Git Branching & Merging

AWS EC2 Auto Scaling, AWS RDS Auto Backup & Restore

AWS Regions & Availability Zones

Docker – Installation, Docker Images, Docker Desktop, Docker Hub, Creating Docker Images

AWS – Amazon Web Service

Amazon.com – Register as a user, select the product, add it to the cart and check out, pay the bill and it’ ll be delivered to your door step.

Amazon.com --- It’s a website which sells many products in online mode.

Amazon.com --- uses lot of micro-services.

AWS – Cloud Service Provider – Many Services over the internet – Cloud Technologies

1. Virtual Machines (EC2 – Elastic Cloud Compute, ECS, EKS - Elastic Container/Kubernetes Service)
2. DB Services (RDS, DynamoDB,…)
3. Storage Services (S3, S3 Bucket ….)
4. Network Services (IGW – Ineternet Gate Way, Security Groups, VPN – Virtual Private Networks)
5. Users & Group policy Services (IAM – Id and Access Management Service..)
6. AR & VR Services
7. AI & ML Services
8. Robotics Services
9. Crypto Services

So many data centers – which is located in various parts of the world.

24 Regions & 77 Availability Zones

24 Regions (US Region/APAC/Africa/Europe

Auto Scaling an Instance [Automatically increasing/decreasing the number of running instances]

When the Load/Demand increases the number of running instances increases.

When the Load/Demand decreases the number of running instances will be reduced.

Git Branching

Git Repo – It’s the virtual place where we can store all the source code and track all the changes to it.

Github repositories will have a default branch (main/master) -- creating a cloud repository

**…or create a new repository on the command line**

echo "# branchandmerge" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/syskantechnosoft/branchandmerge.git

git push -u origin main

**…or push an existing repository from the command line**

git remote add origin https://github.com/syskantechnosoft/branchandmerge.git

git branch -M main

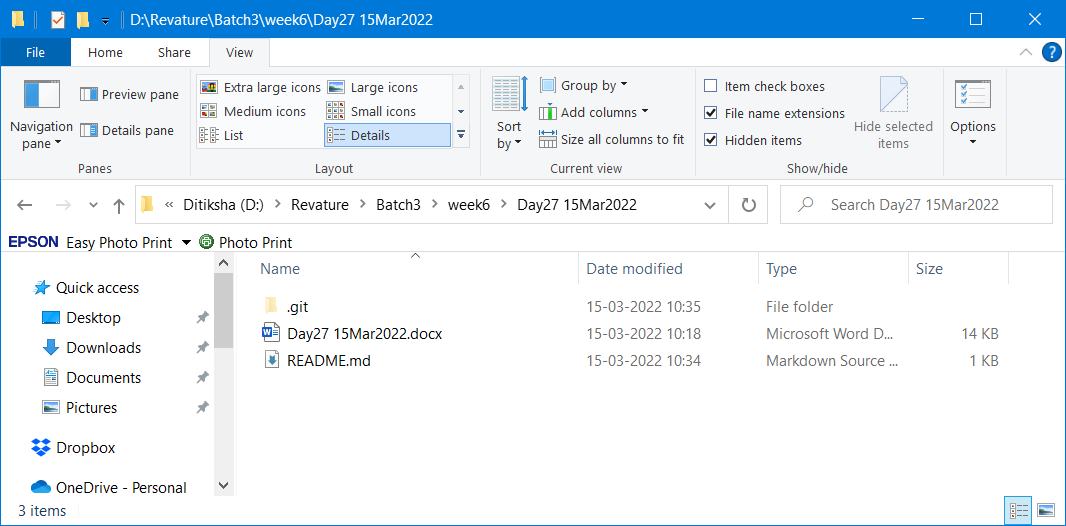
git push -u origin main

**…or import code from another repository**

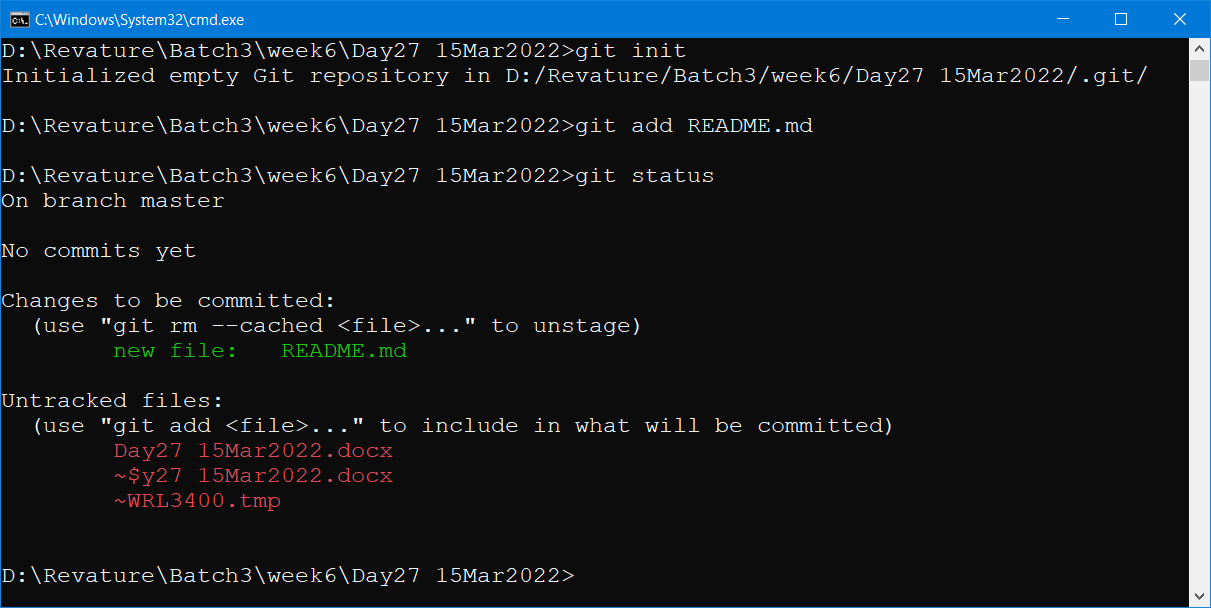
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](https://github.com/syskantechnosoft/branchandmerge/import)

.git (Hidden) Folder will be created in the Git Local Repo (A folder in the file system of the desktop/laptop)

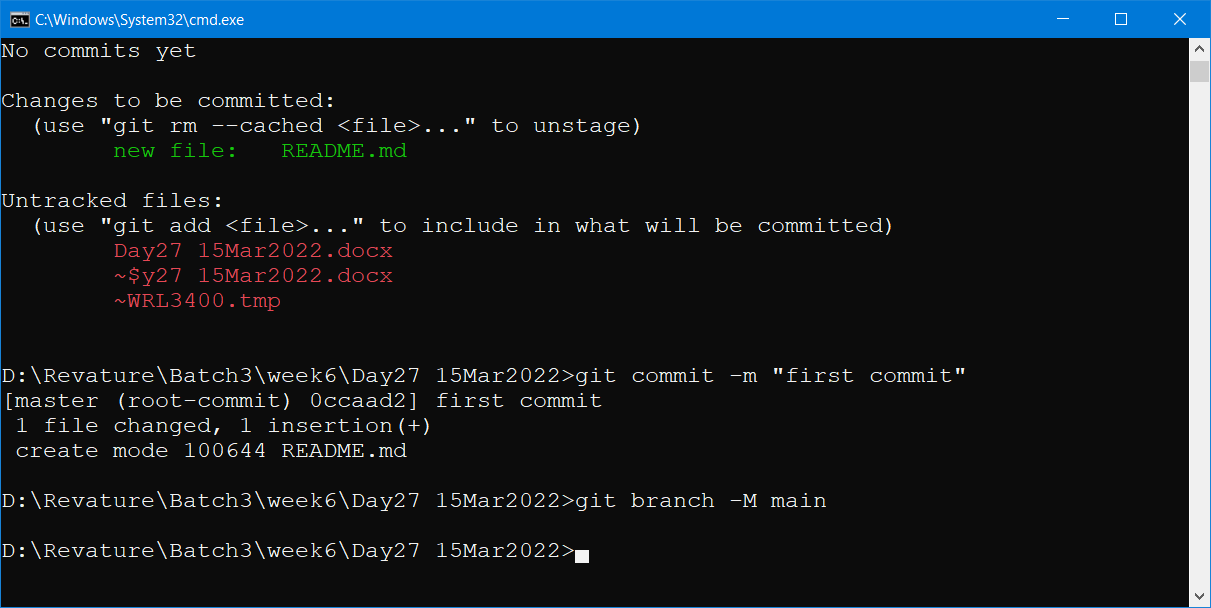


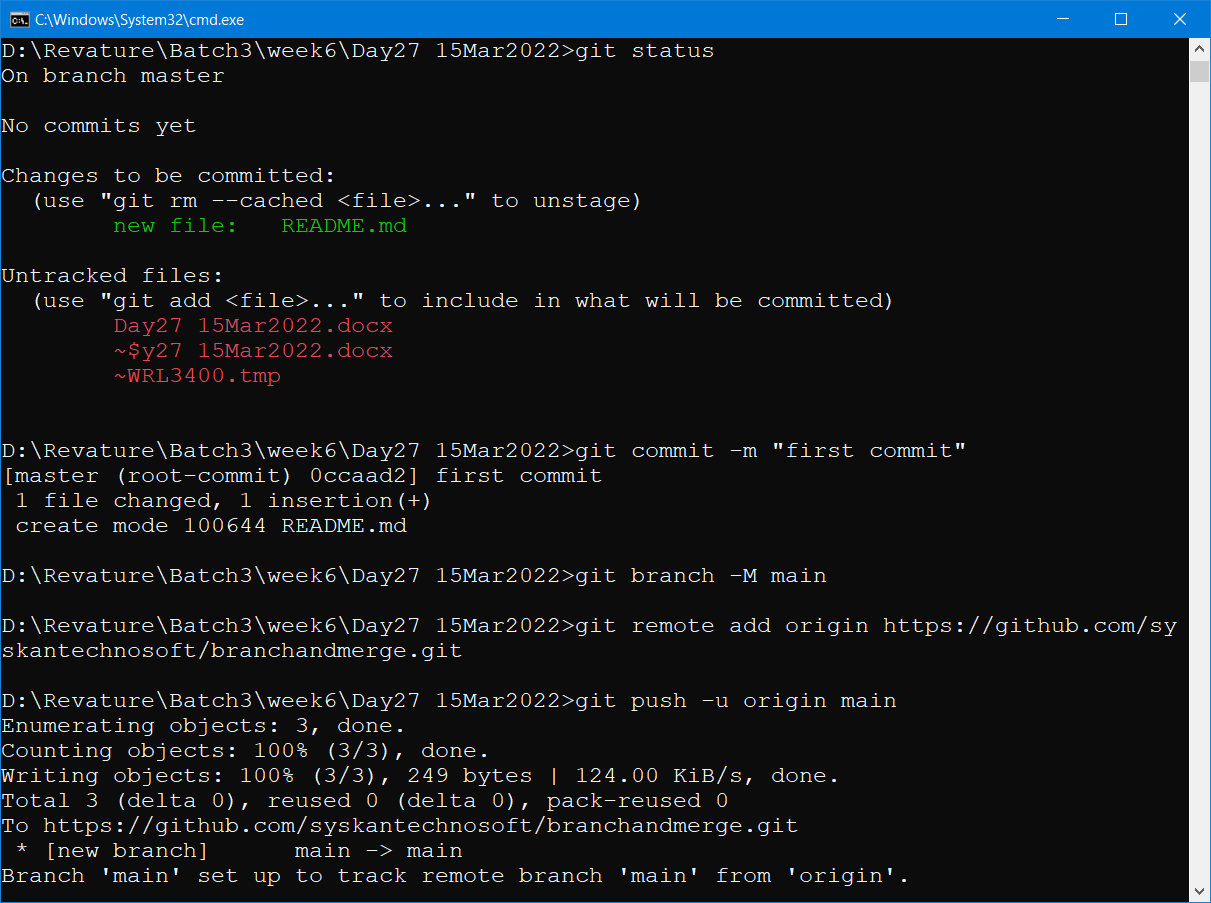


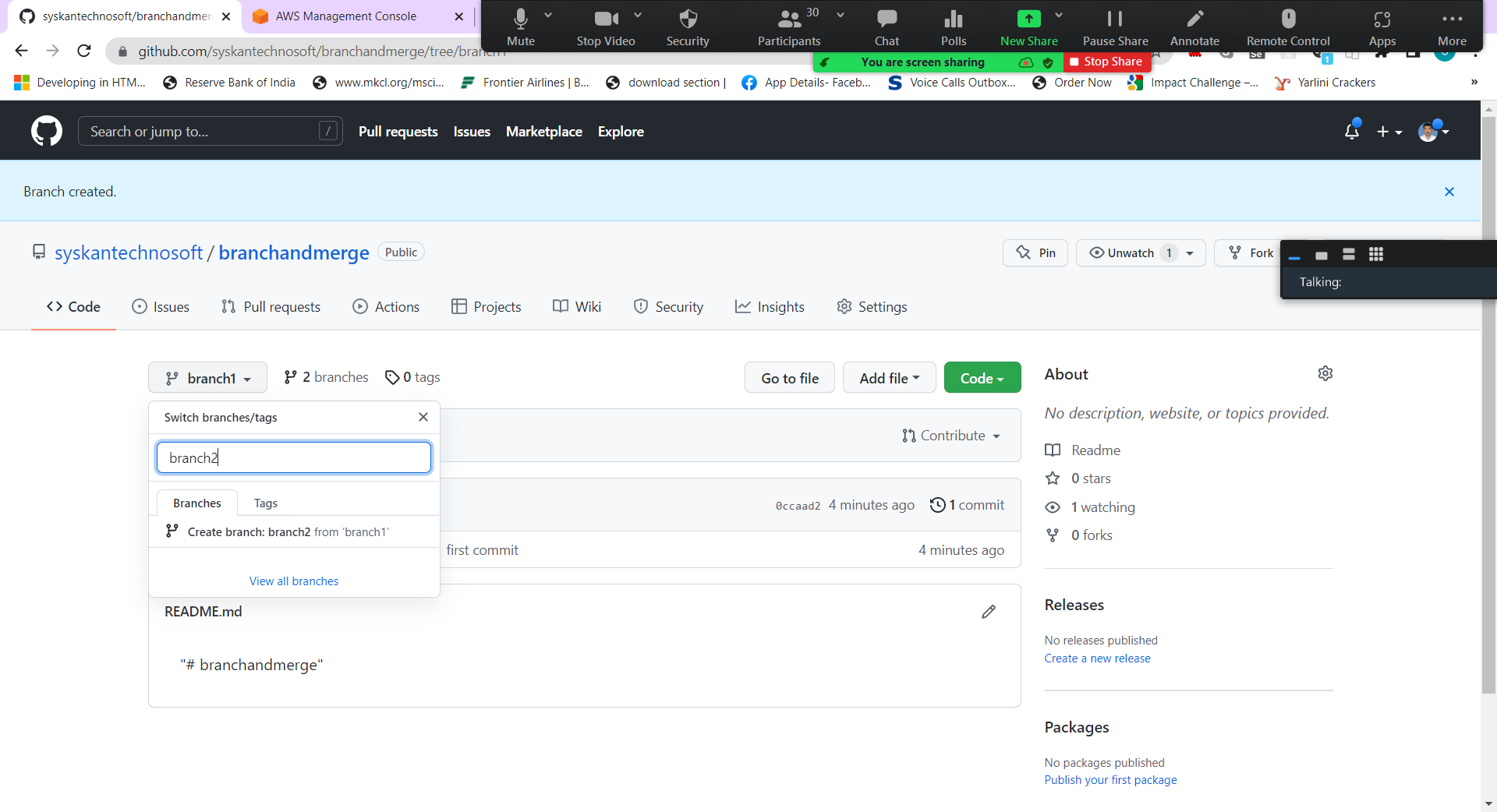


After the ADD command the tracked resource go to the staging area (This is temporary area where the resources will be saved for a shorter duration till executing the commit command)

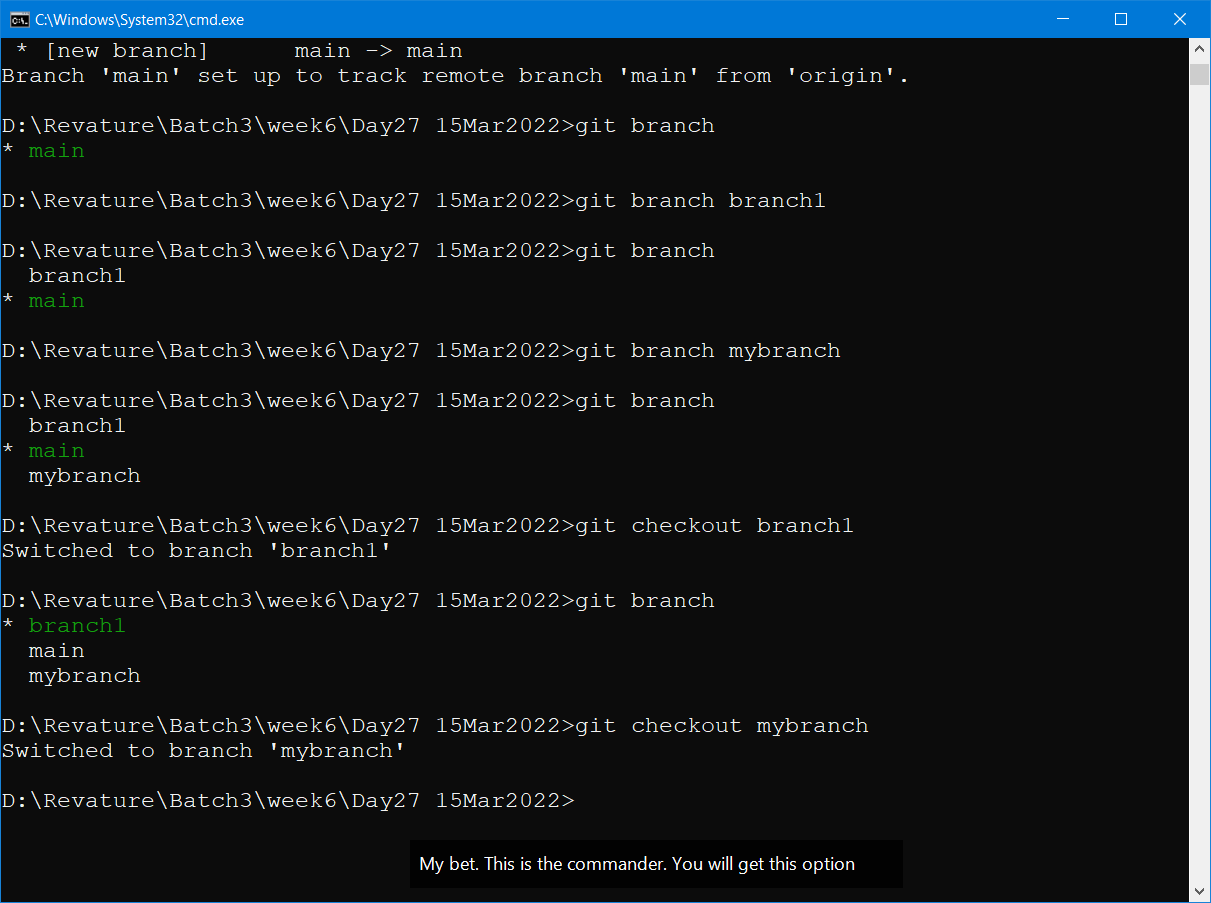
Git add . (dot represents all the files & folders)





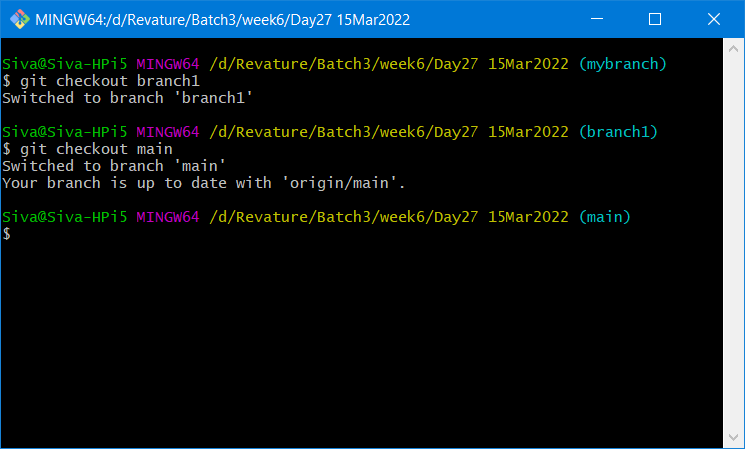


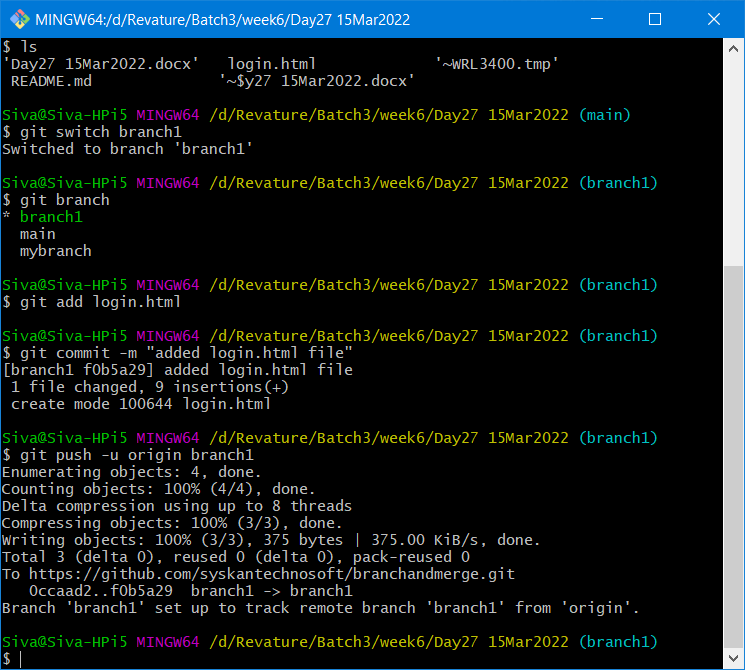




Git branch <existing/new\_branch\_name> --- To create a new branch or to check the current branch details

Git checkout <name\_of\_branch> -- To move the HEAD pointer to a different branch





Merge conflict – login.html

Line 5 added < marquee> tag -- first merge – no issues

Line 5 added <h3> tag -- second merge – merge conflict – Resolving the merge conflict

**Note : Empty files/folders/branches will not be tracked by git.**

**You can’t add empty file/ folder or a branch to git.**

Docker is a One of the DevOps Tool

DevOps – Development & Operations

Development – Creating a project using any programming Lang (JAVA Enterprise Project – Spring Boot based microservice, Core JAVA stand-alone application, Simple Web Application using JSP & Servlets) – offshore team

Operations – They will deploy and maintain the project in the server. – onsite team.

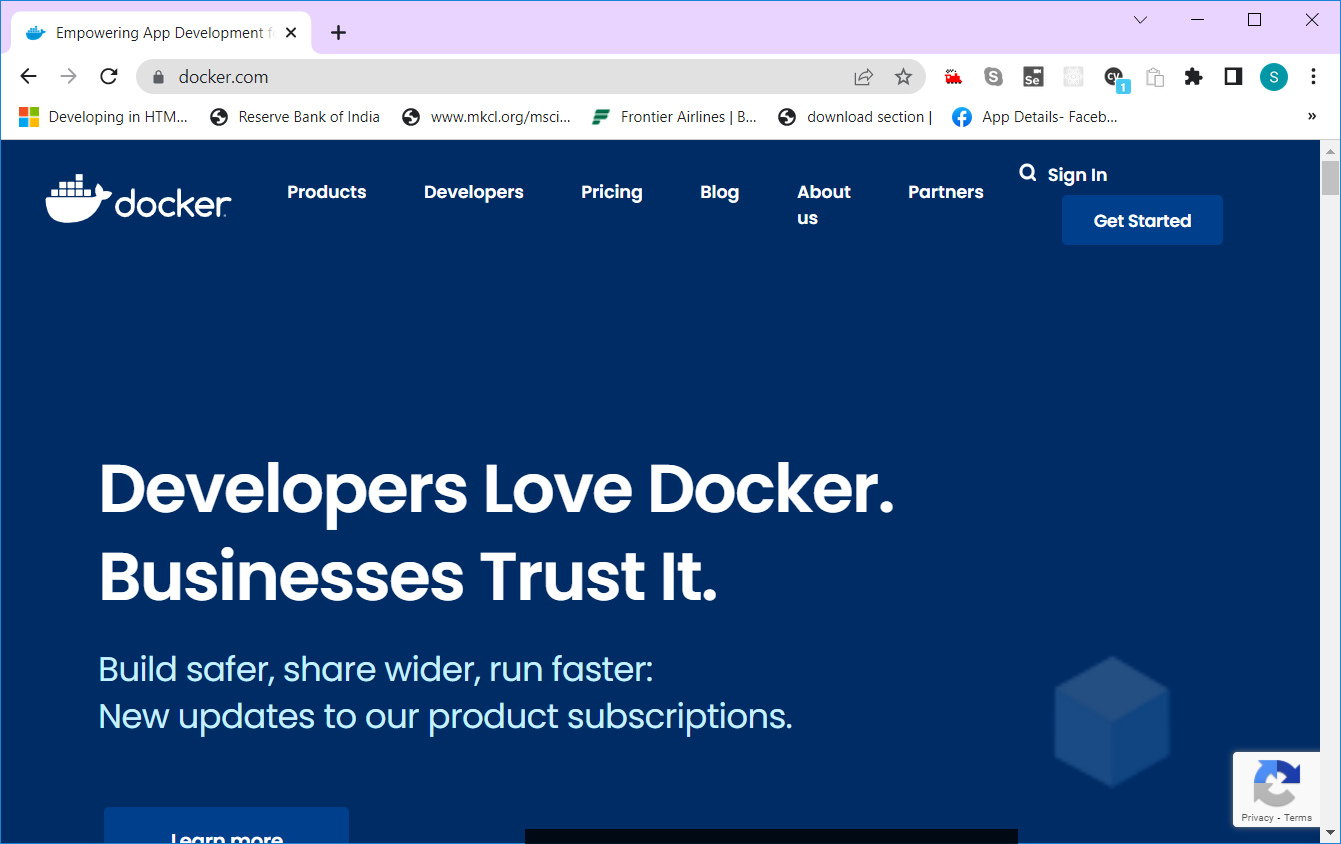
FSD – Full Stack Developer (Development & Operations)

CI/CD – Continuous Integration and Continuous Delivery

DevOps Tools

1. Docker – Containerization Tool
2. Jenkins – CI/CD Build management Tool
3. Kubernetes – Container Orchestration Tool

Docker official site : <https://www.docker.com/>



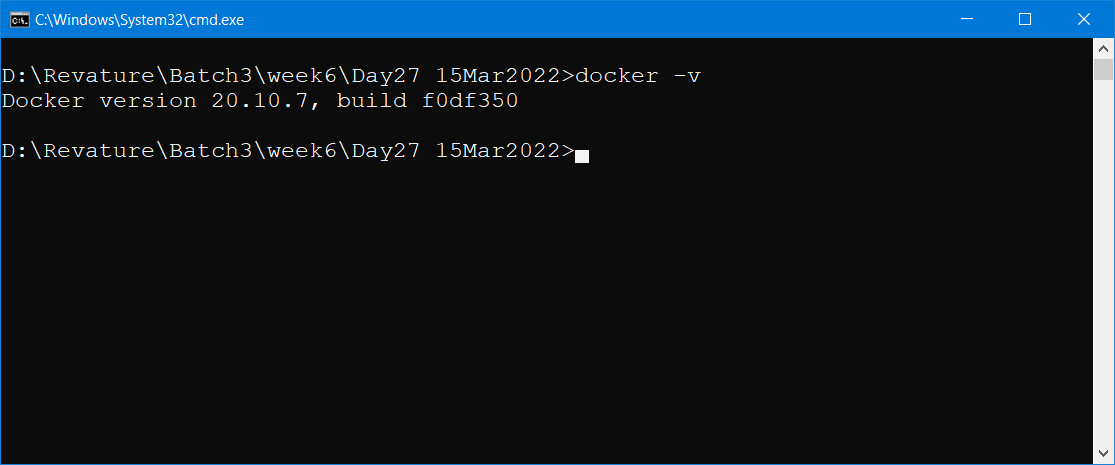
Docker Desktop Tool : -- <https://www.docker.com/products/docker-desktop>

Docker Desktop – Is a desktop client. It will help to interact with docker cloud.

GitBash/GitGUI == Docker Desktop (Stand – alone desktop GUI based application)

Github == Docker Cloud --- <https://hub.docker.com/>

Docker can also be accessed using command prompt also



Note : All the docker commands starts with “docker”

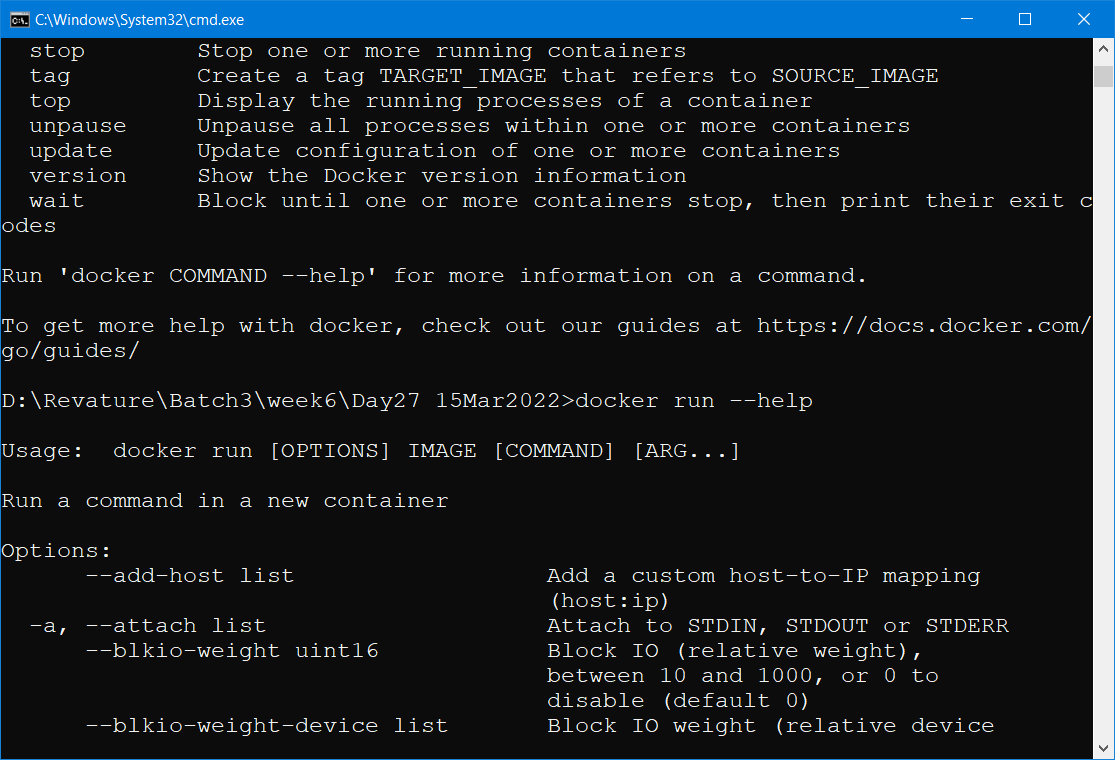
Docker -v --- displays the version of docker installed in your system.

Docker –help ---- displays all the available docker commands

Docker <command> --help – displays more details about that particular command

Docker Architecture

1. Docker Engine / Docker Daemon /Docker Server – This will run in the background always
2. Docker Desktop /Docker Client – GUI Stand-alone Client which helps to access docker engine
3. REST API – Docker commands will create REST api calls to interact with docker Engine.



Docker is DevOps tool which is used to containerize an application and also to manage those containers.

Containerization – Packing application along with all the required libraries & dependencies

myProject1.jar -🡪 Containerize -🡪 myProject1 image (docker image) – single file

all the images (official & 3rd party) can be downloaded from <https://hub.docker.com>

Images available for Operating System , Databases, Application Softwares,

Docker eliminated uses of Virtual Machines.

1. Dual Booting (Install Windows & Linux along side -20 GB to 40 GB) – TWO different partitions
2. VM Ware (Install Linux (other OS) inside Windows (another OS) ) – virtualization
3. Dockerizing -- No need to install, just download the docker image and run it.

<https://phoenixnap.com/kb/mysql-docker-container>

Image – A Single file which contains all the code along with required libraries and dependencies.

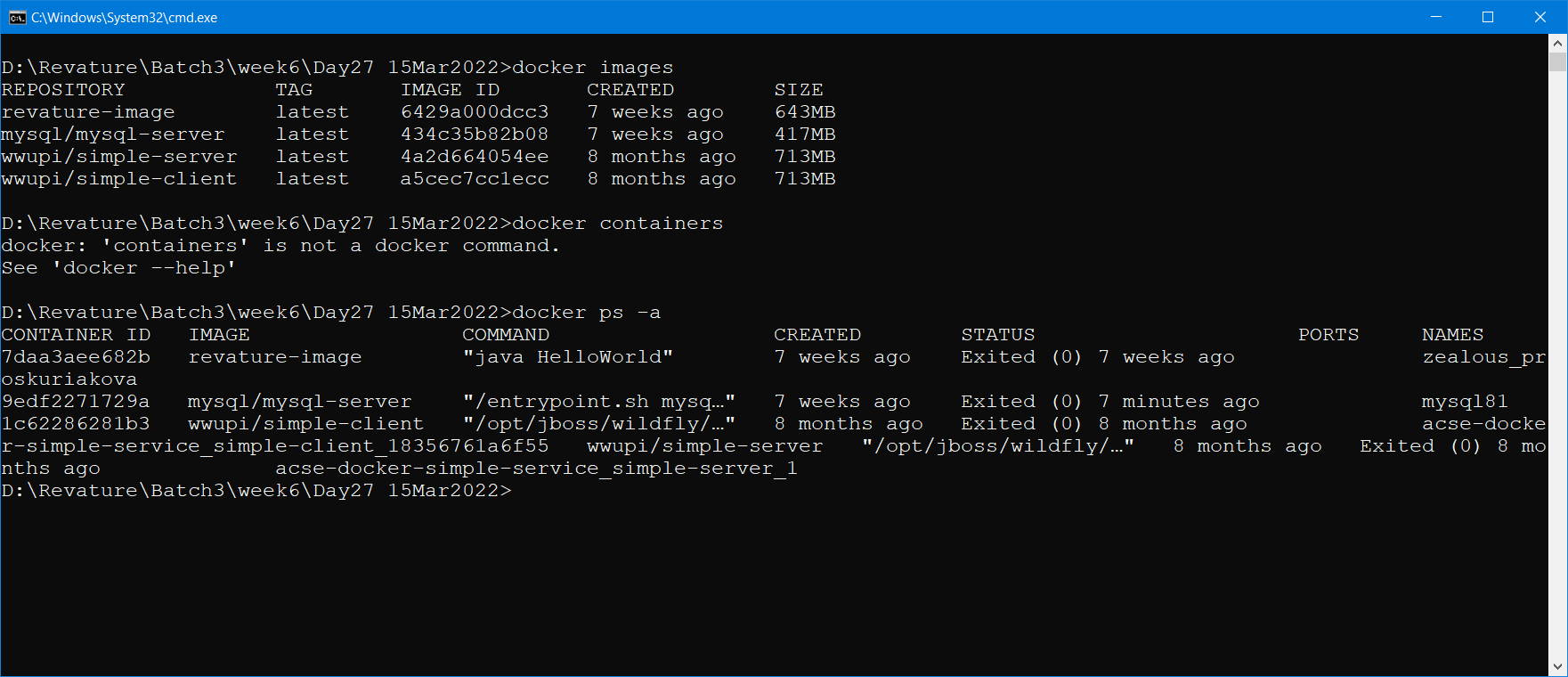
Each image will have a name and a unique Id.

Using a single image, we can create many containers.

Container – Is a running version of an image.

Container also will have a name, unique id, and port number.

Host Port (underlying OS port) : Container Port



Class --- Object (Class is a blue print – where as Object is a instance of a class)

A single class can be used to create many objects.

Similarly A single image can be used to create many containers.

Image is a blue-print and container is a instance of a image.

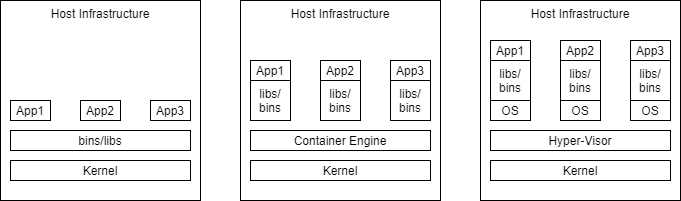
Image file will not have any extension.

Container file also will not have any extension.

Both images and containers can be managed (create/delete/update/run) with the help of docker desktop/ docker cli

Dockerfile, Docker compose, docker swarm



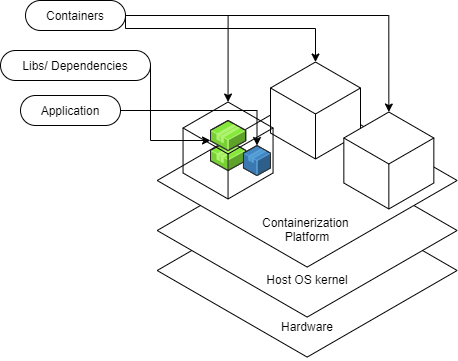


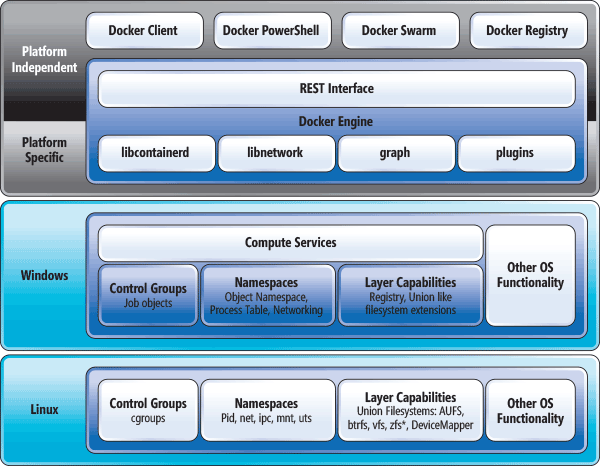


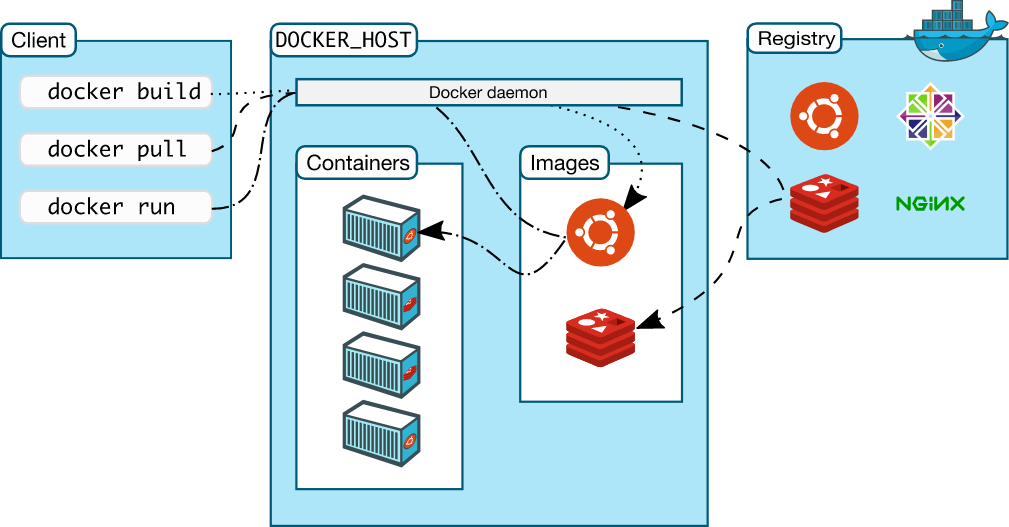
Containerization -- A **container** is a mechanism for packaging an application(or limited number of processes) with its dependencies so that it runs in its own isolated sandbox.

To Run a JAVA application

1. .class file
2. JRE to call the JVM
3. All the dependencies like jar files, configuration details and application.properties file, xml file
4. Command to execute the code (javac HelloWorld)







Docker client – GUI Client (Docker Desktop) & CLI Client (Docker CLI)

Docker Server – Docker Engine/Docker Daemon

Docker Registry – <https://hub.docker.com>

### States of a container:

* created
* restarting
* running
* paused
* exited
* dead

Dockerfile is a file without any extension that will be used to create a docker image. It will have all the dependencies and instruction to create a docker image.

## Dockerfile Commands

FROM image name

* specifies the parent image from which the new image should be based.
* forms the first layer of this new image.
* might say FROM imagename AS temp-name if we want to have use multi-stage builds
  + This will essentially create our final image by building on the previous images that we set up in the same dockerfile.
* i.e. If we wanted to start with a ubuntu OS base (where we would take advantage of multi-stage builds) we might have:

FROM ubuntu AS ubuntu-with-java-example

RUN

* There are two forms of the command.
  + RUN <command>
    - Runs the command in a shell by default (the particular shell depends on whether the parent image defines a linux or windows container)
  + RUN ["executable", "param1", "param2" ]
    - Runs the command in the executable form without using a shell
* RUN will be used to set up your image- the state of the image after each run command is committed- forming a new layer
* i.e. If we wanted to install some programs as part of the image, you might have:

RUN apt-get update

RUN apt-get -y dist-upgrade

RUN apt-get -y install default-jdk

ADD <src> <destination>

* adds files from build context or url to the image
* copy is preferred over add
* can perform auto-extraction into the image from a local tar file (this differs from copy and is one of the few instances add is preferred) -i.e. The following command would copy HelloWorld.java into a file of the same name in the container's working directory.

ADD HelloWorld.java HelloWorld.java

COPY <src> <destination>

* adds files from the build context to the image -i.e. The following command would copy HelloWorld.java into a file of the same name in the container's working directory.

COPY HelloWorld.java HelloWorld.java

EXPOSE

* outlines the ports that the are being listened on by processes in the container-- i.e. it suggests what ports to bind to host ports when running the image -The following example would inform the host to bind to port 80 in the container

EXPOSE 80

VOLUME ["/nameofdir"]

* creates a mount point in the image and thus container with a particular name- it indicates that the files in this directory will be shared with the resources outside of the container
* it indicates what directory to connect a volume to when running the docker container -i.e. The following suggests that we should connect a volume to the data directory in the container when we spin it up.

VOLUME ["/data"]

WORKDIR <nameofdirectory>

* sets the working directory in the image and eventual container of commands that follow. i.e.

WORKDIR /example

RUN mkdir a

* Would make a directory a inside of the example folder. example/a

CMD

* used to run the app, processes etc. needed inside of your container
* only the last CMD will run when the built image is launched as a container
* CMD ["executable","param1","param2"] (exec form, this is the preferred form)
  + invokes the command with out a shell
* CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
  + the ENTRYPOINT instruction must be specified if you use the default format, it's another way you can specify the first commands to be run upon spinning up a
* CMD command param1 param2 (shell form)
  + invokes the command inside of a shell in /bin/sh -c
* i.e. The last line of a docker file might be a command running a java program:

CMD ["java", "HiWorld"]