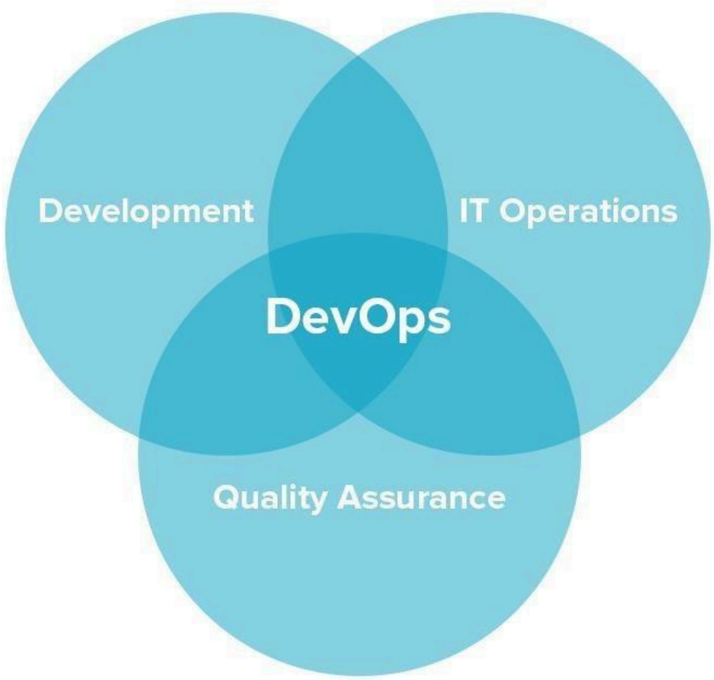
Aim: To Understand DevOps: Principles, Practices, DevOps Engineer Roles and Responsibilities.

Lab Outcome: To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements

Theory:

DevOps Definition:

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to testing, deployment, and operations. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.



DevOps Principles:

The main principles of DevOps are Continuous delivery, automation, and fast reaction to the feedback.

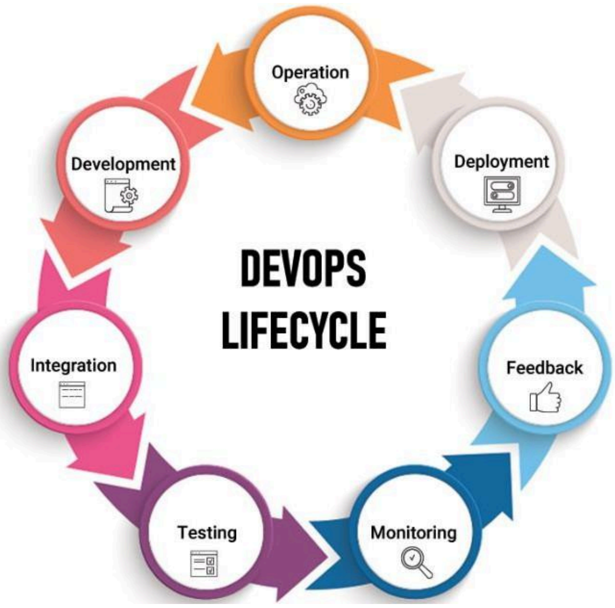
1. **End to End Responsibility:** DevOps team need to provide performance support until they become the end of life. It enhances the responsibility and the quality of the products engineered.
2. **Continuous Improvement:** DevOps culture focuses on continuous improvement to minimize waste. It continuously speeds up the growth of products or services offered.
3. **Automate Everything:** Automation is an essential principle of the DevOps process. This is for software development and also for the entire infrastructure landscape.
4. **Custom Centric Action:** DevOps team must take customer-centric for that they should continuously invest in products and services.
5. **Monitor and test everything:** The DevOps team needs to have robust monitoring and testing procedures.
6. **Work as one team:** In the DevOps culture role of the designers, developers, and testers are already defined. All they needed to do is work as one team with complete collaboration.

DevOps Practices:

Some identified DevOps practices are:

- Self-service configuration
- Continuous build
- Continuous integration
- Continuous delivery
- Incremental testing
- Automated provisioning
- Automated release management

Stages of Devops:



1.Continuous development

Continuous development involves planning and coding the software. Here, the entire development process gets broken down into smaller development cycles. This process makes it easier for the DevOps team to accelerate the overall software development process. This phase is instrumental in mapping the vision for the entire development cycle, enabling developers to fully understand project expectations. Through this, the team starts visualizing its end goal as well.

2. Continuous integration

Continuous integration (CI) includes different steps related to the execution of the test process. Along with this, clients also provide information to be incorporated for adding new features to the application. Most changes happen in the source code during this phase. CI becomes the hub for resolving these frequent changes on a daily or monthly basis.

3. Continuous testing

Next in the DevOps lifecycle is the testing phase, wherein the developed code is tested for bugs and errors that may have made their way into the code. This is where quality analysis (QA) plays a major role in checking the usability of the developed software.

4. Continuous deployment

Continuous deployment (CD) ensures hassle-free product deployment without affecting the application's performance. It is necessary to ensure that the code is deployed precisely on all available servers during this phase.

5. Continuous monitoring

Monitoring the performance of a software product is essential to determine the overall efficacy of the product output. This phase processes important information about the developed app.

6. Continuous feedback

Continuous feedback is essential to ascertain and analyze the final outcome of the application. It sets the tone for improving the current version and releasing a new version based on stakeholder feedback.

7. Continuous operations

The last stage in the DevOps lifecycle is the shortest and easiest to grasp. Continuity is at the heart of all DevOps operations that helps automate release processes, allows developers to detect issues quickly, and build better versions of software products. Continuation is key to eliminate diversions and other extra steps that hinder development.

DevOps Engineer Roles and Responsibilities:

DevOps engineers work full time. They are responsible for the production and continuing maintenance of a software application platform.
Below are some roles, responsibilities, and skills which are expected from DevOps engineers, such as:

- Manage projects effectively through an open standard based platform.
- Increases project visibility through traceability.
- Improve quality and reduce the development cost with collaboration.
- DevOps should have the soft skill of problem solver and a quick learner.
- Analyze, design, and evaluate automation scripts and systems.
- Able to perform system troubleshooting and problem-solving across the platform and application domains.
- Ensuring the critical resolution of system issues by using the best cloud security solution services.

Conclusion:

The basic introduction and the DevOps concept were clearly explained in this experience. The definition, principles and practices for DevOps were understood.

Aim: To Understand Version Control System / Source Code Management, install git and to perform various GIT operations on local remote repositories.

Lab Outcome: To obtain complete knowledge of the "version control system" to effectively track changes augmented with Git and GitHub

Theory:

Definition of Git:

- Git is an open-source distributed version control system. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.
- Git is foundation of many services like GitHub and GitLab, but we can use Git without using any other Git services. Git can be used privately and publicly.
- Git was created by Linus Torvalds in 2005 to develop Linux Kernel. It is also used as an important distributed version-control tool for the DevOps.
- Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

Features of Git:

1. **Open Source**:
   - Git is an open-source tool. It is released under the GPL (General Public License) license.

2. **Scalable**:
   - Git is scalable, which means when the number of users increases, the Git can easily handle such situations.
3. **Distributed**:
   - One of Git's great features is that it is distributed. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository.
   - Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project.
   - We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.
4. **Security**:
   - Git is secure. It uses the SHA1 (Secure Hash Function) to name and identify objects within its repository.
   - Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit.
   - Once it is published, one cannot make changes to its old version.
5. **Speed**:
   - Git is very fast, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a huge speed.
   - Also, a centralized version control system continually communicates with a server somewhere.
     Performance tests conducted by Mozilla showed that it was extremely fast compared to other VCSs.
   - Fetching version history from a locally stored repository is much faster than fetching it from the remote server. The core part of Git is written in C, which ignores runtime overheads associated with other high-level languages.
   - Git was developed to work on the Linux kernel; therefore, it

     is capable enough to handle large repositories effectively. From the beginning, speed and performance have been Git's primary goals.

GitHub Definition:

GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

Features of GitHub:

- Collaboration
- Integrated issue and bug tracking
- Graphical representation of branches
- Git repositories hosting
- Project management
- Team management
- Code hosting
- Track and assign tasks
- Conversations

Difference of Git & GitHub

S.No.	Git	GitHub
1.	Git is a software.	GitHub is a service.
2.	Git is a command-line tool	GitHub is a graphical user interface
3.	Git is installed locally on the system	GitHub is hosted on the web
4.	Git is maintained by linux.	GitHub is maintained by microsoft.
5.	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6.	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7.	Git was first released in 2005.	GitHub was launched in 2008.
8.	Git has no user management feature.	GitHub has built-in user management feature.

Version Control System:

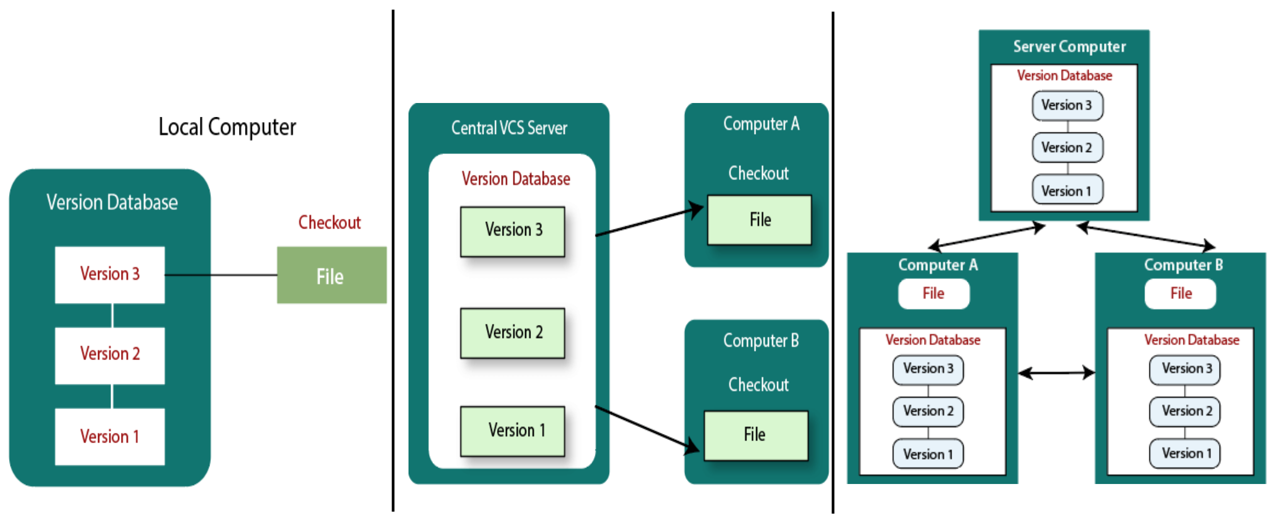
- A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

- The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.
- Developers can compare earlier versions of the code with an older version to fix the mistakes.

Benefits:

- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability

Types of Version Control System



1) Localized version Control System

2) Centralized version control systems
3) Distributed version control systems

Git Cheat Sheet

1. Git configuration

- **Git config**

  Get and set configuration variables that control all facets of how Git looks and operates.
  Set the name:

  $ git config --global user.name "User name"

Set the email:

$ git config --global user.email "Krisha.sayla@gmail.com"

Set the default editor:
$ git config --global core.editor Vim

Check the setting:

$ git config -list

- **Git alias**
  Set up an alias for each command:
  $ git config --global alias.co checkout

  $ git config --global alias.br branch

  $ git config --global alias.ci commit

  $ git config --global alias.st status

2. Starting a project

- **Git init**

  Create a local repository:

  $ git init
- **Git clone**
  Make a local copy of the server repository.

  $ git clone

3. Local changes

- **Git add**

  Add a file to staging (Index) area:

  $ git add Filename

  Add all files of a repo to staging (Index) area:

  $ git add*
- **Git commit**

  Record or snapshots the file permanently in the version history **with a message**.
  $ git commit -m " Commit Message"

4. Track changes

- **Git diff**

  Track the changes that have not been staged:

$ git diff

Track the changes that have staged but not committed:

$ git diff --staged
Track the changes after committing a file:

$ git diff HEAD

Track the changes between two commits:

$ git diff Git Diff Branches:
$ git diff < branch 2>

- **Git status**

  Display the state of the working directory and the staging area.

  $ git status
- **Git show Shows objects:**
  $ git show

5. Commit History

- **Git log**

  Display the most recent commits and the status of the head:
  $ git log

  Display the output as one commit per line:

  $ git log -oneline

  Displays the files that have been modified:

  $ git log -stat
  Display the modified files with location:

  $ git log -p
- **Git blame**
  Display the modification on each line of a file:

  $ git blame <file name>

6. Ignoring files

- **.gitignore**
  Specify intentionally untracked files that Git should ignore. Create

  .gitignore:
  $ touch .gitignore List the ignored files:

  $ git ls-files -i --exclude-standard

7. Branching

- **Git branch Create branch:**

  $ git branch List Branch:

  $ git branch --list Delete a Branch:
  $ git branch -d Delete a remote Branch:

  $ git push origin -delete Rename Branch:

  $ git branch -m
- **Git checkout**
  Switch between branches in a repository.
  Switch to a particular branch:
  $ git checkout

  Create a new branch and switch to it:

  $ git checkout -b Checkout a Remote branch:

  $ git checkout
- **Git stash**
  Switch branches without committing the current branch. Stash current work:

  $ git stash

  Saving stashes with a message:

  $ git stash save ""
  Check the stored stashes:

  $ git stash list

  Re-apply the changes that you just stashed:

  $ git stash apply

  Track the stashes and their changes:

  $ git stash show
  Re-apply the previous commits:

  $ git stash pop

  Delete a most recent stash from the queue:

  $ git stash drop

  Delete all the available stashes at once:
  $ git stash clear

  Stash work on a separate branch:

  $ git stash branch
- **Git cherry pic**
  Apply the changes introduced by some existing commit:
  $ git cherry-pick

8. Merging

- **Git merge**
  Merge the branches:

$ git merge

Merge the specified commit to currently active branch:

$ git merge

- **Git rebase**

  Apply a sequence of commits from distinct branches into a final commit.

  $ git rebase

  Continue the rebasing process:

  $ git rebase -continue Abort the rebasing process:
  $ git rebase --skip

- **Git interactive rebase**

  Allow various operations like edit, rewrite, reorder, and more on existing commits.
  $ git rebase -i

9. Remote

- **Git remote**

  Check the configuration of the remote server:
  $ git remote -v
  Add a remote for the repository:

  $ git remote add Fetch the data from the remote server:

  $ git fetch

  Remove a remote connection from the repository:

  $ git remote rm
  Rename remote server:
  $ git remote rename

  Show additional information about a particular remote:

  $ git remote show
  Change remote:
  $ git remote set-url

- **Git origin master**

  Push data to the remote server:

  $ git push origin master Pull data from remote server:

  $ git pull origin master

10. Pushing Updates

- **Git push**

Transfer the commits from your local repository to a remote server. Push data to the remote server:

$ git push origin master Force push data:

$ git push -f

Delete a remote branch by push command:
$ git push origin -delete edited

11. Pulling updates

- **Git pull**
  Pull the data from the server:

  $ git pull origin master
  Pull a remote branch:
  $ git pull

- **Git fetch**
  Download branches and tags from one or more repositories. Fetch the remote repository:
  $ git fetch< repository Url> Fetch a specific branch:

  $ git fetch

  Fetch all the branches simultaneously:

  $ git fetch -all
  Synchronize the local repository:

  $ git fetch origin

12. Undo changes

- **Git revert**

  Undo the changes:

  $ git revert

  Revert a particular commit:
  $ git revert

- **Git reset**

  Reset the changes:

  $ git reset -hard

  $ git reset -soft:

  $ git reset --mixed

13. Removing files

- **Git rm**

Remove the files from the working tree and from the index:

$ git rm <file Name>

Remove files from the Git But keep the files in your local repository:

$ git rm –cached

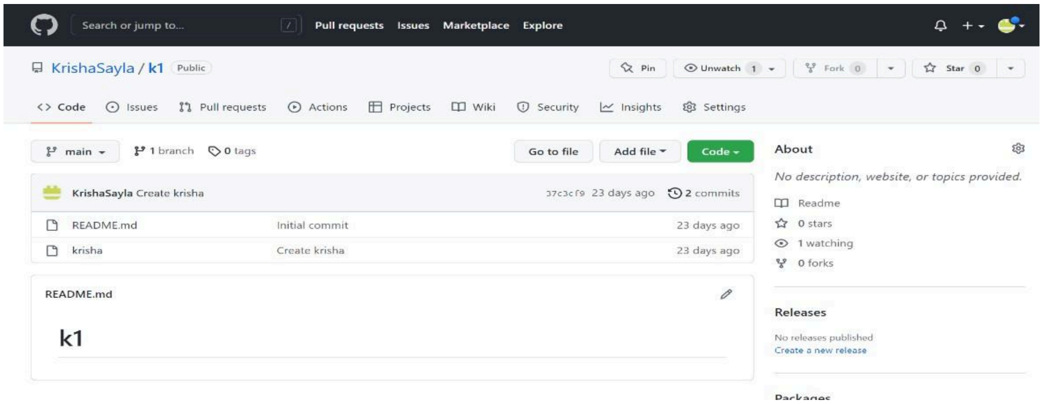
Output:

Installation of Git:

- Go to the website and download the 'git' file according to your system configuration.

Creation of GitHub account:

- Go to the website of GitHub and register with your email address for account creation.

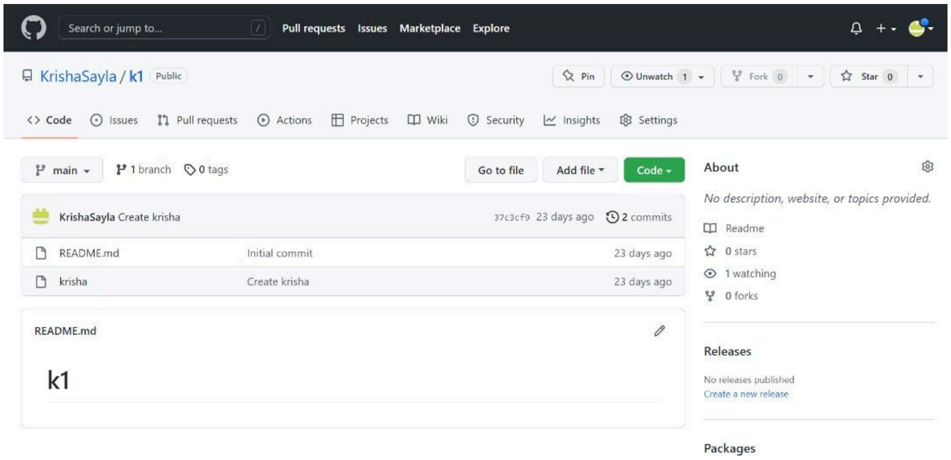


Operations on GIT:

```
Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config --global user.name krishasayla

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config --global user.email krisha.sayla@gmail.com

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config --global color.ui auto

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file <file>     use given config file
    --blob <blob-id>      read config from given blob object

Action
    --get                 get value: name [value-pattern]
    --get-all             get all values: key [value-pattern]
    --get-regexp          get values for regexp: name-regex [value-pattern]
    --get-urlmatch        get value specific for the URL: section[.var] URL
    --replace-all         replace all matching variables: name value [value-pattern]
    --add                 add a new variable: name value
    --unset               remove a variable: name [value-pattern]
    --unset-all           remove all matches: name [value-pattern]
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    --fixed-value         use string equality when comparing values to 'value-pattern'
    -e, --edit            open an editor
    --get-color           find the color configured: slot [default]
    --get-colorbool       find the color setting: slot [stdout-is-tty]

Type
    -t, --type <type>     value is given this type
    --bool                value is "true" or "false"
```

```
Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ mkdir exp3

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ cd exp3

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3
$ git init
Initialized empty Git repository in C:/Users/Krish/OneDrive/Desktop/exp3/.git/

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ touch demo.txt

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ git add .

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   demo.txt


Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ git commit -m "testcommit"
[master (root-commit) 803cf4e] testcommit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 demo.txt

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ git branch testbranch

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ git branch
* master
  testbranch

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (master)
$ git checkout testbranch
Switched to branch 'testbranch'

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (testbranch)
$ git branch
  master
* testbranch
```

```
Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (testbranch)
$ git clone https://github.com/KrishaSayla/k1.git
Cloning into 'k1'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop/exp3 (testbranch)
$
```



Conclusion:

Version Control System was understood in this experiment. Git was installed and Git bash was exercised. GitHub account was created and a repository was made. Difference between Git and GitHub was made clear.

Aim: To Perform various GIT operations on local and Remote repositories using GIT Cheat-Sheet.

Lab Outcome: To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub

Theory:

Git Cheat Sheet

1. Git configuration

- **Git config**

  Get and set configuration variables that control all facets of how Git looks and operates.
  Set the name:

  $ git config --global user.name "User name"

  Set the email:

  $ git config --global user.email "Krisha.sayla@gmail.com"

  Set the default editor:
  $ git config --global core.editor Vim

  Check the setting:

  $ git config -list
- **Git alias**
  Set up an alias for each command:
  $ git config --global alias.co checkout

  $ git config --global alias.br branch

  $ git config --global alias.ci commit

  $ git config --global alias.st status

2. Starting a project

- **Git init**

  Create a local repository:

  $ git init

- **Git clone**

  Make a local copy of the server repository.

  $ git clone

3. Local changes

- **Git add**

  Add a file to staging (Index) area:
  $ git add Filename

  Add all files of a repo to staging (Index) area:

  $ git add*
- **Git commit**
  Record or snapshots the file permanently in the version history **with a message**.

  $ git commit -m " Commit Message"

4. Track changes

- **Git diff**

  Track the changes that have not been staged:

  $ git diff

  Track the changes that have staged but not committed:

  $ git diff --staged

  Track the changes after committing a file:
  $ git diff HEAD
  Track the changes between two commits:

  $ git diff Git Diff Branches:

  $ git diff < branch 2>

- **Git status**
  Display the state of the working directory and the staging area.

  $ git status
- **Git show Shows objects:**
  $ git show

5. Commit History

- **Git log**

  Display the most recent commits and the status of the head:

$ git log
Display the output as one commit per line:

$ git log -oneline

Displays the files that have been modified:

$ git log -stat

Display the modified files with location:

$ git log -p

- **Git blame**

  Display the modification on each line of a file:

  $ git blame <file name>

6. Ignoring files

- **.gitignore**

  Specify intentionally untracked files that Git should ignore. Create .gitignore:

  $ touch .gitignore List the ignored files:
  $ git ls-files -i --exclude-standard

7. Branching

- **Git branch Create branch:**

  $ git branch List Branch:

  $ git branch --list Delete a Branch:

  $ git branch -d Delete a remote Branch:

  $ git push origin -delete Rename Branch:

  $ git branch -m

- **Git checkout**

  Switch between branches in a repository.
  Switch to a particular branch:
  $ git checkout

  Create a new branch and switch to it:
  $ git checkout -b Checkout a Remote branch:

  $ git checkout

- **Git stash**

  Switch branches without committing the current branch. Stash current work:

  $ git stash

  Saving stashes with a message:

  $ git stash save ""

  Check the stored stashes:

  $ git stash list

Re-apply the changes that you just stashed:

```
$ git stash apply
```

Track the stashes and their changes:

$ git stash show

Re-apply the previous commits:
$ git stash pop

Delete a most recent stash from the queue:

$ git stash drop

Delete all the available stashes at once:
$ git stash clear
Stash work on a separate branch:

$ git stash branch

- **Git cherry pic**

  Apply the changes introduced by some existing commit:

  $ git cherry-pick

8. Merging

- **Git merge**

  Merge the branches:

  $ git merge
  Merge the specified commit to currently active branch:

  $ git merge
- **Git rebase**
  Apply a sequence of commits from distinct branches into a final commit.

  $ git rebase
  Continue the rebasing process:

  $ git rebase -continue Abort the rebasing process:

  $ git rebase --skip
- **Git interactive rebase**
  Allow various operations like edit, rewrite, reorder, and more on existing commits.
  $ git rebase -i

9. Remote

- **Git remote**
  Check the configuration of the remote server:

  $ git remote -v

  Add a remote for the repository:

  $ git remote add Fetch the data from the remote server:

```
$ git fetch
```

Remove a remote connection from the repository:

$ git remote rm
Rename remote server:
$ git remote rename

Show additional information about a particular remote:

$ git remote show
Change remote:
$ git remote set-url

- **Git origin master**
  Push data to the remote server:

  $ git push origin master Pull data from remote server:

  $ git pull origin master

10. Pushing Updates

- **Git push**

  Transfer the commits from your local repository to a remote server. Push data to the remote server:
  $ git push origin master Force push data:

  $ git push -f
  Delete a remote branch by push command:

  $ git push origin -delete edited

11. Pulling updates

- **Git pull**

  Pull the data from the server:

  $ git pull origin master
  Pull a remote branch:
  $ git pull
- **Git fetch**

  Download branches and tags from one or more repositories. Fetch the remote repository:
  $ git fetch< repository Url> Fetch a specific branch:

  $ git fetch
  Fetch all the branches simultaneously:

  $ git fetch -all

  Synchronize the local repository:
  $ git fetch origin

12. Undo changes

- **Git revert**

  Undo the changes:

  $ git revert

  Revert a particular commit:

  $ git revert

- **Git reset**

  Reset the changes:

  $ git reset -hard

  $ git reset -soft:

  $ git reset --mixed

13. Removing files

- **Git rm**

  Remove the files from the working tree and from the index:

  $ git rm <file Name>

  Remove files from the Git But keep the files in your local repository:

  $ git rm –cached

OUTPUT:

```
Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config --global user.name krishasayla

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config --global user.email krisha.sayla@gmail.com

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config --global color.ui auto

Krish@DESKTOP-9LA87DS MINGW64 ~/OneDrive/Desktop
$ git config
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file <file>     use given config file
    --blob <blob-id>      read config from given blob object

Action
    --get                 get value: name [value-pattern]
    --get-all             get all values: key [value-pattern]
    --get-regexp          get values for regexp: name-regex [value-pattern]
    --get-urlmatch        get value specific for the URL: section[.var] URL
    --replace-all         replace all matching variables: name value [value-pattern]
    --add                 add a new variable: name value
    --unset               remove a variable: name [value-pattern]
    --unset-all           remove all matches: name [value-pattern]
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    --fixed-value         use string equality when comparing values to 'value-pattern'
    -e, --edit            open an editor
    --get-color           find the color configured: slot [default]
    --get-colorbool       find the color setting: slot [stdout-is-tty]

Type
    -t, --type <type>     value is given this type
    --bool                value is "true" or "false"
```

Conclusion:

Hence, I have learned and executed various git commands to perform task like creating directory, uploading data to directory, fetching data from directory et

Aim: To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers.

Lab Outcome No: 5.

Lab Outcome: To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.

Theory:

Docker architecture

- Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

The Docker daemon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

- The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

- A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

- When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

- When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

- An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.
- You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
- A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

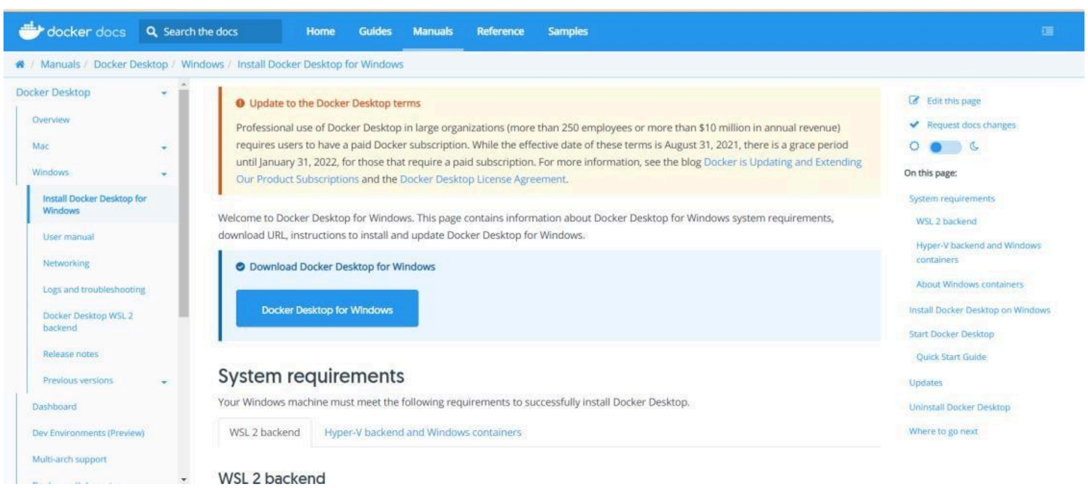
The life cycle of a container

If you working in a DevOps or SRE environment, it is very rare to find a person who has not heard of the name containers. In this article, we talk about the life cycle of a container. For this purpose, we will use docker for a container runtime. Let's have a look at the docker commands first.

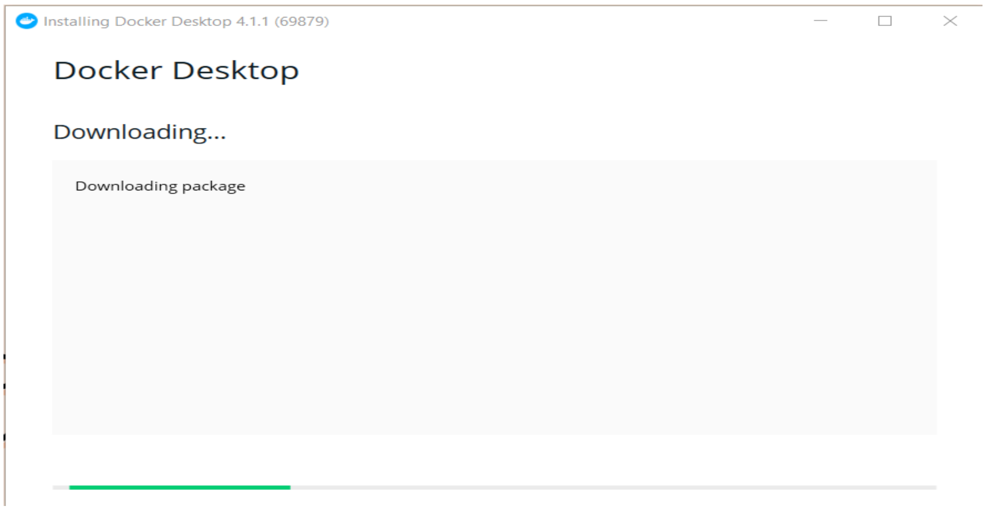
- docker build: This is used to build the docker image and then put it to the image registry.
- docker pull: This is used to pull the image build in the above portion from the registry.
- docker run: This will run the image as a docker container
- docker pause: It is used to pause the docker container.
- docker unpause: It is used to unpause the docker container.
- docker stop: Stops the docker container
- docker start: Starts back the docker container.
- docker kill: Kills the docker container.

Docker Installation:

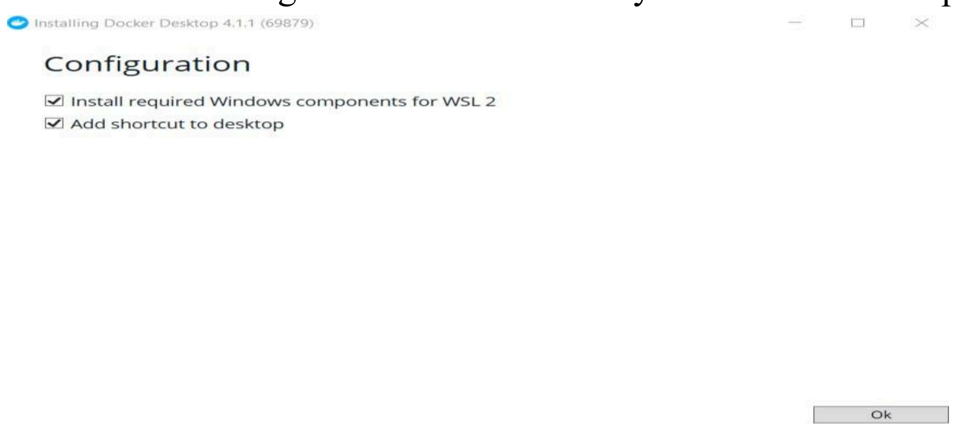
1. To install docker go to https://docs.docker.com/desktop/windows/install/ and click on docker desktop for windows. But before installing we need to make sure the system requirements are satisfied, once all the necessary changes is done start download the docker.



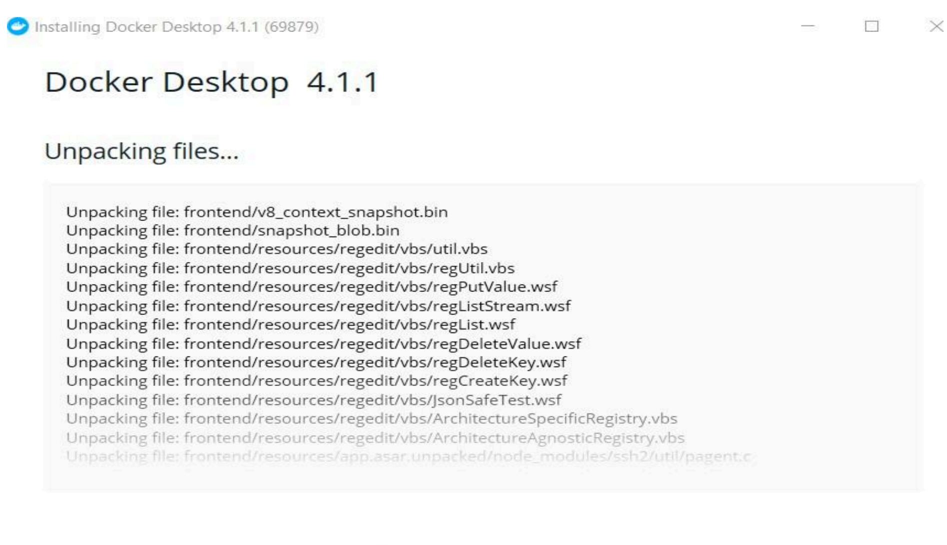
2. Once the docker is downloaded double click on the file and start installing



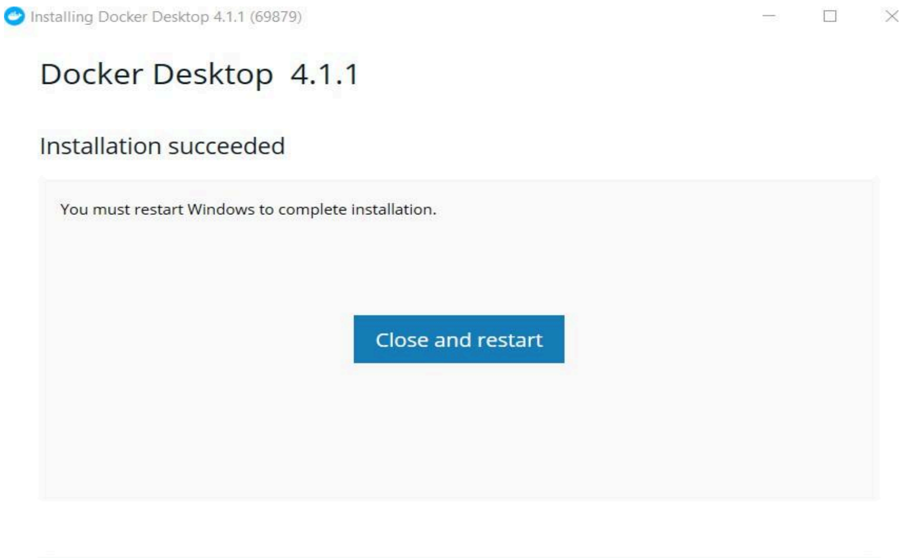
3. While downloading the docker make sure you select both the options and click ok



4. After this step the installation process begins.



5. Once the docker is installed click on close and restart the PC to update the configurations.



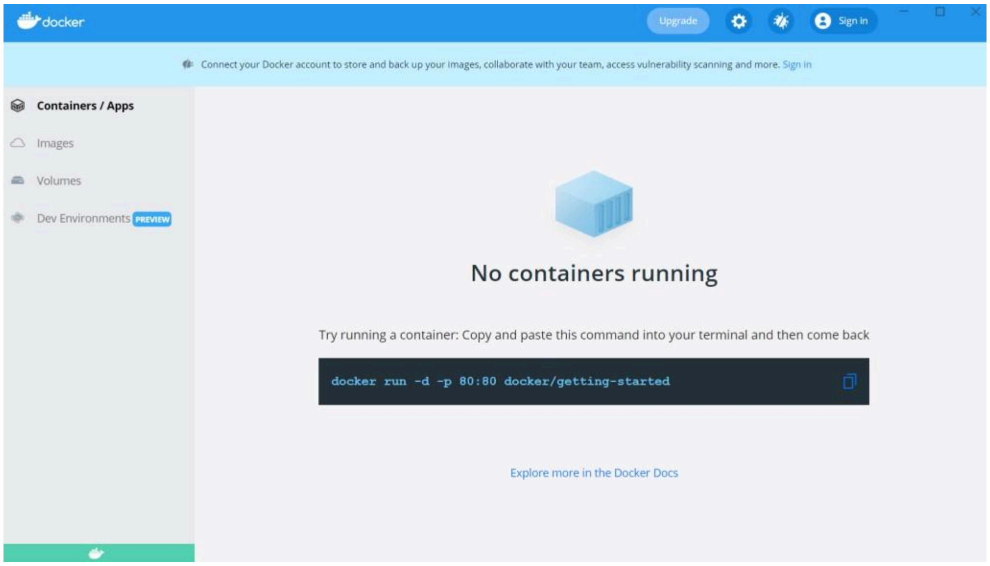
6. Once the system is started open cmd and type docker to check if docker is properly installed or not.

```
Command Prompt
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sadaf>docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default
                           "C:\\Users\\sadaf\\.docker")
  -c, --context string     Name of the context to use to connect to the
                           daemon (overrides DOCKER_HOST env var and
                           default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level
                           ("debug"|"info"|"warn"|"error"|"fatal")
                           (default "info")
```

7. Check docker version

```
C:\Users\sadaf>docker -v
Docker version 20.10.8, build 3967b7d
```

8. Once the docker is installed start the app and it will show as Docker engine running



- Docker Images Commands:

Description	Screenshots

docker ps: List containers
 	
docker images: List images
	`C:\Users\sadaf>docker ps`
	`CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES`
	
	`C:\Users\sadaf>docker images`
	`REPOSITORY   TAG   IMAGE ID   CREATED   SIZE`

docker pull: Pull an image or a repository from a registry	`C:\Users\sadaf>docker pull ubuntu`
	`Using default tag: latest`
	`latest: Pulling from library/ubuntu`
	`7b1a6ab2e44d: Pull complete`
	`Digest: sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322`
	`Status: Downloaded newer image for ubuntu:latest`
	`docker.io/library/ubuntu:latest`
	
	`C:\Users\sadaf>docker images`
	`REPOSITORY   TAG      IMAGE ID       CREATED       SIZE`
	`ubuntu       latest   ba6acccedd29   6 hours ago   72.8MB`
	
	`C:\Users\sadaf>docker pull ubuntu:18.04`
	`18.04: Pulling from library/ubuntu`
	`284055322776: Pull complete`
	`Digest: sha256:0fedbd5bd9fb72089c7bbca476949e10593cebed9b1fb9edf5b79dbbacddd7d6`
	`Status: Downloaded newer image for ubuntu:18.04`
	`docker.io/library/ubuntu:18.04`
	
	`C:\Users\sadaf>docker images`
	`REPOSITORY   TAG      IMAGE ID       CREATED       SIZE`
	`ubuntu       latest   ba6acccedd29   6 hours ago   72.8MB`
	`ubuntu       18.04    5a214d77f5d7   2 weeks ago   63.1MB`

<table>
<tr>
<td>-q: Only show image IDs

-f: Filter output based on conditions provided
-a: Show all images (default hides intermediate images)
--name: Assign a name to the container</td>
<td><pre>
C:\Users\sadaf>docker images -q
ba6acccedd29
5a214d77f5d7

C:\Users\sadaf>docker images -f "dangling=false"
REPOSITORY   TAG   IMAGE ID   CREATED   SIZE
ubuntu   latest   ba6acccedd29   6 hours ago   72.8MB
ubuntu   18.04   5a214d77f5d7   2 weeks ago   63.1MB

C:\Users\sadaf>docker images -f "dangling=true"
REPOSITORY   TAG   IMAGE ID   CREATED   SIZE

C:\Users\sadaf>docker images -f "dangling=false" -q
ba6acccedd29
5a214d77f5d7

C:\Users\sadaf>docker images -a
REPOSITORY   TAG   IMAGE ID   CREATED   SIZE
ubuntu   latest   ba6acccedd29   6 hours ago   72.8MB
ubuntu   18.04   5a214d77f5d7   2 weeks ago   63.1MB
</pre>
<pre>
C:\Users\sadaf>docker run ubuntu

C:\Users\sadaf>docker ps -a
CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES
fdb675092358   ubuntu   "bash"   30 seconds ago   Exited (0) 24 seconds ago   awesome_albatt
ani

C:\Users\sadaf>docker run --name MyUbuntu1 -it ubuntu bash
root@96659a3f2211:/#
root@96659a3f2211:/# ls -l
total 48
lrwxrwxrwx   1 root root   7 Oct 6 16:47 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 15 2020 boot
drwxr-xr-x   5 root root 360 Oct 16 06:35 dev
drwxr-xr-x   1 root root 4096 Oct 16 06:35 etc
drwxr-xr-x   2 root root 4096 Apr 15 2020 home
lrwxrwxrwx   1 root root   7 Oct 6 16:47 lib -> usr/lib
lrwxrwxrwx   1 root root   9 Oct 6 16:47 lib32 -> usr/lib32
lrwxrwxrwx   1 root root   9 Oct 6 16:47 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Oct 6 16:47 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Oct 6 16:47 media
drwxr-xr-x   2 root root 4096 Oct 6 16:47 mnt
drwxr-xr-x   2 root root 4096 Oct 6 16:47 opt
dr-xr-xr-x 200 root root   0 Oct 16 06:35 proc
drwx------   2 root root 4096 Oct 6 16:58 root
</pre></td>
</tr>
<tr>
<td>docker inspect: Return low-level information on Docker objects</td>
<td><pre>
C:\Users\sadaf>docker ps
CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES
96659a3f2211   ubuntu   "bash"   47 seconds ago   Up 43 seconds   MyUbuntu1

C:\Users\sadaf>docker inspect ubuntu
[
    {
        "Id": "sha256:ba6acccedd2923aee4c2acc6a23780b14ed4b8a5fa4e14e252a23b846df9b6c1",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2021-10-16T00:37:47.578710012Z",
        "Container": "249e88be79ad9986a479c71c15a056946ae26b0c54c1f634f115be6d5f9ba1c8",
        "ContainerConfig": {
            "Hostname": "249e88be79ad",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
</pre></td>
</tr>
</table>

rmi: Remove one or more images
stop: Stop one or more running containers

```
C:\Users\sadaf>docker rmi ubuntu
Error response from daemon: conflict: unable to remove repository reference "ubuntu" (must force)
ainer fdb675092358 is using its referenced image ba6acccedd29

C:\Users\sadaf>docker stop MyUbuntu1
MyUbuntu1

C:\Users\sadaf>docker rmi -f ubuntu
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322
Deleted: sha256:ba6acccedd2923aee4c2acc6a23780b14ed4b8a5fa4e14e252a23b846df9b6c1

C:\Users\sadaf>docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       18.04     5a214d77f5d7   2 weeks ago   63.1MB

C:\Users\sadaf>
```

- Docker Container Commands:

docker run: Run a command in a new container. The docker run command first creates a writeable container layer over the specified image, and then starts it using the specified command.

```
C:\Users\sadaf>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:37a0b92b08d4919615c3ee023f7ddb068d12b8387475d64c622ac30f45c29c51
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

Ps: List containers

-a: Show all containers (default shows just running)

```
C:\Users\sadaf>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\sadaf>docker ps -a
CONTAINER ID   IMAGE          COMMAND    CREATED              STATUS
NAMES
af07e803540e   hello-world    "/hello"   About a minute ago   Exited (0) About a minute ago
admiring_yonath
96659a3f2211   ba6acccedd29   "bash"     46 minutes ago       Exited (0) 43 minutes ago
MyUbuntu1
fdb675092358   ba6acccedd29   "bash"     49 minutes ago       Exited (0) 49 minutes ago
awesome_albattani
```

Start: Start one or more stopped containers
Stop: Stop one or more running containers

```
C:\Users\sadaf>docker ps
CONTAINER ID   IMAGE    COMMAND   CREATED         STATUS         PORTS     NAMES
76663ea7ce86   ubuntu   "bash"    3 minutes ago   Up 3 minutes             MyUbuntu2

C:\Users\sadaf>docker start MyUbuntu2
MyUbuntu2

C:\Users\sadaf>docker stop MyUbuntu2
MyUbuntu2
```

<table>
<tr>
<td>Pause: Pause all processes within one or more containers
Top: Display the running processes of a container</td>
<td>C:\Users\sadaf>docker start MyUbuntu2
MyUbuntu2

C:\Users\sadaf>docker pause MyUbuntu2
MyUbuntu2

C:\Users\sadaf>docker top MyUbuntu2
UID PID PPID C STIME
Y TIME CMD
root 1466 1444 0 07:28
00:00:00 bash</td>
</tr>
<tr>
<td>Stats: Display a live stream of container(s) resource usage statistics</td>
<td>C:\Users\sadaf>docker stats MyUbuntu2
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 836B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS
76663ea7ce86 MyUbuntu2 0.00% 812KiB / 6.105GiB 0.01% 906B / 0B 0B / 0B 1</td>
</tr>
<tr>
<td>Attach: Attach local standard input, output, and error streams to a running container</td>
<td>C:\Users\sadaf>docker attach MyUbuntu2
root@76663ea7ce86:/# ls -l
total 48
lrwxrwxrwx 1 root root 7 Oct 6 16:47 bin -> usr/bin
drwxr-xr-x 2 root root 4096 Apr 15 2020 boot
drwxr-xr-x 5 root root 360 Oct 16 07:28 dev
drwxr-xr-x 1 root root 4096 Oct 16 07:23 etc
drwxr-xr-x 2 root root 4096 Apr 15 2020 home
lrwxrwxrwx 1 root root 7 Oct 6 16:47 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Oct 6 16:47 lib32 -> usr/lib32
lrwxrwxrwx 1 root root 9 Oct 6 16:47 lib64 -> usr/lib64
lrwxrwxrwx 1 root root 10 Oct 6 16:47 libx32 -> usr/libx32
drwxr-xr-x 2 root root 4096 Oct 6 16:47 media
drwxr-xr-x 2 root root 4096 Oct 6 16:47 mnt
drwxr-xr-x 2 root root 4096 Oct 6 16:47 opt
dr-xr-xr-x 213 root root 0 Oct 16 07:28 proc
drwx------ 1 root root 4096 Oct 16 07:27 root
drwxr-xr-x 5 root root 4096 Oct 6 16:58 run
lrwxrwxrwx 1 root root 8 Oct 6 16:47 sbin -> usr/sbin
drwxr-xr-x 2 root root 4096 Oct 6 16:47 srv
dr-xr-xr-x 11 root root 0 Oct 16 07:28 sys
drwxrwxrwt 2 root root 4096 Oct 6 16:58 tmp
drwxr-xr-x 13 root root 4096 Oct 6 16:47 usr
drwxr-xr-x 11 root root 4096 Oct 6 16:58 var
root@76663ea7ce86:/# exit
exit</td>
</tr>
</table>

Rm: Remove one or more containers History: Show the history of an image/containers	See terminal output below

```
C:\Users\sadaf>docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                     PORTS     NAMES
76663ea7ce86   ubuntu        "bash"     8 minutes ago    Exited (0) 14 seconds ago            MyUbunt
u2
af07e803540e   hello-world   "/hello"   11 minutes ago   Exited (0) 11 minutes ago            admirin
g_yonath
96659a3f2211   ubuntu        "bash"     56 minutes ago   Exited (0) 52 minutes ago            MyUbunt
u1
fdb675092358   ubuntu        "bash"     59 minutes ago   Exited (0) 58 minutes ago            awesome
_albattani

C:\Users\sadaf>docker rm MyUbuntu1
MyUbuntu1

C:\Users\sadaf>docker history ubuntu
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
ba6acccedd29   7 hours ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      7 hours ago   /bin/sh -c #(nop) ADD file:5d68d27cc15a80653…   72.8MB
```

Conclusion:

Learned the concept of docker. Installed docker in our system. Executed docker images commands such as pull, inspect, basic images commands, rmi etc. Created docker containers and execute different containers commands.

7

Aim: To learn Docker file instructions, build an image for a sample web application using Docker file.

Lab Outcome no: ITL503.5

Lab Outcome: To understand concept of containerization and Analyze the Containerization of OS images and deployment of applications over Docker.

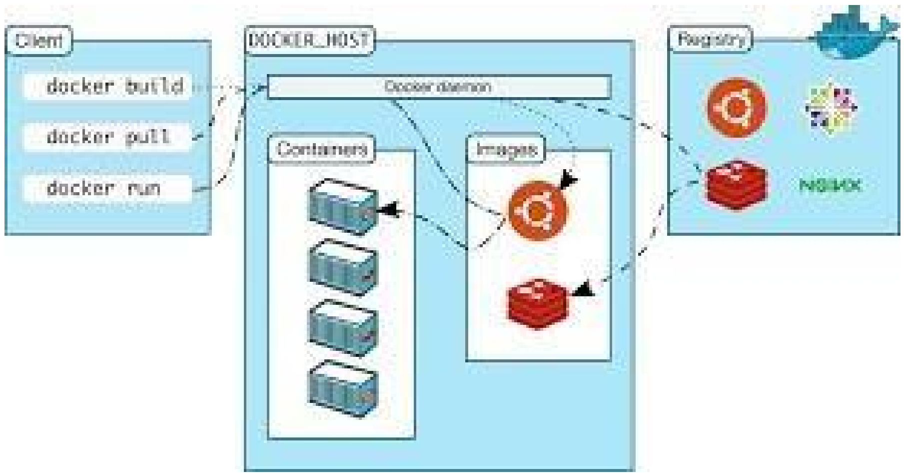
Theory:

Docker Architecture:

What Is Container (Docker)? Containers are a software package into a logical boxwith everything that the application needsto run. That includes the operating system, application code, runtime, system tools, systemlibraries, and etc. Dockercontainers are built off Docker images. Since images are read-only, Docker adds aread-write file system over the read-only file system of the image to create a container. Containers are compared with virtual machines (VMs). VMs are the guest operating system such as Linux or Windows runs on top of a host operatingsystem with virtualized access to the underlying hardware. Containers allow you to package your application together with libraries and other dependencies, providing isolated environments for running your software services.

What is Docker? Docker is an open-source platform based on Linux containers fordeveloping and running applications inside containers. Docker is used to deploy many containers simultaneously on a given host. Containers are very fast and lightweight because they don't need the extra load of a hypervisor as they run
directly within the host machine's kernel. Docker Architecture and Components Docker uses a client-server architecture. The docker client talks to the Docker daemon, which used to building, running, and distributing the Docker containers.

The Docker client and daemon communicate using a REST API, over UNIX sockets,or a network interface.



There are five major components in the Docker architecture:

a) Docker Daemon listens to Docker API requests and manages Docker objectssuch as images, containers, networks and volumes.

b) Docker Clients: With the help of Docker Clients, users can interact with Docker. Docker client provides a command-line interface (CLI) that allowsusers to run, and stop application commands to a Docker daemon.

c) Docker Host provides a complete environment to execute and run applications. It comprises of the Docker daemon, Images, Containers,Networks, and Storage.

d) Docker Registry stores Docker images. Docker Hub is a public registry thatanyone can use, and Docker is configured to use images on Docker Hub bydefault. You can run your own registry on it.

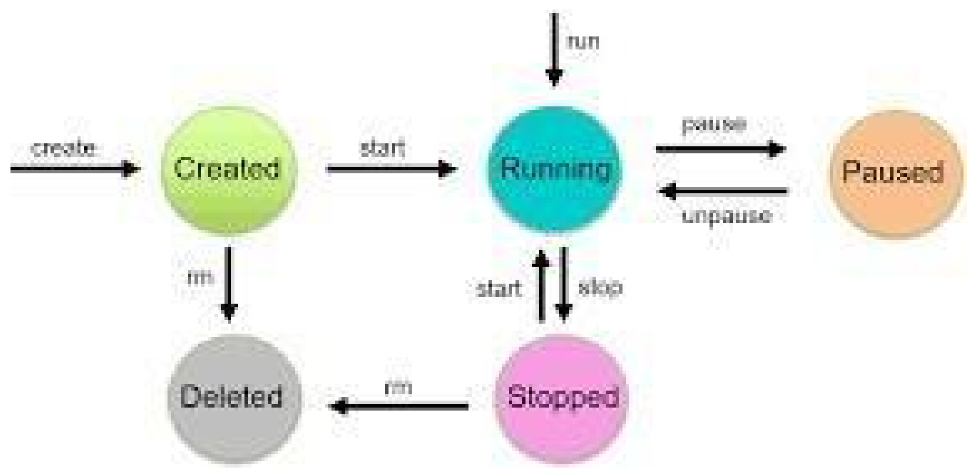
e) Docker Images are read-only templates that you build from a set of instructions written in Dockerfile. Images define both what you want your packaged application and its dependencies to look like what processes to runwhen it's launched.

- Docker Engine Components Docker Engine is the layer on which Dockerruns. It is installed on the host machine. It's a lightweight runtime and tooling that manages containers, images, builds, and more.

There are three components in the Docker Engine:
a) Server: It is the docker daemon called dockerd. It can create and managedocker images, i.e, Containers, networks.
b) Rest API: It is used to instruct docker daemon what to do.

c) Command Line Interface (CLI): It is a client that is used to enter dockercommands

There are different stages when we create a container which is known as Lifecycleof container i.e create, run, pause, delete & stopped.



- The first phase is the created state. Further, the container moves into therunning state while we use the Docker run command.

- We can stop or pause the container, using Docker stop/pause command. And, toput a container back from a stopped state to a running state, we use the Docker run command.
- We can delete a running or stopped container, using Docker rm command.

- <u>Step-By-Step Docker Installation on Windows :</u>

1. Go to the website https://docs.docker.com/docker-for-windows/install/ and download the docker file. Note: A 64-bit processor and 4GB system RAM are thehardware prerequisites required to successfully run Docker on Windows 10.
2. Then, double-click on the Docker Desktop Installer.exe to run the installer. Note: Suppose the installer (Docker Desktop Installer.exe) is not downloaded; youcan get it from Docker Hub and run it whenever required.
3. Once you start the installation process, always enable Hyper-V WindowsFeature on the Configuration page.
4. Then, follow the installation process to allow the installer and wait till theprocess is done.
5. After completion of the installation process, click Close and restart.

Conclusion:

In this experiment, we have learned and studied about Dockerfile. A DockerFile isa text document that contains all the commands a user could call on the command line to assemble images.