KPLABS Course

Certified Kubernetes Application Developer 2022

New Exam Updates

ISSUED BY
Zeal Vora

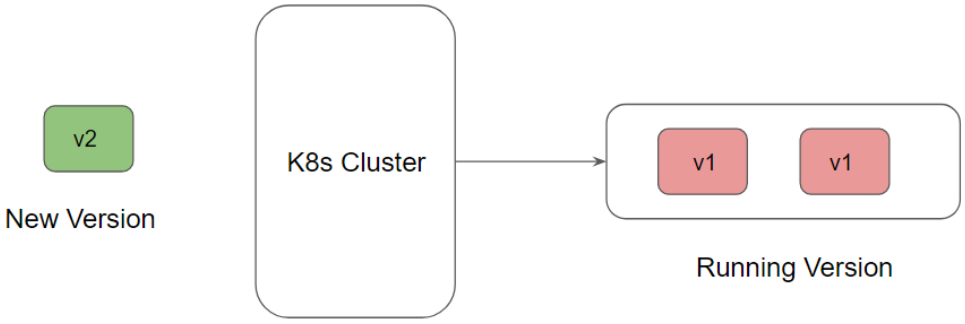
REPRESENTATIVE
instructors@kplabs.in

Module 1: K8s Deployment Strategy

1.1 Overview of Deployment Strategy

A deployment strategy is a way to change or upgrade an application

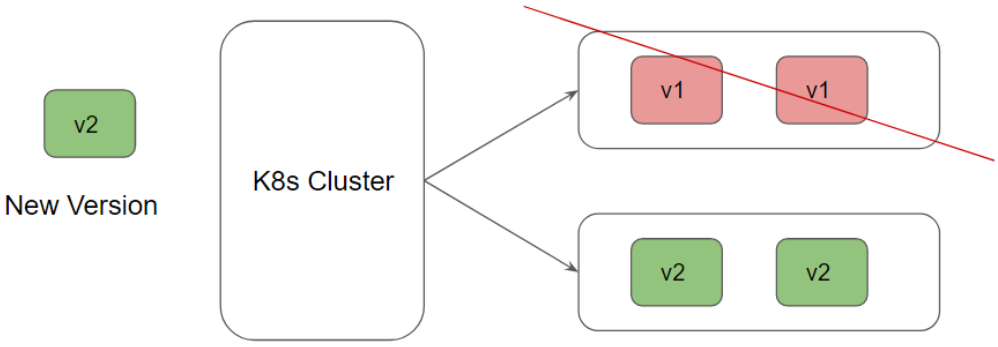
How will you deploy v2 of the application to the Kubernetes cluster?



1.2 Deployment Strategy - Recreate

All of the PODS get killed all at once and get replaced all at once with the new ones.

Recommended for Dev/Test environment.



Advantages of Recreate Strategy:

- Overall easy to setup and deploy.

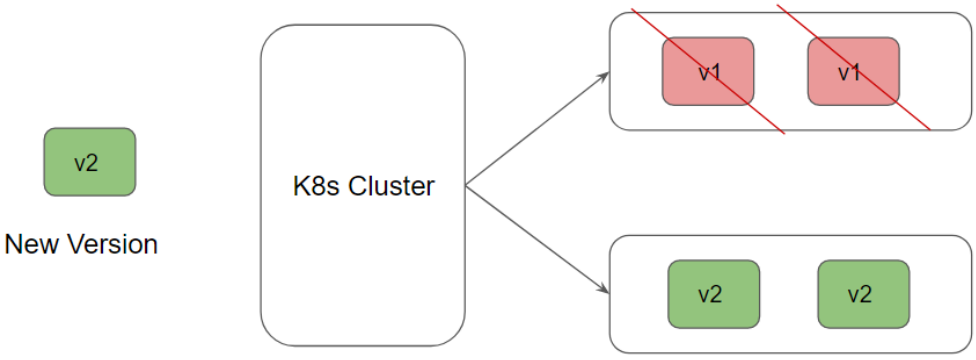
Disadvantages of Recreate Strategy:

- There will be a certain amount of downtime for the users.

Traffic Pattern - Recreate Strategy

1.3 Deployment Strategy - RollingUpdate

Your application is deployed to your environment one batch of instances at a time.



Traffic Pattern - Rolling Update Strategy

Advantages of Recreate Strategy

- New version is slowly released and can reduce the downtime.

Disadvantages of Recreate Strategy:

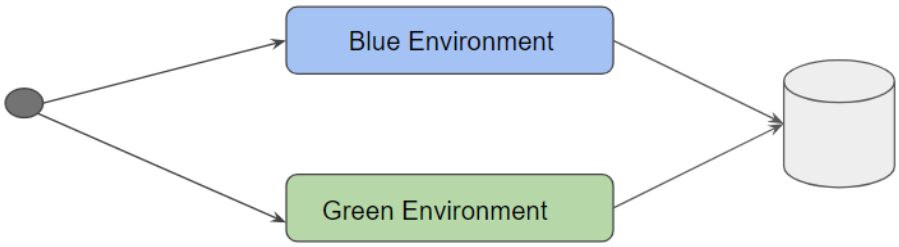
- Takes more amount of time for the overall rollout process.
- No control over the traffic.

1.4 Blue Green Deployment Strategy

Blue environment is an existing environment in production receiving live traffic.

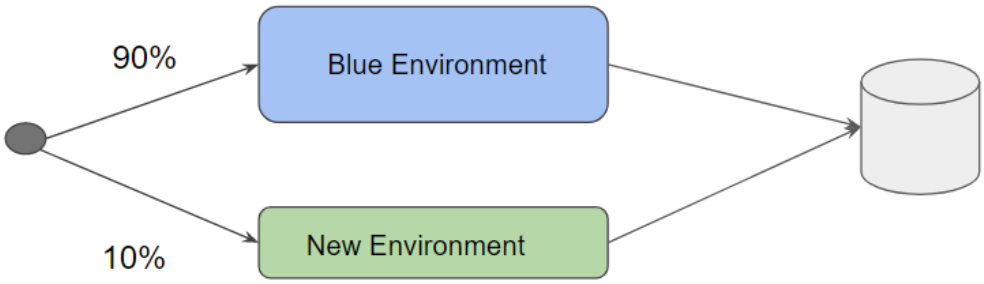
Green environment is a parallel environment running different versions of the application.

Deployment = Routing production traffic from blue to green environment.



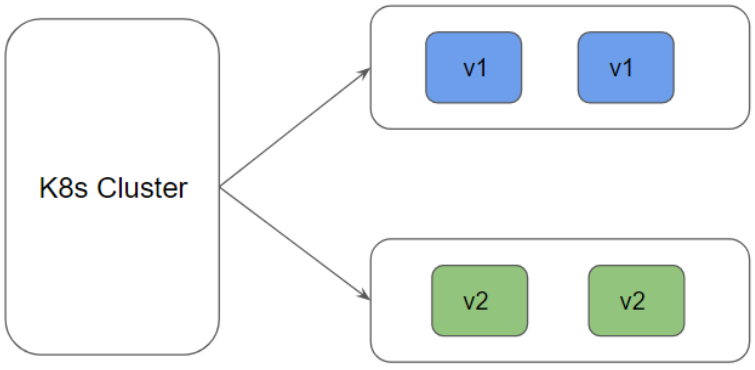
1.5 Understanding Canary Deployment Model

Canary Deployment is a process where we deploy a new feature and shift some % of traffic to the new feature to perform some analysis to see if the feature is successful.

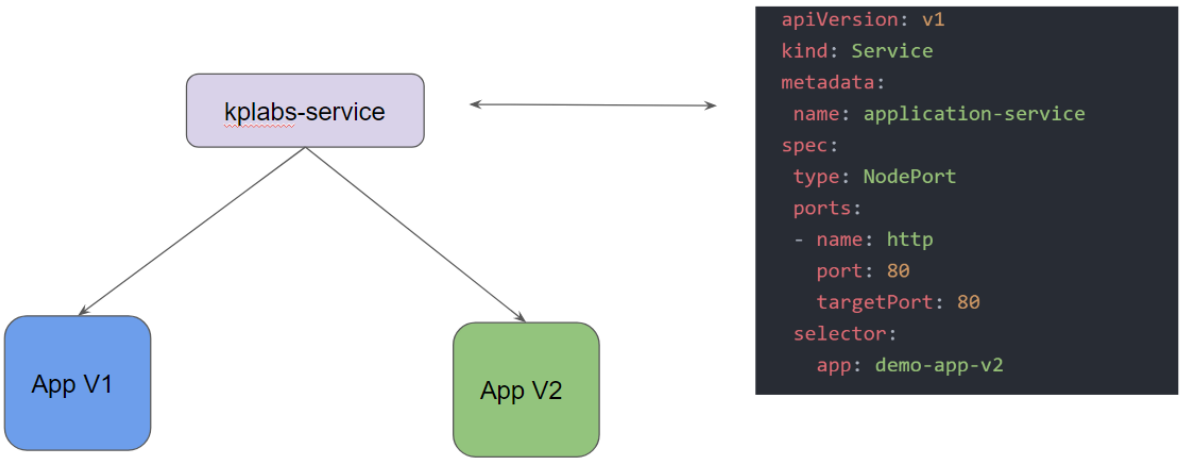


Module 2: Blue/Green Deployments

Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.



Possible Implementation Method - 1



Advantages of Blue/Green Strategy

- Instantaneous rollouts and rollbacks if required.

Disadvantages of Blue/Green Strategy

- Requires double the amount of resources.

Module 3: Canary Deployments

3.1 Story behind Canary

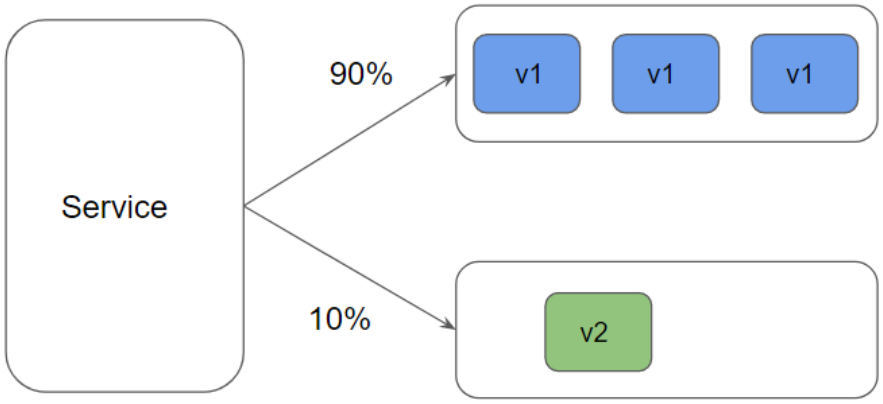
The story goes back to the old British mining practice.

Practice included using "canaries in coal mines to detect carbon monoxide and other toxic gases before they hurt humans."

To make sure mines were safe for them to enter, miners would take canaries; if something bad happened to the canary, it was a warning for the miners to abandon the mine.

3.2 Deployment Strategy - Canary

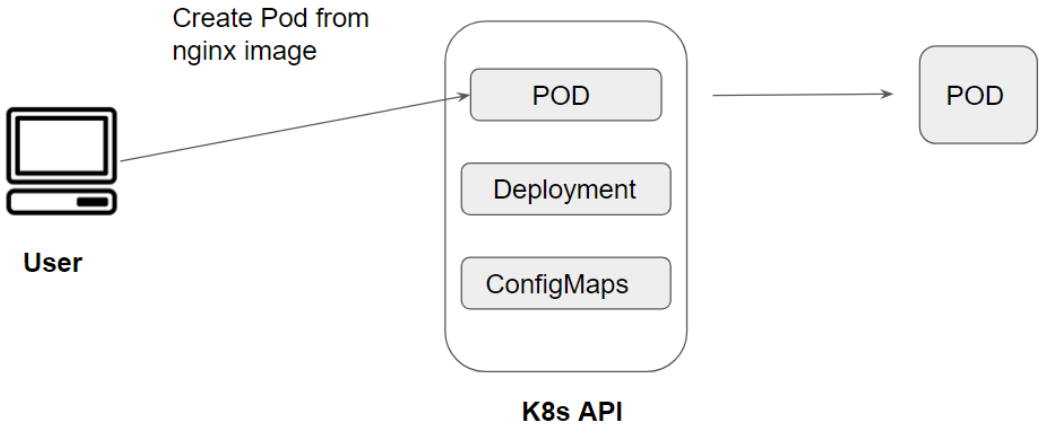
Canary Deployment is a process where we deploy a new feature and shift some % of traffic to the new feature to perform some analysis to see if feature is successful.



Module 4: Custom Resources

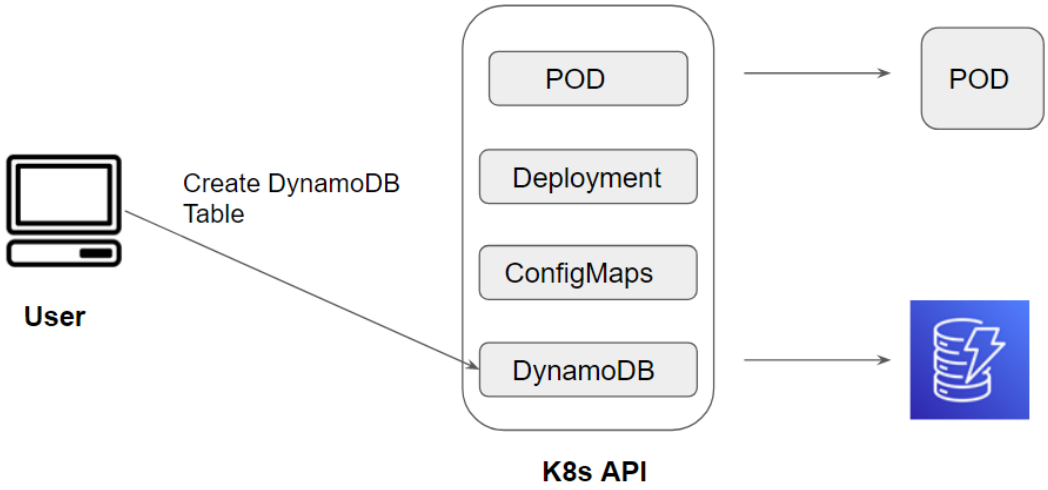
4.1 Understanding the Basics

A resource is an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind.



4.2 Understanding the CRDs

Custom Resource Definition (CRDs) are extensions of Kubernetes API that stores collections of API objects of certain kind.



4.3 Step 1 - Create Custom Resource Definition

To create a CRD, you need to create a file that defines your object kinds.

Applying a CRD into the cluster makes the Kubernetes API server to serve the specified custom resource.

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.kplabs.internal
spec:
  group: kplabs.internal
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: number
```

Step 2 - Create Custom Objects

After the CustomResourceDefinition object has been created, you can create custom objects.

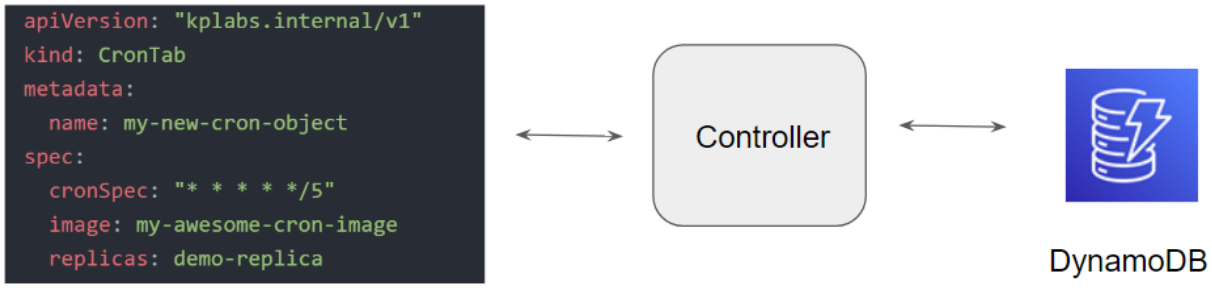
Applying a CRD into the cluster makes the Kubernetes API server to serve the specified custom resource.

```yaml
apiVersion: "kplabs.internal/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
  replicas: demo-replica
```

Importance of Kubernetes Controller

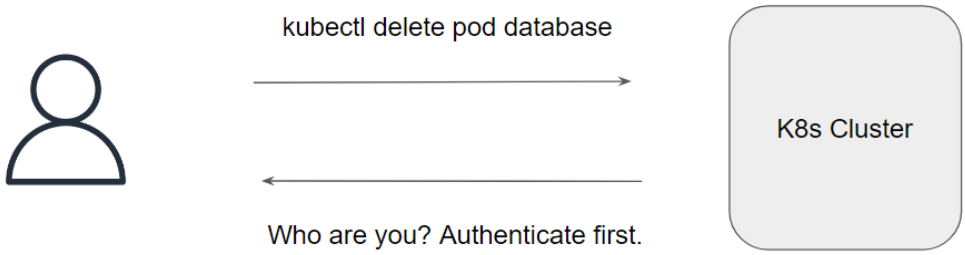
A controller tracks at least one Kubernetes resource type.

These objects have a spec field that represents the desired state.



Module 5: Authentication

Authentication is the process or action of verifying the identity of a user or process.

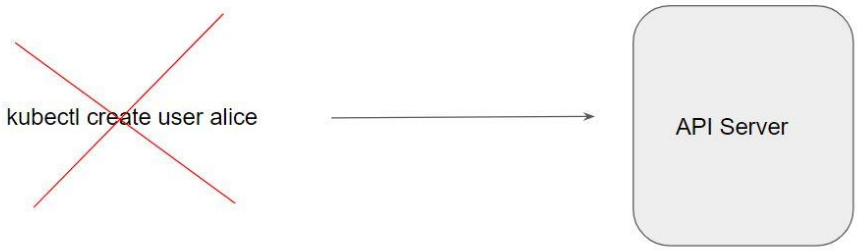


Kubernetes Clusters have two categories of users:

- Normal Users.
- Service Accounts

Kubernetes does not have objects which represent normal user accounts.

Kubernetes does not manage the user accounts natively.

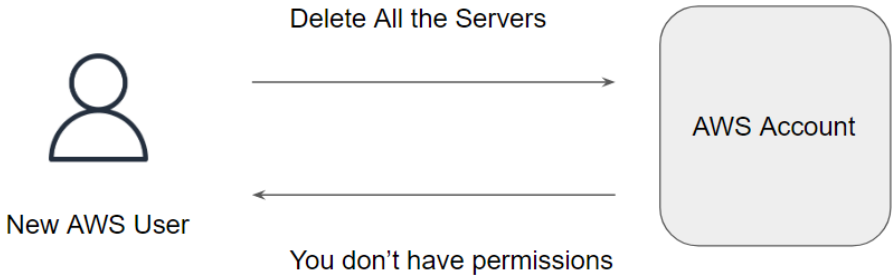


There are multiple ways in which we can authenticate. Some of these include:

- Usernames / Passwords.
- Client Certificates
- Bearer Tokens

Module 5: Authorization

Authorization is the process of giving someone permission to perform some operation.



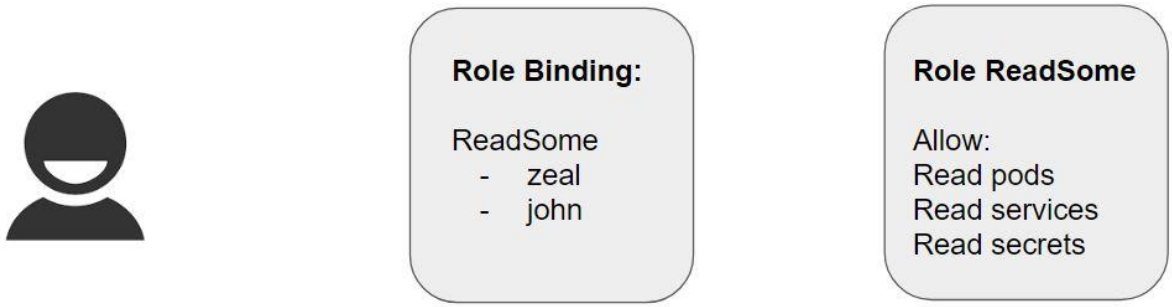
By default, the user does not have any permission granted to them.

Kubernetes provides multiple authorization modules like:

- AlwaysAllow
- AlwaysDeny
- Attribute-Based Access Control (ABAC)
- Role-based access control (RBAC)
- Node
- WebHook

A role contains rules that represent a set of permissions

A role binding grants the permissions defined in a role to a user or set of users.



Module 6: ClusterRole and ClusterRoleBinding

A Role can only be used to grant access to resources within a single namespace.

A ClusterRole can be used to grant the same permissions as a Role, but because they are cluster-scoped, they can also be used to grant access to:

cluster-scoped resources (like nodes)
namespaced resources (like pods) across all namespaces

Important Pointer:

A RoleBinding may also reference a ClusterRole to grant the permissions to resources defined in the ClusterRole within the Role Bindings namespace

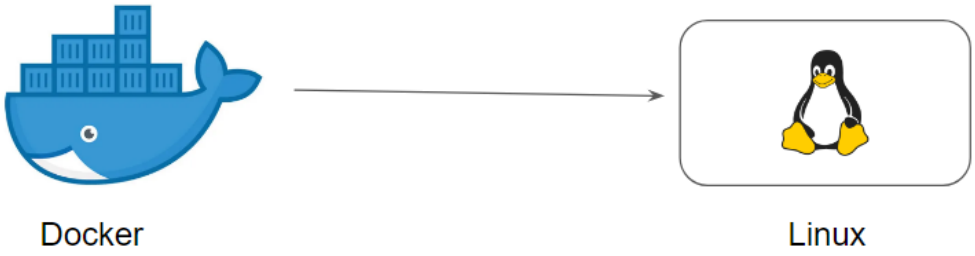
This allows the administrator to have set of central policy which can be attached via RoleBinding so it is applicable at a per-namespace level.

Module 7: Infrastructure for Docker

7.1 Revising the Preferred Choice

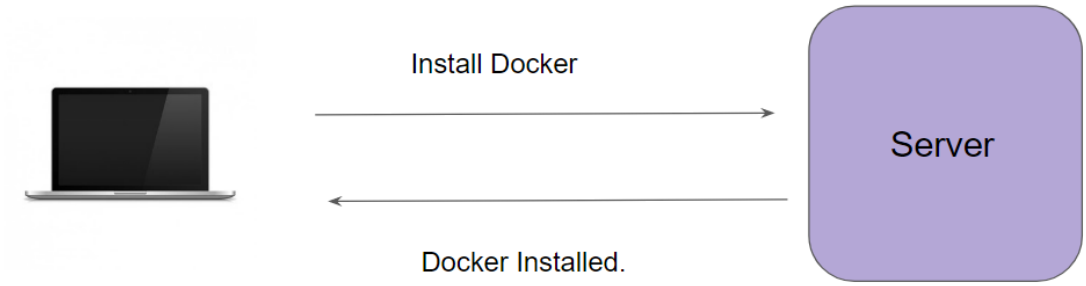
To begin with, you can install Docker Desktop directly within your laptop.

The preferred OS for Docker installation would be Linux.



7.2 Creating Infrastructure for Docker

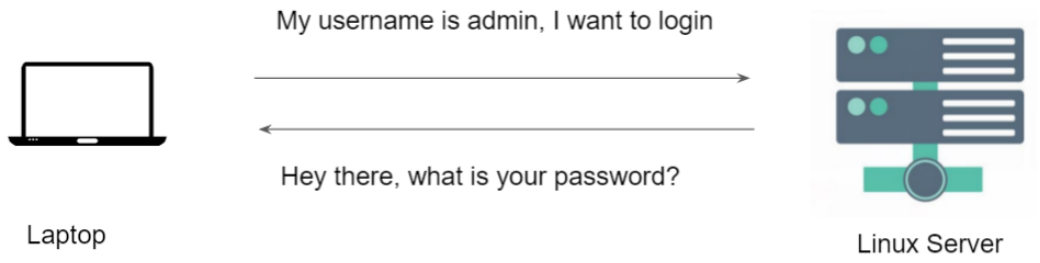
To begin with the Docker installation process, we need one server hosting Ubuntu OS.



7.3 Password-Based Authentication

There can be multiple methods for authentication against a system.

Password-based authentication is the simplest form.



7.4 Challenges with Password-Based Authentication

Password-based authentication is generally considered to be less secure.

Many users write down the passwords in notepad files or as part of sticky notes.

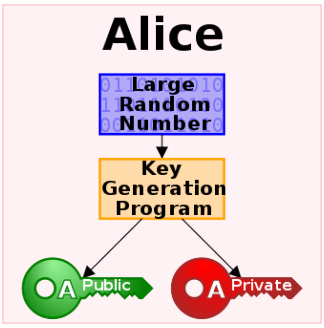
Most users would not create a complex password that is difficult to hack.

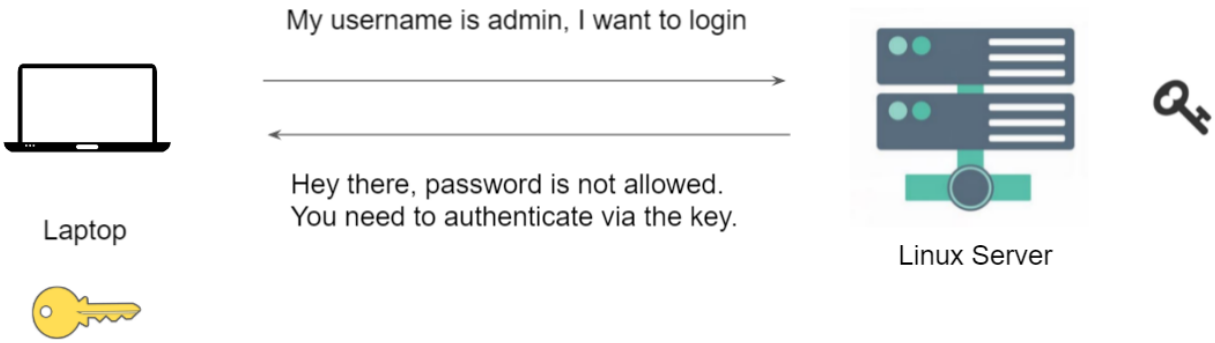
7.5 Key Based Authentication

In this type of authentication, there are two special keys that are generated.

One key is called a Public Key and the second key is called a Private key.

If the public key is stored in the server and is used as an authentication mechanism, only the corresponding private key can be used to successfully authenticate.





7.6 Firewall Rules

We do not want the entire internet to connect to our server.

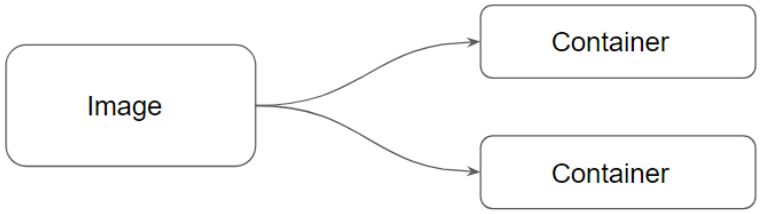
With the help of Firewall, you can restrict the connection to your Docker instance.

Ports	Description
22	Connection to SSH.

Module 8: Docker Image vs Docker Container

Docker Image is a file which contains all the necessary dependencies and configurations which are required to run an application.

Docker Containers is basically a running instance of an image.



Module 9: Container Identification

When you create a Docker container, it is assigned a universally unique identifier (UUID).

These can help identify the docker container among others.

```
[root@docker-demo ~]# docker run -dt -p 80:80 nginx
d5187cb1c7f4380b3e37e0c0c811a437d7b8a49d5beb705711a4e54e99d72d77
```

To help humans, Docker also allow us to supply container names.

By default, if we do not specify the name, docker supplies randomly-generated name from two words, joined by an underscore

```
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
      PORTS                NAMES
d5187cb1c7f4        nginx               "nginx -g 'daemon ..."   47 minutes ago      Up 31 minutes
      0.0.0.0:80->80/tcp   inspiring_poitras
```

By adding --name=meaningful_name argument during docker run command, we can specify our own name to the containers.

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
      PORTS                  NAMES
9fba1f62038d        nginx               "nginx -g 'daemon ..."   5 seconds ago       Up 5 seconds
      0.0.0.0:800->80/tcp   mynginx
```

Module 10: Working with Docker Images

Every Docker container is based on an image.

Till now we have been using images which were created by others and available in Docker Hub.

Docker can build images automatically by reading the instructions from a Dockerfile

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

Module 11: Revising Dockerfile

11.1 Format of Dockerfile

The format of Dockerfile is similar to the below syntax:

Comment
INSTRUCTION arguments

A Dockerfile must start with a FROM` instruction.
The FROM instruction specifies the Base Image from which you are building.

11.2 Dockerfile Commands

Thee format of Dockerfile is similar to the below syntax:

- FROM
- RUN
- CMD
- LABEL
- EXPOSE
- ENV
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER
- Many More ...

11.3 Use-Case - Create Custom Nginx Image

We want to create a custom Nginx Image from which all containers within the organization will be launched.

The base container image should have custom index.html file which states:

"Welcome to Base Nginx Container from KPLABS"

Module 12: Docker Tags

Docker tags convey useful information about a specific image version/variant.

They are aliases to the ID of your image which often look like this: 8f5487c8b942

```
[root@docker-demo ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              caf27325b298        15 hours ago        5.53MB
<none>              <none>              7c116aacaae3        17 hours ago        172MB
```

Module 13: Docker Commit

Whenever you make changes inside the container, it can be useful to commit a container's file changes or settings into a new image.

By default, the container being committed and its processes will be paused while the image is committed.

Syntax:

docker container commit CONTAINER-ID myimage01

The --change option will apply Dockerfile instructions to the image that is created.

Supported Dockerfile instructions:

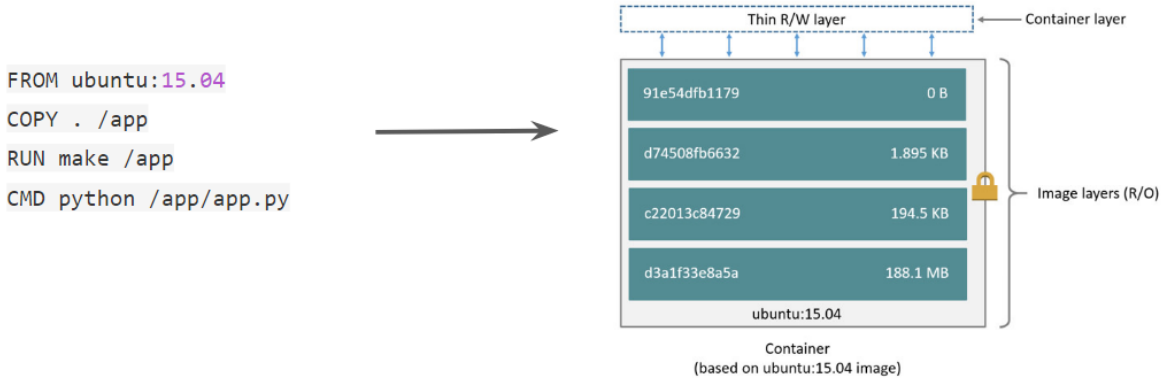
CMD | ENTRYPOINT | ENV | EXPOSE
LABEL | ONBUILD | USER | VOLUME | WORKDIR

Module 14: Layers of Docker Image

14.1 Understanding Layers

A Docker image is built up from a series of layers.

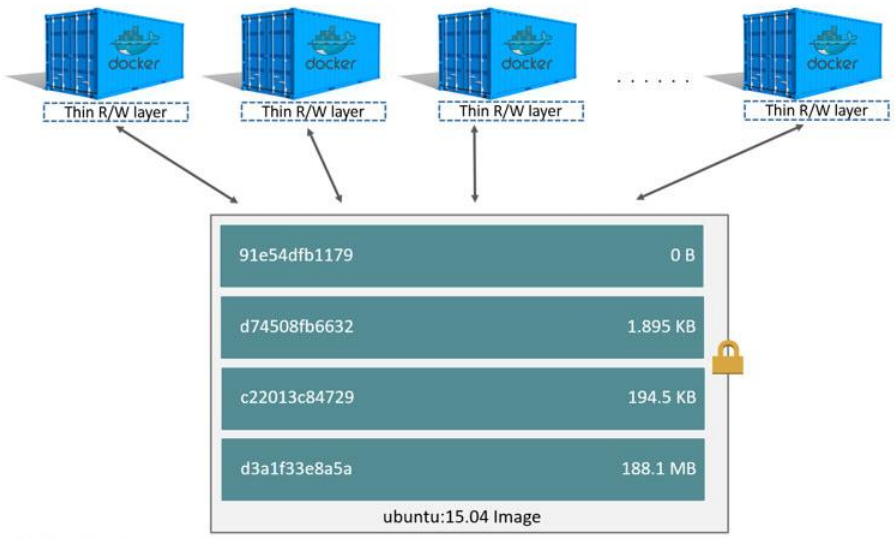
Each layer represents an instruction in the image's Dockerfile.



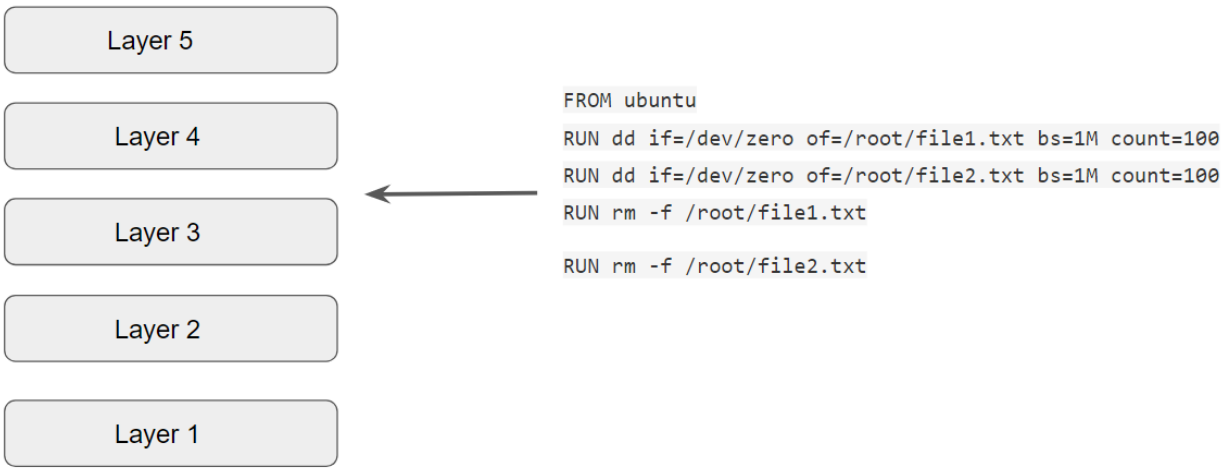
14.2 Containers and Layers

The major difference between a container and an image is the top writable layer.

All writes to the container that add new or modify existing data are stored in this writable layer.

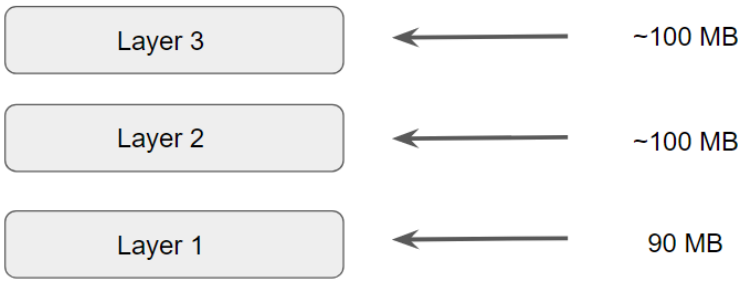


14.3 Image and Layer

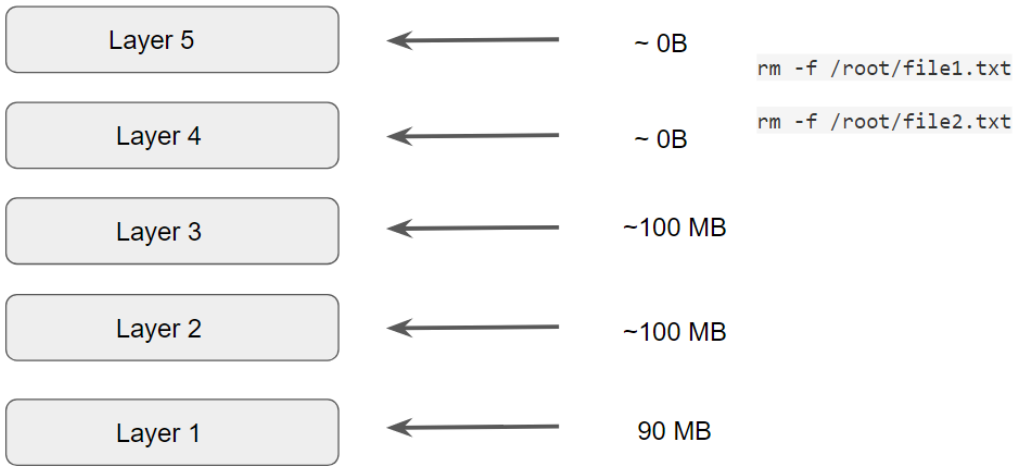


14.4 First Three Instructions

```
FROM ubuntu
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
```



14.4 ALL Instruction



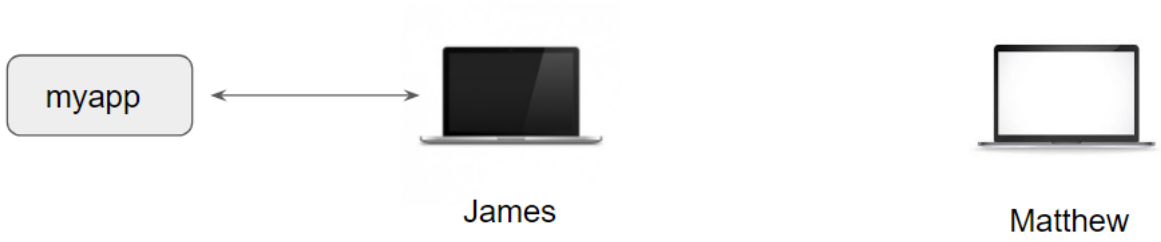
.

Module 15: Moving Images Across Hosts

15.1 Example Use-Case

James has created an application based on Docker. He has the image file in his laptop.

He wants to send the image to Matthew over email.



15.2 Overview of Docker Save

The docker save command will save one or more images to a tar archive

Example Snippet:

```
docker save busybox > busybox.tar
```

15.3 Overview of Docker Load

The docker load command will load an image from a tar archive

Example Snippet:

```
docker load < busybox.tar
```

.

Join Our Discord Community

We invite you to join our Discord community, where you can interact with our support team for any course-based technical queries and connect with other students who are doing the same course.

Joining URL:

http://kplabs.in/chat