted as a variable and you can see the example, how it will be represented. A variable in shell scripting can consists of any data type without any type declaration. Similarly , python variables also can store any datatype without any pre-datatype declaration.

Following are list and tuple examples:



```python
# Lists []
print("")
print("Lists...")
tools=["Jenkins", "Docker", "K8s", "Terraform", 90]
print(tools)
print(tools[0])
print(tools[-1])

# Tuple
print("")
print("Tuple...")
tools=("Jenkins", "Docker", "K8s", "Terraform", 90)
print(tools)
```

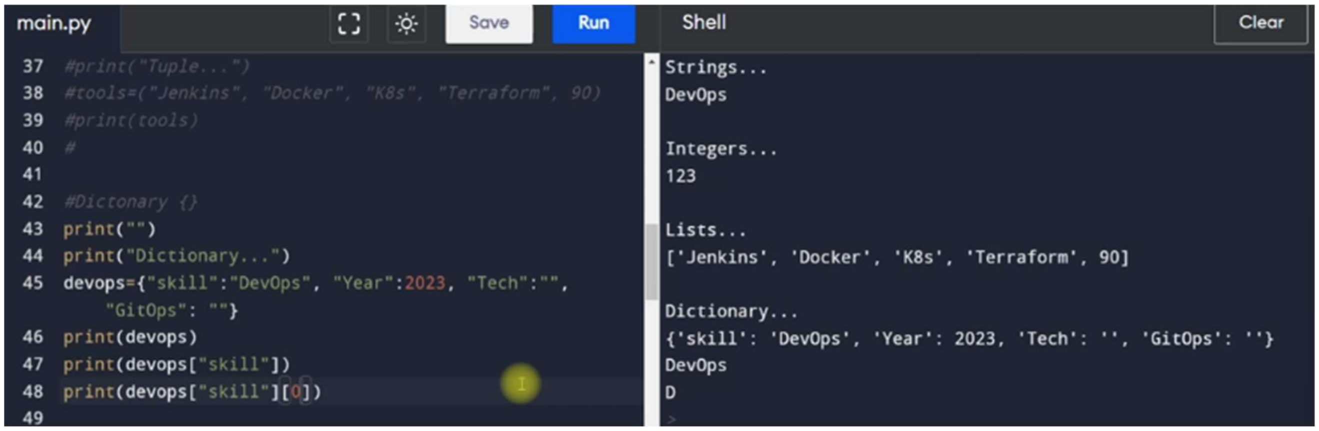
```
Strings...
DevOps

Integers...
123

Lists...
['Jenkins', 'Docker', 'K8s', 'Terraform', 90]
Jenkins
90

Tuple...
('Jenkins', 'Docker', 'K8s', 'Terraform', 90)
```



```python
# Lists []
print("")
print("Lists...")
tools=["Jenkins", "Docker", "K8s", "Terraform", 90]
print(tools)
print(tools[0])
#print(tools[-1])
print(tools[1])
print(tools[1:4])
print(tools[1:4][1])
```

```
Strings...
DevOps

Integers...
123

Lists...
['Jenkins', 'Docker', 'K8s', 'Terraform', 90]
Jenkins
Docker
['Docker', 'K8s', 'Terraform']
K8s
```

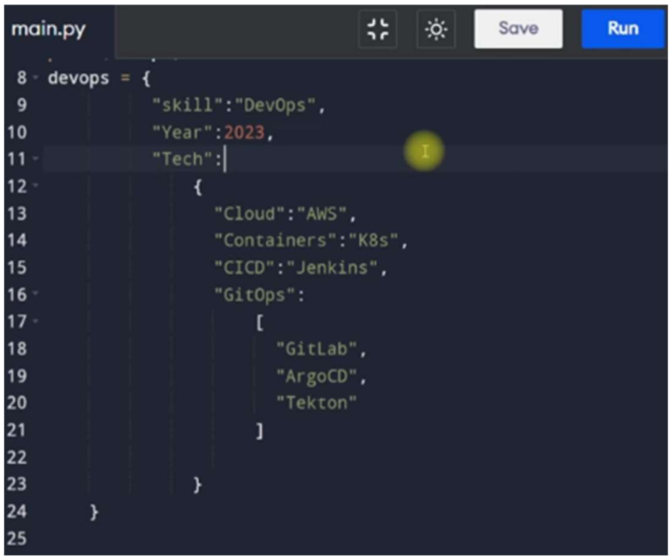
Following is the example of dictionary datatype:



```
#print("Tuple...")
#tools=("Jenkins", "Docker", "K8s", "Terraform", 90)
#print(tools)
#


#Dictonary {}
print("")
print("Dictionary...")
devops={"skill":"DevOps", "Year":2023, "Tech":"",
    "GitOps": ""}
print(devops)
print(devops["skill"])
print(devops["skill"][0])
```

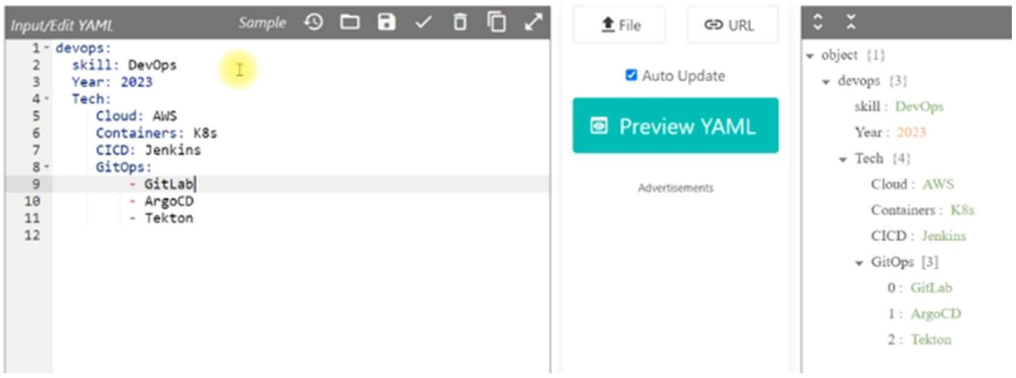
```
Strings...
DevOps

Integers...
123

Lists...
['Jenkins', 'Docker', 'K8s', 'Terraform', 90]

Dictionary...
{'skill': 'DevOps', 'Year': 2023, 'Tech': '', 'GitOps': ''}
DevOps
D
```

Basically JSON is a dictionary of dictionary, which will be storing it's values in a list, if we prettify the definition which I said now, it will be the following:



```
devops = {
          "skill":"DevOps",
          "Year":2023,
          "Tech":
              {
                "Cloud":"AWS",
                "Containers":"K8s",
                "CICD":"Jenkins",
                "GitOps":
                    [
                      "GitLab",
                      "ArgoCD",
                      "Tekton"
                    ]

              }
     }
```

If we remove all these extra braces and the keys of the python dictionary will be represented as individual keys in yaml, and their numeric and string values will be represented with ':' correspondingly. And in the second level, to declare key's value's list of strings as values, we are using list in python and JSON, and instead of that, to represent multiple values under a key's value's value, we need to represent those values with '-' hi-Phen correspondingly. That will be the YAML file. To write the configurations, mostly yaml is used, and to read the data mostly Json is used.

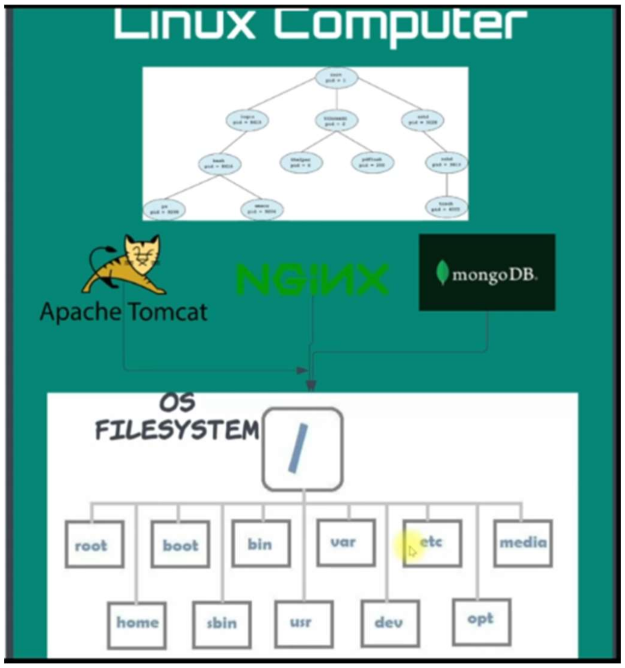


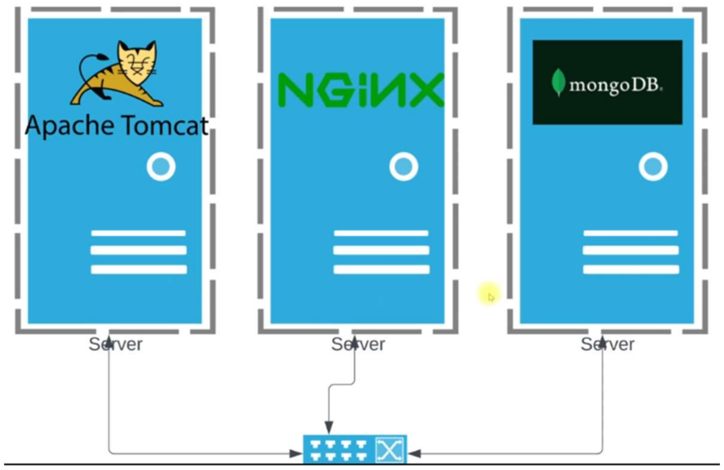
```
devops:
  skill: DevOps
  Year: 2023
  Tech:
     Cloud: AWS
     Containers: K8s
     CICD: Jenkins
     GitOps:
          - GitLab
          - ArgoCD
          - Tekton
```

```
object {1}
  devops {3}
    skill : DevOps
    Year : 2023
    Tech {4}
      Cloud : AWS
      Containers : K8s
      CICD : Jenkins
      GitOps [3]
        0 : GitLab
        1 : ArgoCD
        2 : Tekton
```

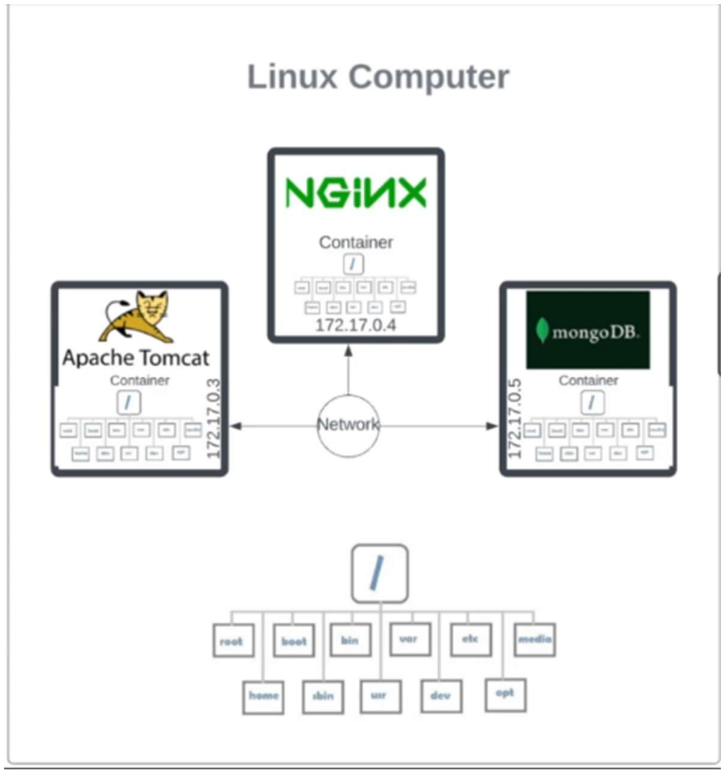
Containers Introduction

Basically , this is the linux file system , and we will be having multiple processes running on the host , where each processes make different changes to the file system , which can also affect other process of the file system , so , to maintain , isolation of these processes , we can use different computers , which will have independent file system , but that would be resource costly. These computers can be a physical machine or even a virtual machine.

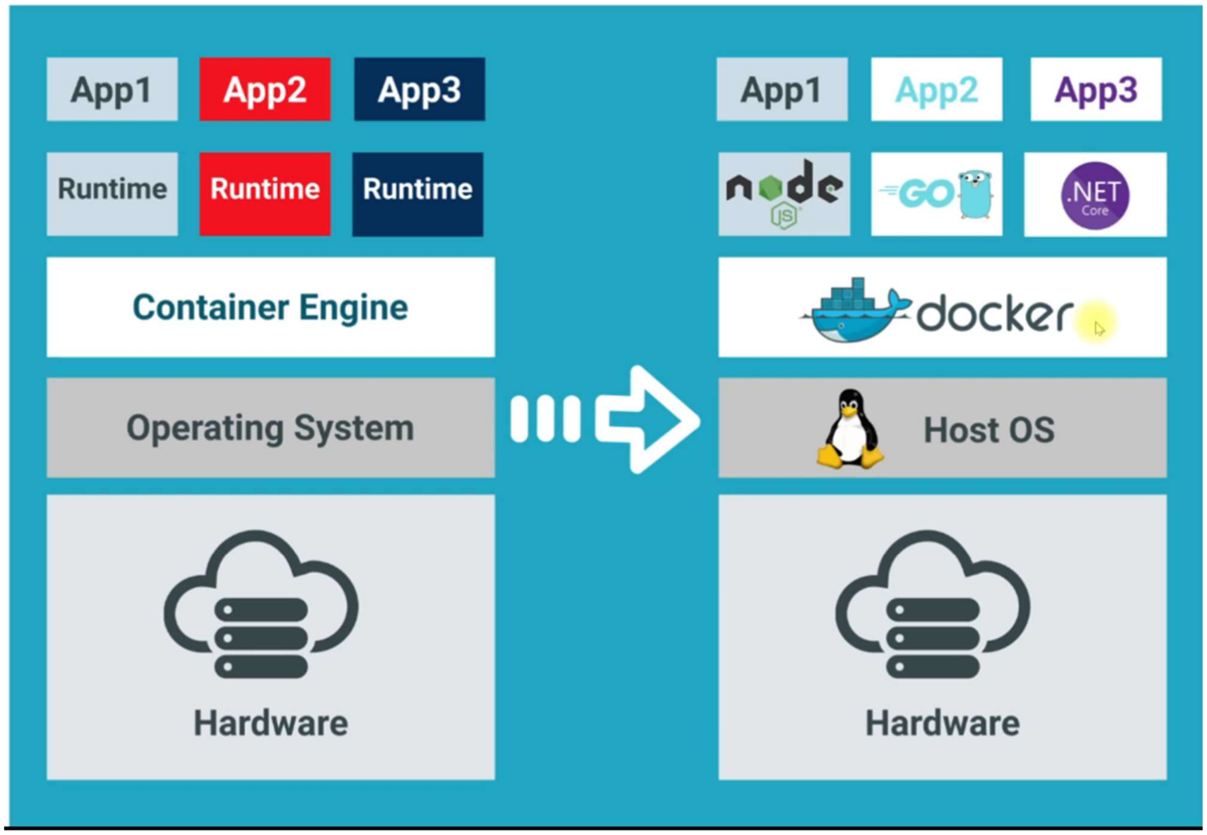




So, instead of maintaining different computers for isolation, which will be costly, we can introduce the concept containers, so that each process will be running on an individual container, where the communication between them will be through a virtual network. Where these containers have their own individual file systems.



These containers are so small in size, as they have only the software that need to run and related files in it, it doesn't have the whole OS running in it, we will have a snapshot of the OS that is required to run the process, so it will be light weight and isolated in environment. As this is small, we can achieve the image and run the containers wherever we need to run either on dev environment or production or on testing wherever we need.



The containerization runs in similar architecture on the host machine or virtual machine that we run containers on. We will be having host / virtual machine on which we have host OS, on top of that, we would install docker container engine, which helps us to create multiple runtime environment for different applications to work on. This architecture , helps us in isolation process.