Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group

Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Q - 1) What is Docker, and how does it differ from a virtual machine?

Docker is an open-source platform that enables developers to automate the deployment of applications inside lightweight, portable containers. Containers are isolated from one another and include everything needed to run the application, such as code, runtime, libraries, and settings.

Difference from Virtual Machine (VM):

- Docker Containers: Share the host operating system's kernel, making them faster and more efficient in terms of resource usage.
- Virtual Machines: Require a full guest operating system with each VM, which consumes more resources and has a slower startup time compared to containers.

Q - 2) Explain the architecture of Docker. What are its main components?

The architecture of Docker is based on a client-server model. The main components of Docker include:

- Docker Client: The user interface to interact with Docker. Commands like `docker build` or `docker run` are executed by the client.
- Docker Daemon (dockerd): The background service that builds, runs, and manages Docker containers. It listens to requests from the Docker client.
- Docker Images: The blueprint or template for creating containers. It is a read-only file that contains the application code, runtime, libraries, and dependencies.
- Docker Containers: The running instances of Docker images. Containers are the standardized unit in which the application service resides.
- Docker Registry (Docker Hub): The repository for storing Docker images. Docker Hub is the default public registry where you can push and pull images.

Q - 3) What is a Docker image, and how is it different from a Docker container?

- Docker Image: A read-only template that contains the instructions to create a Docker container. It includes all the dependencies, application code, libraries, and configurations required to run the application.
- Docker Container: A runnable instance of a Docker image. While the image is a static file, the container is a dynamic, live environment where the application is executed.

Difference: An image is like a blueprint, while a container is a running instance created from that blueprint.

Q - 4) What is a Dockerfile? How do you create one?

- Dockerfile: A text file that contains a series of instructions on how to build a Docker image. It defines what goes into the image, such as the base image, application dependencies, environment variables, and commands to execute.
- How to Create: You create a Dockerfile using a text editor, typically with the filename `Dockerfile`. It includes commands like `FROM`, `RUN`, `COPY`, `CMD`, and `ENTRYPOINT` to specify the instructions.

Q - 5) How do you create a Docker image from a Dockerfile?

To create a Docker image from a Dockerfile:

1. Navigate to the directory containing the Dockerfile.
2. Run the command:

```
docker build -t image-name:tag.
```

   - `-t` flag assigns a name and tag to the image.
   - `.` refers to the current directory where the Dockerfile is located.

Q - 6) What are Docker containers, and how do they work?

Docker containers are isolated environments created from Docker images that package all the necessary software to run applications. They are lightweight and share the host system's kernel but run independently, with their own processes, file systems, and network interfaces. Containers can be easily started, stopped, and scaled up or down, making them ideal for microservices and scalable deployments.

Q - 7) How do you list all running Docker containers?

To list all running Docker containers, use the following command:

```
docker ps
```

- This command displays the container ID, image name, command, creation time, status, ports, and container names.

To list all containers (including stopped ones), use:

```
docker ps -a
```

Q - 8) How do you stop, start, or remove a Docker container?

- Stop a running container:

```
docker stop container-id/container-name
```

- Start a stopped container:

```
docker start container-id/container-name
```

- Remove a container:

```
docker rm container-id/container-name
```

You need to stop the container before removing it.

Q - 9) What are Docker volumes, and why are they used?

Docker volumes are a way to store data used by Docker containers outside the container's writable layer. They are used to persist data beyond the lifecycle of a container. Volumes are stored on the host file system and allow data to be shared between containers or backed up easily.

Benefits of using volumes:

- Decouple container data from the container itself.
- Share data between multiple containers.
- Persist data even when containers are removed.

Q - 10) How can you persist data in Docker containers?

There are three main ways to persist data in Docker containers:

1. Docker Volumes: The most preferred way to manage persistent data. Volumes are created using the command:

   ```
   docker volume create volume-name
   ```

   and mounted to containers with the `-v` or `--mount` option.

2. Bind Mounts: Maps a file or directory on the host machine to a location inside the container. It is specified during container creation with:

   ```
   docker run -v /host-path:/container-path image-name
   ```

3. Tmpfs Mounts: Store data in memory and are used mainly for temporary data that doesn't need to persist beyond the lifecycle of the container.

Q - 11) What is the difference between `COPY` and `ADD` in a Dockerfile?

Both `COPY` and `ADD` are used in Dockerfiles to copy files and directories from the host machine to the Docker image. However, they have some key differences:

- `COPY`: It is a straightforward command that copies files or directories from the host to the image. It is used only for basic file copying.

```
COPY source_path /destination_path
```

- `ADD`: In addition to copying files and directories, it can also handle the following:
  - It supports the extraction of tar files.
  - It allows copying files from remote URLs.

```
ADD source_path /destination_path
```

Best Practice: Use `COPY` unless you specifically need the extra functionality provided by `ADD`.

Q - 12) How do you optimize Docker images to reduce their size?

To optimize Docker images and reduce their size, you can follow these best practices:

- Use a smaller base image: Choose lightweight base images like `alpine` to reduce the image size.
- Minimize the number of layers: Combine multiple commands into a single `RUN` instruction to reduce the number of layers in the image.

```
RUN apt-get update && apt-get install -y \
    package1 \
    package2 && \
    apt-get clean
```

- Remove unnecessary files: Delete temporary files, caches, and logs in the same `RUN` command.
- Multi-stage builds: Use multi-stage builds to create optimized production images by copying only the necessary files to the final image.
- Use `.dockerignore` file: Add unnecessary files and directories to `.dockerignore` to prevent them from being included in the build context.

Q - 13) What is Docker Compose, and why would you use it?

Docker Compose is a tool that allows you to define and manage multi-container Docker applications using a YAML file (`docker-compose.yml`). It simplifies the process of running multiple containers that work together, such as a web server, database, and caching layer.

Why use Docker Compose?

- Easy orchestration of multiple services.
- Manage the entire lifecycle of an application with a single command (`docker-compose up`).
- Supports service scaling and updates.
- Portable configurations for different environments.

Q - 14) How do you define services in a Docker Compose file?

Services in a Docker Compose file are defined under the `services` section of the YAML configuration. Each service represents a container that runs a specific part of the application. Here is an example of a `docker-compose.yml` file with two services:

```yaml
version: '3.8'


services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html


  database:
    image: mysql:5.7
    environment:
```

```
      MYSQL_ROOT_PASSWORD: examplepassword
```

This file defines two services: `web` (an Nginx server) and `database` (a MySQL server).

Q - 15) How do you scale services using Docker Compose?

Docker Compose allows you to scale services using the `--scale` flag with the `docker-compose up` command. This flag creates multiple instances of a service, useful for load balancing. For example:

```
docker-compose up --scale web=3
```

This command will scale the `web` service to run three instances. Scaling is generally configured for services that can run multiple instances without interfering with each other.

Q - 16) What are the different networking modes in Docker?

Docker supports several networking modes for containers:

- Bridge (default mode): Containers use a private internal network on the Docker host, allowing communication with each other.
- Host mode: The container shares the network stack with the Docker host, meaning there is no network isolation.
- None mode: Disables all networking for the container, providing complete isolation.
- Container mode: The container shares the network namespace of another container, effectively using the same network interface.
- Overlay network: Used in multi-host Docker networks, typically with Docker Swarm, to allow containers to communicate across different Docker hosts.

Q - 17) How do you link containers using Docker networks?

In Docker, you can link containers by creating a user-defined bridge network and then attaching the containers to it. This setup allows containers to communicate with each other by their names.

Steps to link containers:

1. Create a network:

   ```
   docker network create my-network
   ```

2. Start the containers on the network:

```
docker run -d --name container1 --network my-network image1

docker run -d --name container2 --network my-network image2
```

3. Containers `container1` and `container2` can now communicate using their names.

Q - 18) What is the difference between `CMD` and `ENTRYPOINT` in a Dockerfile?

Both `CMD` and `ENTRYPOINT` are used to specify the default commands to run when a container starts. However, they differ in behavior:

- CMD: Provides default arguments to the entry point of the container, but it can be overridden by passing command-line arguments when starting the container.

```
CMD ["nginx", "-g", "daemon off;"]
```

- ENTRYPOINT: Specifies the command that always runs when the container starts, and any arguments passed at runtime are appended to the entry point.

```
ENTRYPOINT ["nginx"]
```

Best Practice: Use `ENTRYPOINT` for the main application logic and `CMD` for default arguments.

Q - 19) How can you pass environment variables to Docker containers?

Environment variables can be passed to Docker containers in several ways:

1. Using the `-e` flag:

```
docker run -e MY_ENV_VAR=myvalue image-name
```

2. Using an environment file:

   Create a file named `.env` with the variables:

```
MY_ENV_VAR=myvalue
```

   Then use the `--env-file` flag:

```
docker run --env-file .env image-name
```

3. In the Docker Compose file:

```
environment:
```

```
      - MY_ENV_VAR=myvalue
```

Q - 20) What are Docker registries, and what is Docker Hub?

- Docker Registries: A Docker registry is a storage and distribution system for Docker images. Registries allow users to store and share images. You can have public or private registries for image storage.
- Docker Hub: Docker Hub is the default and most popular public Docker registry. It contains a vast library of official and community-contributed Docker images. Users can push their custom images to Docker Hub and pull images from it for use in their Docker environments.

Q - 21) What is Docker Swarm, and how does it differ from Kubernetes?

- Docker Swarm: Docker Swarm is Dockerâ€™s native clustering and orchestration tool. It allows you to manage and deploy a group of Docker nodes as a single logical unit. Swarm mode enables you to create a swarm of Docker nodes and deploy services to the swarm.

Differences between Docker Swarm and Kubernetes:

- Complexity: Kubernetes is more complex and has a steeper learning curve compared to Docker Swarm, which is easier to set up and use.
- Scalability: Kubernetes provides more advanced features for large-scale deployments, while Docker Swarm is more suitable for smaller, simpler setups.
- Ecosystem Integration: Kubernetes has a broader ecosystem and is more widely adopted, with many tools and extensions available. Docker Swarm is more tightly integrated with the Docker ecosystem.
- Networking: Kubernetes provides a more sophisticated networking model with network policies and service meshes, whereas Docker Swarm has simpler networking options.

Q - 22) How do you set up a Docker Swarm cluster?

To set up a Docker Swarm cluster, follow these steps:

1. Initialize the Swarm manager: Run this command on the master node:

   ```
   docker swarm init
   ```

   The command will return a join token.

2. Add worker nodes to the swarm: Run the following command on each worker node using the token provided by the master node:

```
docker swarm join --token <SWARM-TOKEN> <MANAGER-IP>:2377
```

3. Verify the cluster setup: Run this command on the master node to view the nodes in the cluster:

```
docker node ls
```

Q - 23) What are Docker secrets, and how are they used for secure data handling?

- Docker Secrets: Docker secrets are a secure way to manage sensitive data such as passwords, API keys, certificates, or any other confidential information in Docker Swarm. Secrets are encrypted and only accessible to the services that need them.

How to use Docker secrets:

1. Create a secret:

```
echo "mysecretdata" | docker secret create my_secret -
```

2. Use the secret in a service:

   When deploying a service, specify the secret:

```
docker service create --name my_service --secret my_secret
my_image
```

3. Access the secret: Inside the container, secrets are accessible as files in the `/run/secrets/` directory.

Q - 24) How do you handle logging in Docker containers?

Docker supports different logging drivers to handle container logs, such as:

- json-file: Stores logs in JSON format on the host machine.
- syslog: Sends logs to the syslog daemon.
- fluentd: Uses the Fluentd log collector.
- gelf: Logs to a Graylog Extended Log Format (GELF) server.
- awslogs: Sends logs to Amazon CloudWatch.

To specify a logging driver:

```
docker run --log-driver json-file image-name
```

For more extensive logging, you can use external tools like the ELK (Elasticsearch, Logstash, Kibana) stack or Splunk.

Q - 25) What is a multi-stage build in Docker, and how does it help in building optimized images?

- Multi-stage Build: A multi-stage build allows you to use multiple `FROM` statements in a Dockerfile to create intermediate images, but only the final image contains the necessary files for production.

Benefits:

- Reduces image size by including only the files required for the final production environment.
- Improves security by removing unnecessary tools and dependencies from the final image.

Example:

```
# First stage

FROM golang:alpine AS builder

WORKDIR /app

COPY . .

RUN go build -o myapp


# Second stage

FROM alpine:latest

COPY --from=builder /app/myapp /myapp

CMD ["/myapp"]
```


Q - 26) How can you monitor Docker containers?

Docker containers can be monitored using the following tools and methods:

- Docker CLI commands: Use commands like `docker stats` to monitor real-time metrics, including CPU, memory, and network usage.
- Monitoring tools: Use tools like Prometheus, Grafana, cAdvisor, Datadog, and Nagios to collect metrics and visualize container performance.

- Docker logging: Utilize the logging drivers and external log management tools to analyze logs and troubleshoot issues.

Q - 27) How do you perform health checks on Docker containers?

Health checks in Docker are used to determine if a container is running as expected. You can define a health check in the Dockerfile or in the Docker Compose file.

Dockerfile example:

```
HEALTHCHECK --interval=30s --timeout=5s --retries=3 CMD curl -f
http://localhost:80 || exit 1
```

- This command checks if the web server on port 80 is running. If the check fails three times, the container is marked as unhealthy.

Q - 28) What are the best practices for securing Docker containers?

Best practices for securing Docker containers include:

- Use official images: Always use trusted and verified images from Docker Hub or other registries.
- Limit container privileges: Run containers as non-root users to reduce security risks.
- Set resource limits: Limit CPU and memory usage to prevent resource exhaustion.
- Update regularly: Keep Docker and container images updated to mitigate vulnerabilities.
- Use Docker secrets: Manage sensitive information securely using Docker secrets.
- Network security: Isolate container networks and limit access using firewalls.

Q - 29) How do you handle networking issues between Docker containers?

To handle networking issues between Docker containers, consider the following steps:

- Check network connectivity: Use `docker network ls` to view available networks and ensure containers are attached to the same network.
- DNS resolution: Use container names for inter-container communication as Docker's built-in DNS resolves container names to their IP addresses.
- Inspect network settings: Use `docker network inspect` to review network configurations and troubleshoot connectivity issues.
- Firewall rules: Verify firewall settings on the host machine that could be blocking traffic between containers.

Q - 30) What are the main differences between Docker and Podman?

- Architecture: Podman does not require a central daemon like Docker does (Docker Daemon). Instead, Podman runs as a single process for each container.
- Rootless containers: Podman supports running containers as a non-root user out of the box, enhancing security.
- Compatibility: Podman is designed to be compatible with Docker CLI commands, making it easy for Docker users to transition.
- Security: Since Podman runs rootless, it offers better security by avoiding running containers with elevated privileges.
- Kubernetes integration: Docker requires third-party tools like `dockershim` to integrate with Kubernetes, whereas Podman can natively generate Kubernetes YAML files.

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project