Spring Platform Engineer Fieldbook

Spring Platform Engineer Fieldbook

> A simple, production-focused guide using the Feynman technique: learn an idea, explain it plainly, apply it, and expose what you do not yet understand.

\newpage

00. The Spring application architect role

Spring Framework is the foundation for many Java enterprise applications. Spring Boot makes that foundation fast to start and practical to operate.

Explain it like I am five

Imagine building a city from Lego. Plain Java gives you the bricks. Spring is the helpful organizer that connects the right buildings, adds common services, and lets you replace one part without rebuilding the

whole city. Spring Boot opens the box with sensible parts already selected so the city can start fast.

The architect's job

1. Understand the business journey, data, traffic, security, and recovery.
2. Decide the application boundary before choosing annotations.
3. Keep business rules independent from web, database, and messaging details.
4. Design dependency injection, transactions, APIs, security, and testing.
5. Make configuration external, observable, and safe to change.
6. Prove behavior with tests, metrics, failure drills, and production feedback.

Evidence of strong Spring judgment

- A package/module map showing dependency direction.
- A request and transaction sequence diagram.
- Explicit bean boundaries and configuration properties.
- API contracts, validation, errors, authentication, and authorization.
- Repository and transaction choices with concurrency behavior.
- Unit, slice, integration, contract, and end-to-end test strategy.
- Actuator exposure, metrics, traces, logs, alerts, and runbooks.
- Deployment, rollback, database migration, and recovery plan.

The core principle

Annotations save typing; they do not replace architecture. Use Spring to make business code easier to assemble, test, observe, and change.

Feynman check

Can you explain why a class receives its dependency instead of constructing it? If yes, you already understand the heart of Spring.

\newpage

01. Spring in simple terms

Spring manages **objects** and the relationships between them.

Four words to learn first

Word	Plain meaning
Bean	An object managed by Spring
Container	The place that creates and connects beans
Dependency	Something one object needs to do its job
Injection	Giving an object its dependency from outside

The main container interface is `ApplicationContext`. It reads configuration, creates beans, resolves dependencies, applies post-processors, publishes events, and later closes managed resources.

A tiny example

```
@Service
class CheckoutService {
  private final PaymentGateway gateway;

  CheckoutService(PaymentGateway gateway) {
    this.gateway = gateway;
  }
}
```

`CheckoutService` does not create a payment gateway. Spring supplies one. Constructor injection makes the dependency visible and makes ordinary unit testing easy.

Framework versus Boot

Spring Framework provides the programming model: container, AOP, transactions, web, data access, testing, and integration. Spring Boot chooses compatible dependencies, auto-configures common infrastructure, embeds a server, and adds production features. Boot uses Spring; it does not replace it.

Inversion of Control

Without IoC, application code decides how infrastructure is built. With IoC, configuration decides how objects are assembled. Business objects focus on business rules.

Feynman check

A chef should cook, not build the oven. Dependency injection gives the chef a working oven. The chef can later receive a test oven without changing recipes.

\newpage

02. IoC and dependency injection

Dependency injection separates **using** an object from **constructing** it.

How beans enter the container

- `@Bean` methods in `@Configuration` classes.
- Component scanning of `@Component`, `@Service`, `@Repository`, and `@Controller`.
- Framework and Boot auto-configuration.
- Programmatic registration for specialized cases.

Prefer explicit package boundaries and focused configuration. Scanning an entire organization package makes accidental beans harder to see.

Constructor injection

Constructor injection is normally the best default. Required dependencies are visible, final fields are possible, and tests can construct the class directly. Avoid field injection because it hides dependencies and couples tests to the container.

Multiple candidates

If two beans implement one interface, Spring needs help. Use one `@Primary`, use `@Qualifier` at the injection point, or redesign the boundary. A qualifier describes meaning better than relying on a bean name.

Circular dependencies

If A needs B and B needs A, the design may mix responsibilities. Break the cycle with a third service, domain event, or clearer direction. Do not hide a bad boundary with lazy injection unless the lifecycle truly demands it.

Optional dependencies

Use `Optional`, `ObjectProvider`, or conditional configuration only when the capability is genuinely optional. Required business collaborators should fail fast at startup.

Feynman check

The container is a seating planner. Constructors say who must sit together. Qualifiers distinguish two people with the same job. A circular dependency is two people each refusing to sit until the other sits first.

\newpage

03. Beans, lifecycle, and scopes

The container creates a bean, injects dependencies, applies processors, calls initialization callbacks, exposes it for use, and invokes destruction callbacks when the context closes.

Lifecycle tools

- Constructor: establish valid required state.
- `@PostConstruct`: initialize after dependencies exist.
- `@PreDestroy`: release managed resources.
- `BeanPostProcessor`: change or wrap many beans as infrastructure.
- `SmartLifecycle`: coordinate start/stop for active components.

Keep constructors fast and avoid remote calls during startup when possible. Startup should fail clearly for invalid configuration but not depend on a temporary network response unless absolutely required.

Scopes

Scope	Meaning
Singleton	One bean instance per application context; default
Prototype	New instance each time requested
Request	One instance per HTTP request
Session	One instance per HTTP session
Application	One per servlet application

A singleton may serve many threads. Mutable fields can create races. Prefer stateless singleton services and keep request data in method variables.

Proxies

Spring may wrap a bean with a proxy to add transactions, security, caching, or AOP behavior. Calls entering through the proxy receive that behavior. Self-invocation—one method calling another on `this`—can bypass proxy advice.

Feynman check

A proxy is a receptionist standing before the real worker. If the worker calls their own second task internally, the call never passes the receptionist.

\newpage

04. Spring Boot and auto-configuration

Spring Boot starts with the classpath, configuration properties, and existing beans, then supplies sensible missing infrastructure.

What `@SpringBootApplication` combines

- `@Configuration`: this class can define beans.
- `@EnableAutoConfiguration`: apply matching auto-configuration.
- `@ComponentScan`: find components below the application package.

Conditional configuration

Auto-configuration uses conditions such as “class exists,” “property is set,” or “no bean of this type exists.” Your bean usually makes Boot back away. The conditions report and startup diagnostics explain why a configuration matched.

Starters and dependency management

Starters group common dependencies. Boot's dependency management chooses a tested set of versions. Avoid overriding individual library versions casually; you can leave the tested compatibility set.

Current baseline

As checked on 2026-08-05, the current docs list Spring Boot 4.1.0 with Spring Framework 7.0.8+, Java 17+, Maven 3.6.3+, and supported Gradle 8/9 lines. Always verify the system-requirements page before starting or upgrading.

Customize safely

Prefer configuration properties or a user bean over excluding broad auto-configuration. Use exclusions when the whole feature truly does not fit. Keep the main application class in a root package so scanning is predictable.

Feynman check

Boot is a hotel room prepared from your reservation. If the classpath says “web guest,” it prepares a server. If you provide your own pillow—a bean—Boot usually does not replace it.

\newpage

05. Configuration, properties, and profiles

Code should describe behavior. Configuration should describe what changes between environments.

Typed properties

Use `@ConfigurationProperties` for a group of related settings. Typed objects provide validation, metadata, and one place to understand configuration. Prefer them over many scattered `@Value` expressions.

```
@ConfigurationProperties("payments")
public record PaymentProperties(
    @NotBlank URI endpoint,
    @Positive Duration timeout) {}
```

Property precedence

Spring Boot combines files, environment variables, system properties, command line arguments, test properties, and other sources with a defined precedence. Know which source wins. Do not let an emergency command-line override become an invisible permanent configuration.

Profiles

Profiles activate groups of beans or properties. Use them sparingly for genuine environment capabilities. Avoid profile combinations such as `prod-eu-blue-customer7`; they create untestable configuration branches.

Secrets

Do not store passwords or tokens in source control. Retrieve them from a secret manager or injected runtime source. Ensure logs and Actuator endpoints do not expose them. Plan rotation and application reload.

Validation and failure

Validate required configuration at startup with clear messages. A service that starts with a blank payment endpoint is not “resilient”; it is misleading.

Feynman check

Configuration is a labeled control panel. Typed properties group related switches, validation stops impossible settings, and a secret manager keeps the master key away from the panel.

\newpage

06. Spring MVC request lifecycle

Spring MVC uses the Servlet model. It is a strong default for most blocking web applications.

Request path

1. The servlet container accepts HTTP.
2. Filters run, including the Spring Security filter chain.
3. `DispatcherServlet` receives the request.
4. A handler mapping finds a controller method.
5. An adapter resolves arguments and invokes the method.
6. The controller calls application services.
7. Return-value handlers and message converters create the response.
8. Exception handlers translate failures.

Controller boundaries

Controllers translate HTTP into application commands and results. Keep business rules in application/domain services. Validate input at the boundary with Bean Validation and use explicit request/response DTOs instead of exposing entities.

MVC or WebFlux?

Choose MVC for blocking libraries and the ordinary thread-per-request model. Choose WebFlux when the whole important path is non-blocking and the workload benefits from many concurrent waiting operations or streaming. Wrapping JDBC in a reactive controller does not make it non-blocking.

REST clients

Modern Spring offers `RestClient` for synchronous use and `WebClient` for reactive use. Configure timeouts, connection pools, observability, error mapping, retries, and authentication. Do not create a new client per request.

Feynman check

`DispatcherServlet` is the reception desk. It finds the right controller, converts the envelope into Java data, and converts the answer back into HTTP.

\newpage

07. REST contracts, validation, and errors

An API contract includes more than a URL. It defines methods, schemas, status codes, errors, authentication, idempotency, pagination, and compatibility.

Resource design

Use nouns for resources and HTTP semantics intentionally. `GET` reads, `POST` creates or commands, `PUT` replaces, `PATCH` changes part, and `DELETE` removes. Return `201 Created` with a location for creation when appropriate.

Validation

Validate syntax and shape at the controller boundary. Validate business rules inside the application/domain layer. A syntactically valid order can still violate credit or inventory rules.

Errors

Use `@ControllerAdvice` and exception handlers to produce a consistent, machine-readable error format. Problem Details is a useful HTTP standard. Do not leak stack traces, SQL, internal hostnames, or secrets.

Compatibility

Prefer additive changes. Make consumers tolerant of new fields. Deprecate with measurements and a removal date. Use contract tests for important consumers.

Idempotency

For retryable commands such as payment or order creation, accept an idempotency key and store the completed outcome. Retries should not duplicate business effects.

Feynman check

An API is a restaurant menu and ordering rule. Validation checks whether the order is readable. Business rules check whether the meal can actually be sold. A stable error format is the clear explanation when it cannot.

\newpage

08. Data access and transactions

Spring translates low-level data exceptions and provides a common transaction model across supported technologies.

Data choices

- `JdbcClient` or `JdbcTemplate`: explicit SQL and mapping.
- Spring Data JDBC: aggregate-focused relational persistence.
- Spring Data JPA: repositories over JPA/Hibernate.
- R2DBC: reactive access for supported relational systems.
- Spring Data modules: consistent repository ideas for other stores.

Choose the model that matches access patterns and team knowledge. A repository interface does not remove database design.

Transactions

`@Transactional` normally uses a proxy. A runtime exception rolls back by default; checked-exception behavior requires deliberate rules. Transaction propagation controls whether a method joins, creates, or avoids a transaction. Isolation controls concurrency phenomena.

Keep transactions short. Do not hold a database transaction open while waiting for a slow network call.

The dual-write problem

Updating a database and publishing a message are two separate systems. Use an outbox pattern: write the business change and outbox event in one transaction, then publish asynchronously with idempotent consumers.

JPA cautions

Understand entity state, dirty checking, lazy loading, fetching, N+1 queries, locking, pagination, and flush timing. Do not serialize entities directly from controllers.

Feynman check

A transaction is a pencil-and-eraser agreement: either all related database changes become permanent or they are erased together. A message broker uses a different notebook, so an outbox safely carries the note across.

\newpage

09. Testing Spring applications

Use the smallest test that proves the behavior.

Test pyramid

- Plain unit test: business class with fakes or mocks; no Spring context.
- Slice test: one layer such as MVC or JPA with focused configuration.
- Integration test: multiple real components and infrastructure.
- Contract test: provider and consumer agree on messages or APIs.
- End-to-end test: a few critical journeys through the deployed system.

Spring tools

`MockMvc` tests Spring MVC without a real server. `WebTestClient` tests WebFlux and can also drive live HTTP. `@SpringBootTest` loads broad Boot context; use it when that breadth is the subject, not as the default for every method.

Testcontainers can run real databases, brokers, and services in containers. Prefer realistic infrastructure for SQL, migration, and serialization behavior that an in-memory substitute can hide.

Transactional tests

A test transaction that rolls back can hide production flush, commit, and event behavior. Force flush or run true integration boundaries when those details matter.

Healthy test design

Assert public behavior, not internal call order. Keep fixtures readable and deterministic. Test validation, authorization, failures, timeouts, and retries, not only success.

Feynman check

A unit test checks one gear. A slice test checks one machine section. An integration test connects machines. An end-to-end test follows one real parcel through the factory.

\newpage

10. Spring Security

Spring Security handles authentication, authorization, common web protections, and security context propagation.

Authentication and authorization

Authentication asks “who are you?” Authorization asks “may you do this?” For browser sessions, protect login, session fixation, CSRF, cookies, and logout. For APIs, validate OAuth 2.0/OIDC tokens as a resource server and authorize scopes/claims plus business ownership.

Filter chain

Web security runs through a `SecurityFilterChain`. Define request rules explicitly and keep a deny-by-default mindset. Method security with `@EnableMethodSecurity` can protect business operations, but it should complement—not replace—HTTP rules.

Passwords and tokens

Store passwords only with an adaptive one-way password encoder. Never log passwords or bearer tokens. Validate token issuer, audience, signature, expiry, and intended use.

CSRF and CORS

CSRF matters when browsers automatically attach credentials such as cookies. Do not disable it blindly. CORS is a browser cross-origin rule, not authentication. Allow only required origins, methods, and headers.

Actuator

Expose only required management endpoints, preferably on a controlled path or network. Health detail, environment, mappings, heap dumps, and loggers can leak information or change behavior.

Feynman check

The filter chain is airport security. Authentication checks the passport. Authorization checks the boarding pass. CORS decides which airport entrances a browser may use; it does not prove identity.

\newpage

11. AOP, events, scheduling, and caching

Spring adds cross-cutting behavior around beans through proxies and supports events, tasks, schedules, and cache abstractions.

AOP

Aspect-oriented programming is useful for concerns applied consistently across many methods: transactions, security, metrics, tracing, or narrowly defined logging. A pointcut selects join points; advice runs before, after, or around them.

Avoid hiding core business flow in aspects. If reading the method cannot reveal that an order is published or a fee is charged, the design is surprising.

Events

Application events decouple in-process components. By default, listeners run in the publisher's thread. `@TransactionalEventListener` can bind handling to a transaction phase. In-process events disappear if the process crashes; use a durable broker/outbox for cross-service guarantees.

Scheduling and async

`@Scheduled` triggers recurring work. In multiple replicas, every replica may run it unless coordination exists. `@Async` uses an executor; configure pool size, queue, rejection, context, and exception handling.

Caching

`@Cacheable` can avoid repeated work, but every cache needs a key, TTL, invalidation, maximum size, consistency expectation, and failure behavior.

Feynman check

A proxy is a gate adding checks. An event is a bell inside one building. A message broker is a postal service between buildings. A cache is a whiteboard copy that can become stale.

\newpage

12. Messaging and enterprise integration

Messaging separates the speed and availability of producers from consumers.

Core patterns

- Queue: one consumer group processes work.
- Publish/subscribe: several consumers receive an event.
- Dead-letter destination: isolates repeatedly failing messages.
- Idempotent consumer: repeated delivery does not repeat the business effect.
- Outbox: database change and event intent commit together.

Spring provides abstractions for Kafka, AMQP/RabbitMQ, JMS, Pulsar, and Spring Integration patterns. Choose the broker from ordering, throughput, retention, replay, routing, and operations—not the annotation you prefer.

Consumer design

Validate schemas, bound retries, add backoff and jitter, route poison messages, control concurrency, preserve correlation IDs, and observe lag/age. Acknowledge only after the durable business effect is safe.

Integration with SAP, Salesforce, and Workday

Wrap each external system with an adapter. Translate external data into the domain language. Apply timeouts, rate limits, circuit breakers, idempotency, and reconciliation. Do not let vendor schemas leak into every service.

Feynman check

A synchronous call is a phone conversation: both sides must be present. A queue is a mailbox: one side can leave a durable message and continue.

\newpage

13. Actuator, observability, and operations

Production readiness begins with user health and controlled management.

Actuator

Spring Boot Actuator provides management endpoints and infrastructure for health, metrics, auditing, and operations. Expose only what operators need. Liveness should mean the process is locally recoverable; readiness should mean the instance may receive traffic.

Observability

Micrometer provides vendor-neutral metrics and observation. Use OpenTelemetry or compatible tracing integration for distributed requests. Correlation IDs connect logs, traces, and messages.

Start with request rate, error rate, duration, and saturation. Add business signals such as completed orders and payment failures. Define SLIs and SLOs from user journeys, then alert on error-budget burn.

Logging

Use structured logs with event name, severity, service, environment, trace ID, and safe business identifiers. Never log credentials, bearer tokens, full payment data, or unnecessary personal information.

Graceful shutdown

Stop accepting new traffic, complete bounded in-flight work, pause consumers, and close pools. Ensure the deployment platform grants enough termination time.

Feynman check

Actuator is the service panel, metrics are gauges, traces follow one parcel, logs are the diary, and an SLO says how often the customer journey must work.

\newpage

14. Performance and resilience

Performance is the user experience under expected load. Resilience is useful behavior when dependencies, instances, or networks fail.

Measure first

Load-test realistic traffic, payloads, database size, cache state, and failure. Measure latency percentiles, throughput, errors, CPU, memory, garbage collection, thread pools, connection pools, queue depth, and dependency time.

Protect resources

Configure HTTP, database, broker, and executor pools from total instance capacity. Bound queues. A huge queue hides overload and increases latency. Use bulkheads so one dependency cannot consume every thread.

Remote-call policy

Every remote call needs a timeout. Retry only transient and safe operations, with a small bound, exponential backoff, and jitter. Use circuit breakers to stop repeated calls during an outage. Provide graceful degradation for optional features.

JVM and native choices

Tune only after profiling. Modern JVM ergonomics are strong. Native images can improve startup and memory for suitable workloads but change build, reflection/resource configuration, debugging, and peak-throughput trade-offs.

Feynman check

A timeout is a clock, a retry is another attempt, a circuit breaker closes the road, and a bulkhead keeps one flooded room from sinking the whole ship.

\newpage

15. Modular monoliths and microservices

Start from business boundaries and team ownership. Do not start from the number of services.

Modular monolith

One deployable application can contain strongly separated business modules. This gives simple transactions, local calls, one deployment, and lower operations cost. Spring Modulith can verify module dependencies, support module-oriented testing, and document the structure.

Microservices

Separate services can isolate deployment, scaling, data, failure, and team ownership. They also introduce network failure, distributed data, versioned contracts, tracing, deployment coordination, and higher platform cost.

Dependency rule

Business modules depend inward on domain concepts. Web, JPA, brokers, and vendors are adapters. Keep SAP, Salesforce, and Workday objects at the edges.

When to extract

Extract a service when a clear business capability needs independent ownership, lifecycle, scaling, data, compliance, or failure isolation. Measure the benefit.

Migration

Use the strangler pattern: establish a stable contract, route one capability, observe, reconcile data, and retire the old path. Avoid a big-bang rewrite.

Feynman check

A modular monolith is one apartment building with strong walls. Microservices are separate houses connected by roads. Houses offer independence but every road can fail.

\newpage

16. Real-life scenario: PeopleFlow

PeopleFlow is a global hiring and employee platform. Customers use its web and mobile APIs. It integrates with Salesforce for sales/customer context, Workday for employee records, and SAP for finance and purchase approvals.

Requirements

- 99.95% candidate application availability.
- Tenfold campaign traffic within ten minutes.
- No duplicate hires, invoices, or onboarding tasks.
- Workday and SAP may be slow or unavailable for hours.
- Five-minute RPO and one-hour RTO for hiring workflows.
- Independent domain teams with consistent security and observability.

Architecture

Build a Spring Boot application organized as modules: Candidate, Job, Application, Offer, Onboarding, and Integration. Start as a modular monolith to keep transactions and delivery simple. Verify module boundaries

with Spring Modulith. Extract only a proven independent capability later.

Spring MVC controllers accept DTOs, apply Bean Validation, and call application services. Business rules live in domain modules. Spring Data JPA persists aggregates to PostgreSQL. Transactions are short.

When an offer is accepted, the application commits the state and an outbox record together. A publisher sends durable events to Kafka. Idempotent adapters update Workday, SAP, and Salesforce with timeouts, backoff, rate limits, dead letters, and reconciliation. A vendor outage never holds the user HTTP request open.

Use Spring Security as an OAuth 2.0 resource server. Validate issuer and audience, then authorize tenant and business ownership in services. Secrets come from a managed store. The application runs as a non-root container with immutable artifacts and database migrations designed with expand/contract.

Actuator exposes controlled health and Prometheus metrics. OpenTelemetry traces HTTP, SQL, Kafka, and vendor calls. SLO alerts focus on successful applications, offer acceptance, integration age, and error-budget burn.

Failure example: Workday is down

Offer acceptance still succeeds after the local durable commit. The Workday event waits in the broker. Consumer concurrency is capped. Queue age alerts operators. When Workday returns, idempotent processing drains the backlog and reconciliation checks missed records.

Prove it

Test module boundaries, transaction rollback, authorization, schema contracts, duplicate messages, vendor timeouts, campaign load, instance termination, database restore, and a one-hour regional recovery.

Decision record

Chosen: modular monolith, MVC, PostgreSQL, outbox plus Kafka, OAuth resource server, controlled Actuator, and evidence-based extraction.

Rejected: one microservice per table, synchronous vendor calls in user requests, shared admin tokens, and unbounded retries.

\newpage

17. Interview, certification, and glossary

The published Broadcom Spring Professional Develop guide (last updated 2024-01-26) describes exam `2V0-72.22`: 60 items, 130 minutes, and a scaled passing score of 300. It covers Core, Data, MVC, Testing,

Security, and Boot. The same guide references Spring Framework 5.3 and Spring Boot 2.5, so verify the live registration portal before paying.

This site's 60-question mock uses the official item count and duration as a format benchmark, but teaches current Spring concepts and contains only original questions.

Interview frame: SPRING

- **S**cope: user journey, business rule, SLO, security, recovery.
- **P**ackages: domain modules, dependency direction, ownership.
- **R**equest: HTTP/message flow, validation, transaction, errors.
- **I**njection: beans, configuration, lifecycle, proxies.
- **N**on-functional: security, tests, observability, resilience, performance.
- **G**o-live: packaging, migration, rollout, rollback, operations, cost.

Essential terminology

Term	Meaning
IoC / DI	Inversion of Control / Dependency Injection
Bean	Object managed by Spring
ApplicationContext	Spring container and application services
AOP	Aspect-Oriented Programming
MVC	Model-View-Controller servlet web framework
WebFlux	Reactive web framework
DTO	Data Transfer Object
ORM / JPA	Object-relational mapping / Java persistence specification
JDBC / R2DBC	Blocking / reactive relational database access APIs
SpEL	Spring Expression Language
AOT	Ahead-of-Time processing
CSRF / CORS	Browser request-forgery defense / cross-origin policy
OAuth 2.0 / OIDC	Delegated authorization / identity layer
SLI / SLO	Service measurement / reliability target
RPO / RTO	Tolerated data loss / restoration time
DLQ	Dead-letter queue

Final Feynman challenge

Draw one request from HTTP through security, controller, service, transaction, repository, outbox, message, and external adapter. Explain where Spring creates objects, where a proxy adds behavior, where failures roll back, and where retries are safe. Anything unclear is your next study target.

Official references

- Spring Framework reference
- Spring Boot reference
- Spring Boot system requirements
- Spring Boot Actuator
- Spring projects
- Broadcom Spring Professional exam guide