What are Virtual Threads?

- **Virtual Threads** are lightweight threads introduced in **Project Loom** (Java 19 preview → stable in Java 21).
- They allow **millions of concurrent tasks** without the heavy memory and scheduling cost of traditional OS threads.
- Think of them as **"threads that don't block the OS"**, managed by the JVM instead of the OS.

Traditional Threads vs Virtual Threads

Feature	Traditional Threads	Virtual Threads
Creation cost	High (memory + OS)	Very low (JVM managed)
Scalability	Hundreds to thousands	Millions
Blocking I/O	Blocks OS thread	Doesn't block OS thread; other virtual threads continue
Ideal use	CPU-bound tasks	I/O-bound or high concurrency tasks

---

2️⃣ Creating a Virtual Thread

```
public class VirtualThreadDemo {
```

```
    public static void main(String[] args) throws InterruptedException
{


        Runnable task = () -> {

            System.out.println(Thread.currentThread() + " is
running");

            try {

                Thread.sleep(1000); // simulate work

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        };


        // Traditional thread

        Thread t1 = new Thread(task);

        t1.start();

        t1.join();


        // Virtual thread

        Thread vt = Thread.ofVirtual().start(task);

        vt.join();

    }

}
```

- `Thread.ofVirtual().start(task)` → creates a **lightweight virtual thread**.
- You can create **thousands or millions** of virtual threads without crashing your JVM.

3️⃣ Virtual Thread Executor

For many tasks, you can use **Executors** with virtual threads:

```
import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;



public class VirtualThreadExecutorDemo {

    public static void main(String[] args) throws InterruptedException
{

        try (ExecutorService executor =
Executors.newVirtualThreadPerTaskExecutor()) {



            for (int i = 0; i < 10; i++) {

                int id = i;

                executor.submit(() -> {

                    System.out.println("Task " + id + " running on " +
Thread.currentThread());

                    try { Thread.sleep(500); } catch
(InterruptedException e) {}

                });

            }
```

```
        } // executor automatically shuts down
    }
}
```

- `Executors.newVirtualThreadPerTaskExecutor()` → creates an executor where **each task runs on its own virtual thread**.
- Ideal for **highly concurrent I/O-bound workloads**, like web servers or REST APIs.

4️⃣ Key Benefits

1. **Massive concurrency** — millions of threads without memory overhead.
2. **Simpler programming model** — no need for reactive libraries just to avoid blocking threads.
3. **Seamless integration** — virtual threads can replace existing thread-based code.
4. **Better CPU utilization** — JVM can schedule tasks efficiently.