Micronaut - Practice Sheet

10 Q&A on AOT DI, data, reactive, GraalVM, and security

Q1. What is Micronaut and why is its dependency injection compile-time?

A: Micronaut uses annotation processors at build time to generate injection code. Unlike Spring's runtime reflection, there is no classpath scanning on startup. This means zero reflection overhead, instant startup, and low memory ideal for FaaS and containers.

Q2. How does Micronaut's AOP work without runtime proxies?

A: Micronaut generates bytecode for interceptor chains at compile time using ASM. @Around advice is woven into a generated subclass. Because no JDK dynamic proxy or CGLIB is involved, there is no class hierarchy restriction and overhead is minimal.

Q3. How do you define a REST controller in Micronaut?

A: @Controller('/users') marks the class. @Get('/{id}'), @Post, @Put, @Delete annotate methods. @PathVariable, @QueryValue, and @Body inject URL parts and request data. Micronaut validates @Body objects with Jakarta Bean Validation automatically.

Q4. What is Micronaut Data and what makes it unique?

A: Micronaut Data generates repository query implementations at compile time using the method name or @Query annotation, producing raw SQL/JPQL. There is no runtime proxy or reflection, making it significantly faster than Spring Data's runtime approach.

Q5. How does Micronaut simplify GraalVM native image creation?

A: Micronaut generates GraalVM reflect-config.json, resource-config.json, and proxy-config.json metadata files automatically during compilation. This eliminates most of the manual configuration required when adding native image support to other frameworks.

Q6. How does Micronaut's declarative HTTP client work?

A: Annotate an interface with @Client('/api') and declare methods with @Get, @Post, etc., mirroring the server-side API. Micronaut generates a compile-time implementation using Netty. Combine with @LoadBalanced for client-side load balancing.

Q7. How does Micronaut support reactive programming?

A: Micronaut controllers can return Project Reactor Publisher<T>/Mono<T>/Flux<T> or RxJava Observable/Single. The Netty event loop handles I/O non-blocking. Annotate @ExecuteOn(TaskExecutors.IO) to shift blocking calls to a worker thread.

Q8. How does Micronaut's configuration system work?

A: Properties from application.yml are bound at compile time to @ConfigurationProperties POJO classes. Multiple environments (test, prod) load environment-specific files. MICRONAUT_APPLICATION_NAME env var and system properties can override file values.

Q9. How does Micronaut Security implement JWT authentication?

A: Add micronaut-security-jwt. Configure jwt.signatures.secret in application.yml. Annotate endpoints with @Secured(SecurityRule.IS_AUTHENTICATED) or @Secured('ROLE_ADMIN'). Micronaut validates Bearer tokens on every request using the configured

Q10. How do you write tests in Micronaut?

A: @MicronautTest starts the application context for the test. Use the generated @Client interface or inject an HttpClient to call endpoints. Replace beans with @MockBean to provide mock implementations. EmbeddedServer starts a real HTTP server.