WebFlux vs Virtual Threads Cheat Sheet

Cheat Sheet: WebFlux vs Virtual Threads (Spring MVC)

Situation	Recommendation
Simple CRUD APIs	Spring MVC + Virtual Threads
Heavy Database interactions (JDBC)	Spring MVC + Virtual Threads
Real-time systems (Websockets, Streaming)	Spring WebFlux
Handling 100k+ concurrent users	Spring WebFlux
Team prefers easy imperative programming	Spring MVC + Virtual Threads
Full reactive stack needed (Kafka, R2DBC, WebClient)	Spring WebFlux
Existing codebase is blocking	MVC + Virtual Threads (no major refactor)
Need backpressure control	WebFlux

How to Decide in 30 Seconds

Is the app mostly API/DB CRUD + REST APIs?

-> Use MVC + Virtual Threads (simpler, stable)

Is the app real-time, stream-heavy, 100k concurrent users?

-> Go with WebFlux (full reactive power)

Do you need non-blocking across DB, messaging, networking?

-> WebFlux + R2DBC + Reactive Kafka

Is your team more comfortable with classic Java coding?

-> MVC + Virtual Threads

Are you migrating old blocking code?

-> MVC + Virtual Threads

Real-world Project Structures

Spring MVC + Virtual Threads (Simple, Clean)

```
/src/main/java/com/example/app
    controller/UserController.java
    service/UserService.java
    repository/UserRepository.java
    config/VirtualThreadConfig.java
    AppApplication.java
```

Spring WebFlux (Full Reactive)

```
/src/main/java/com/example/app
    handler/UserHandler.java
    router/UserRouter.java
    service/UserService.java
    repository/UserRepository.java (Reactive)
    config/WebFluxConfig.java
    AppApplication.java
```