Spring Boot Microservices/API Incident Management and Observability Guide

1. Purpose and Function of Spring Boot in Microservices/API Development

Spring Boot simplifies the development of stand-alone, production-grade Spring applications. It's designed to streamline the creation of microservices and APIs by offering pre-configured defaults, embedded servers, and seamless integration with monitoring and deployment tools.

Key Features:

- Auto Configuration: Minimizes manual setup by automatically configuring based on classpath contents.
- Embedded Servers: Allows apps to run independently without external web servers.
- Spring Boot Starters: Simplify dependency management for common use cases.
- Spring Actuator: Offers production-ready endpoints for monitoring and management.
- Rapid Prototyping: Developers can go from concept to working API quickly.

2. Rationale for Chosen Failure Modes

Two common and impactful failure modes were chosen: application startup failure in development and API unavailability in production. These represent high-frequency, high-impact incidents where structured response is critical.

3. Incident Management Runbook - Development Environment

Failure Mode: Application Fails to Start

• Symptoms:

- ApplicationContextException or BeanCreationException in logs.
- Port already in use errors.
- App fails before 'Started Application' message.

• Diagnostics:

- Check logs (e.g., application.log or stdout).
- Use /actuator/health if partially running.
- Review recent code or config changes.
- Run mvn dependency:tree or ./gradlew dependencies to check dependency conflicts.

• Causes:

- Port conflict (default 8080 in use).

- Invalid or missing configuration values.
- Missing beans or cyclic dependencies.
- Environment variables not set correctly.

• Immediate Actions:

- Kill conflicting process on the port: lsof -i :8080 → kill -9 <PID>.
- Rebuild and restart: mvn clean install && java -jar target/app.jar.
- Comment out suspicious beans/configs and reintroduce incrementally.

• Remediation Steps:

- Specify a non-conflicting port using server.port.
- Introduce better validation for configs.
- Use @ConditionalOnMissingBean to avoid autowiring issues.
- Ensure environment variables are included in launch profiles or scripts.

• Postmortem Requirements:

- Document incident and root cause in the dev wiki.
- Add test cases to prevent regression.
- Update developer onboarding or environment setup guide.
- Automate validation of config files during builds.

4. Incident Management Runbook - Production Environment

Failure Mode: API Unavailability or Latency Issues

• Symptoms:

- API Gateway returns 502/504 errors.
- High latency or timeouts in HTTP responses.
- Actuator /health endpoint reports DOWN or OUT_OF_SERVICE.
- Spikes in CPU or memory usage observed in monitoring tools.

• Diagnostics:

- Check logs via ELK, Splunk, or CloudWatch.
- Inspect /actuator endpoints (health, metrics).
- Use APM (Datadog, New Relic) to trace slow requests.
- Review deployment and infrastructure changes.

• Causes:

- Out-of-memory errors or OOMKilled containers.
- Thread or connection pool exhaustion.
- Failure of downstream services (DB, API, etc.).
- Configuration mismatch or missing secrets.

• Immediate Actions:

- Roll back recent deployment from CI/CD.
- Scale up replicas or restart affected pods.
- Redirect traffic to healthy nodes using load balancer.
- Temporarily disable or degrade non-essential features.

• Remediation Steps:

- Add circuit breakers and retries using Resilience4j.
- Enable auto-scaling policies in orchestrator (K8s, ECS).
- Tune connection/thread pool limits.
- Improve observability and alerts for health and performance metrics.

• Postmortem Requirements:

- Perform RCA and document in incident tracker (e.g., Jira or Confluence).
- Review missed alerts and observability gaps.
- Update runbook based on lessons learned.
- Verify rollback effectiveness and CI/CD pipeline safety checks.

Additional DevOps/SRE Considerations

Area	Tools/Practices
Observability	Spring Boot Actuator + Prometheus + Grafana
Monitoring	ELK/EFK Stack, Datadog, CloudWatch
CI/CD Integration	Jenkins, GitLab CI, GitHub Actions
Containerization	Dockerized Spring Boot apps + Kubernetes deployment
Secrets Management	HashiCorp Vault, AWS Secrets Manager
Performance Testing	JMeter, Gatling, k6
Alerting	Prometheus Alertmanager, PagerDuty, Opsgenie