

# Assessments

## Chapter 2

1. What is the purpose of the `@SpringBootApplication` annotation?

**Answer:** The following functionality is provided by the `@SpringBootApplication` annotation:

- It enables component scanning; that is, looking for Spring components and configuration classes, in the package of the application class and all its sub-packages.
- The application class itself becomes a configuration class.
- It enables auto-configuration, where Spring Boot looks for JAR files in the classpath that it can configure automatically. For example, if you have Tomcat in the classpath, Spring Boot will automatically configure Tomcat as an embedded web server.

2. What are the main differences between the older Spring component for developing REST services, Spring Web MVC, and the new Spring WebFlux?

**Answer:** Spring WebFlux supports the development of reactive, that is, non-blocking, HTTP clients and services. It also supports non-Servlet-based web servers such as Netty. Spring Web MVC only supports Servlet-based web servers. Finally, Spring WebFlux comes with a new functional programming model that complements the annotation-driven model introduced by Spring Web MVC (which Spring WebFlux still supports).

3. How does SpringFox help a developer document REST APIs?

**Answer:** SpringFox can create Swagger-based API documentation at runtime. It does so by examining the application at startup, for example, inspecting WebFlux- and Swagger-based annotations.

4. What is the function of a repository in Spring Data and what is the simplest possible implementation of a repository?

**Answer:** Repositories provide a similar, but not portable, programming model for storing and accessing data in different types of databases; for example, relational, document, key-value, and graph databases. In its most basic form, a repository can be declared as a Java interface and Spring Data will generate its implementation on the fly using optional conventions, for example:

```
public interface MyRepository extends CrudRepository<MyEntity,  
MyPK>
```

5. What is the purpose of a binder in Spring Cloud Stream?

**Answer:** A binder provides the actual integration with a specific messaging system, similar to what a JDBC driver does for a specific type of database. Spring Cloud Stream comes with out-of-the-box support for Apache Kafka and RabbitMQ.

6. What is the purpose of Docker Compose?

**Answer:** Docker Compose is a tool in Docker that's used to manage (for example, start, scale, log, and stop) a group of related Docker containers with a single command.

## Chapter 3

1. What is the command that lists available dependencies when you create a new Spring Boot project using the `spring init` Spring Initializr CLI tool?

**Answer:** `spring init --list`

2. How can you set up Gradle to build multiple related projects with one command?

**Answer:** Create a file named `settings.gradle` in the root folder with one `include` statement for each subproject or module, for example:

```
include ':microservices:product-service'  
include ':microservices:review-service'
```

3. What are the `@PathVariable` and `@RequestParam` annotations used for?

**Answer:** They are used in Spring WebFlux to indicate that a method parameter should be bound either to the URI part or as a request parameter in an API.

For an example usage of the `@PathVariable` annotation, refer to the `getProduct` method in `$BOOK_HOME/Chapter03/2-basic-rest-services/api/src/main/java/se/magnus/api/core/product/ProductService.java`. For an example usage of the `@RequestParam` annotation, refer to the `getRecommendations` method in `$BOOK_HOME/Chapter03/2-basic-rest-services/api/src/main/java/se/magnus/api/core/recommendation/RecommendationService.java`.

4. How can you separate protocol-specific error handling from the business logic in an API implementation class?

**Answer:** You can use a common class annotated as `@RestControllerAdvice` and declare exception handler methods for the exceptions that the business logic might throw. For example, mapping `InvalidInputException` to the HTTP response code `UNPROCESSABLE_ENTITY` might appear as follows:

```
@ResponseStatus(UNPROCESSABLE_ENTITY)
@ExceptionHandler(InvalidInputException.class)
public @ResponseBody HttpErrorInfo handleInvalidInputException(
    ServerHttpRequest request, Exception ex) {

    return createHttpErrorInfo(UNPROCESSABLE_ENTITY, request, ex);
}
```

5. What is Mockito used for?

**Answer:** Mockito can be used to mock external dependencies, for example, an external API, to make it easier to write unit tests. For an example, refer to `$BOOK_HOME/Chapter03/2-basic-rest-services/microservices/product-composite-service/src/test/java/se/magnus/microservices/composite/product/ProductCompositeServiceApplicationTests.java`.

## Chapter 4

1. What are the major differences between a virtual machine and a Docker container?

**Answer:** A virtual machine uses a hypervisor to run a complete copy of an operating system, while containers are processes in a Linux host.

2. What is the purpose of namespaces and cgroups in Docker?

**Answer:** Linux namespaces are used to provide isolation between containers of global system resources, such as users, processes, filesystems, and networking. Linux Control Groups (also known as cgroups) are used to limit the amount of CPU and memory that a container is allowed to consume.

3. What happens with a Java application that doesn't honor the max memory settings in a container and allocates more memory than it is allowed to?

**Answer:** It is killed by Docker without giving the Java application any chance to react by, for example, logging relevant information about what caused it to allocate too much memory.

4. How can we make a Spring-based application run as a Docker container without requiring modifications of its source code?

**Answer:** By using Spring profiles, we can set up different configurations for running with or without Docker, for example, for hostnames and ports. The Spring file to use can be specified using an environment variable, `SPRING_PROFILES_ACTIVE`, for example, when declaring how to start the application using Docker compose:

```
review:
  build: microservices/review-service
  environment:
    - SPRING_PROFILES_ACTIVE=docker
```

5. Why will the following Docker Compose code snippet not work?

```
review:
  build: microservices/review-service
  ports:
    - "8080:8080"
  environment:
    - SPRING_PROFILES_ACTIVE=docker
product-composite:
  build: microservices/product-composite-service
  ports:
    - "8080:8080"
  environment:
    - SPRING_PROFILES_ACTIVE=docker
```

**Answer:** Since both the review and product-composite services try to bind their local port, 8080, to the same port in the Docker host, one of them will get an error during startup along the lines of the following: Bind for 0.0.0.0:8080 failed: port is already allocated.

Changing the port mapping for one of the services will solve the problem, for example, mapping to port 8081 on the Docker host:

```
ports:
  - "8081:8080"
```

## Chapter 5

1. How does springdoc-openapi help us to create API documentation for RESTful services?

**Answer:** springdoc-openapi makes it possible to keep the API documentation together with the source code that implements it. This means that the API documentation will share the same life cycle as the source code, minimizing the risk of the documentation becoming outdated.

2. What specification for documenting APIs does springdoc-openapi support?

**Answer:** OpenAPI specification v3. For details, see <https://swagger.io/specification/>.

3. What is the purpose of the springdoc-openapi OpenAPI bean?

**Answer:** It is used to specify general information about an API, such as the version, license, and contact information.

4. Name some annotations that springdoc-openapi reads at runtime to create the API documentation on the fly.

**Answer:** Related Api annotations from the `io.swagger.v3.oas.annotations` package and related RequestMapping annotations from the `org.springframework.web.bind.annotation` package.

5. What does the code `" | "` mean in a YAML file?

**Answer:** It is the start of a multiline value, for example:

```
field: |  
  value, row 1  
  value, row 2
```

6. How can you repeat a call to an API performed using the embedded Swagger viewer without using the viewer again?

**Answer:** You can copy the prepared curl command in the response part and run it in a Terminal window.

## Chapter 6

1. Spring Data, a common programming model based on entities and repositories, can be used for different types of database engines. From the source code examples in this chapter, what are the most important differences in the persistence code for MySQL and MongoDB?

**Answer:** The MySQL entity class uses JPA annotations, for example, the `@Entity` annotation. The MongoDB entity class uses annotations from the Spring Data MongoDB module, for example, `@Document`. Since MySQL, just like any other relational database, is transactional, custom methods that are added to the MySQL repository must have annotations describing their transactional semantics, for example, using a read-only transaction for methods that only read data from the MySQL database.

2. What is required to implement optimistic locking using Spring Data?

**Answer:** Annotate a field in the entity class with the `@Version` annotation.

3. What is MapStruct used for?

**Answer:** MapStruct is a Java bean mapping tool that simplifies transforming model objects used in the API into entity objects used in the persistence layer.

4. What does it mean if an operation is idempotent and why is that useful?

**Answer:** It means that the operation will return the same result if it's called several times with the same input parameters. The state of an underlying database, if any, will also remain the same if the operation is called one or several times with the same input parameters. Operations that may experience temporary errors, for example, network-related problems, can simply be called several times, in the case of a temporary error, until they return a successful response, given that they are idempotent.

5. How can we access the data that is stored in MySQL and MongoDB databases without using the API?

**Answer:** We can access the data as follows:

1. Since databases run as Docker containers under the control of Docker Compose, the Docker Compose `exec` command can be used to run the default CLI tools for each database, `mysql` and `mongo`.
2. Docker Compose is set up to expose the default ports, 3306 and 27017, for each database in the Docker host. When using Docker Desktop for Mac or Windows, these ports are forwarded to `localhost`. This makes it possible to use a locally installed graphical database client tool, as if the databases were running directly on `localhost`.

## Chapter 7

1. Why is it important to know how to develop reactive microservices?

**Answer:** To scale up the usage of a system landscape of cooperating microservices, it is important that the microservices are designed to be scalable and resilient. The foundation for this is that they are message-driven; in other words, they are either designed as non-blocking synchronous request/reply services or as asynchronous event-driven services.

2. How do you choose between non-blocking synchronous APIs and event/message-driven asynchronous services?

**Answer:** Asynchronous message passing of events is preferable over synchronous APIs. This is because the microservice will only depend on access to the messaging system at runtime, instead of being dependent on synchronous access to a number of other microservices. There are, however, a number of cases where it may be favorable to use non-blocking synchronous APIs, for example:

- For read operations where an end user is waiting for a response
- Where the client platforms are more suitable for consuming synchronous APIs, for example, mobile apps or SPA web applications
- Where the clients will connect to the service from other organizations where it might be hard to agree on a common messaging system to use across organizations

3. What makes an event different from a message?

**Answer:** Messaging systems handle **messages** that typically consist of headers and a body. An **event** is a message that describes something that has happened. For events, the message body can be used to describe the type of event, the event, and the data, and to supply a timestamp for when the event occurred.

4. Name some challenges associated with message-driven asynchronous services. How do we handle them?

**Answer:** The challenges are as follows:

- Scaling up the number of instances that process incoming messages in a consumer, so that a message is only processed by one of the instances. This can be solved by using **consumer groups**.

- Processing invalid messages, also known as poison messages, and handling temporary infrastructure problems. This can be solved with **retries** and **dead-letter queues**.
- Handling requirements on guaranteed delivery order and high throughput. This can be handled by **partitions**.

5. Why is the following test not failing?

```
@Test
void testStepVerifier() {

    StepVerifier.create(Flux.just(1, 2, 3, 4)
        .filter(n -> n % 2 == 0)
        .map(n -> n * 2)
        .log())
        .expectNext(4, 8, 12);
}
```

First, ensure that the test fails. Next, correct the test so that it succeeds.

**Answer:** A call to subscribe is missing, so nothing will happen when running the test. The preceding code example only declares the processing of a stream. The lack of output from the call to the `log` method indicates that no processing happened.

To actually get the stream processed, we need someone to subscribe to it. Add a call to the `.verifyComplete()` method after the call to the `expectNext` method. It will subscribe to the stream and verify that it is completed.

Rerunning the test will result in an error message such as expected:  
`onNext(12); actual: onComplete()`.

The error is caused by a mismatch between the `expectNext` method and the `Flux` source. The `Flux` source does not emit the value 6, required by the `expectNext` method. Fix the issue by either adding 6 as the last value in the `Flux` source or by removing 12 from the expected values.

6. What are the challenges in writing tests with reactive code using JUnit and how can we handle them?

**Answer:** Testing asynchronous, event-driven microservices is, by nature, difficult. Tests typically need to synchronize on the asynchronous background processing in some way to be able to verify their results. Spring Cloud Stream comes with support, the `TestSupportBinder` class, for verifying what messages have been sent without using any messaging system during the tests!



## Chapter 8

1. What is the purpose of Netflix Eureka?

**Answer:** It is used as a discovery service.

2. What are the main features of Spring Cloud Gateway?

**Answer:** Acting as an edge server, it can hide private APIs and secure public APIs from external usage. It also provides a single entry point to the public APIs in a system landscape of microservices.

3. What backends are supported by Spring Cloud Config?

**Answer:** Spring Cloud Config comes with built-in support for the following:

- Git repositories
- Local filesystem
- HashiCorp Vault
- JDBC databases

4. What are the capabilities that Resilience4j provides?

**Answer:** The main capabilities are as follows:

- Circuit breaker
- Rate limiter
- Bulkhead
- Retries
- Timeout

5. What are the concepts of trace tree and span used for in distributed tracing, and what is the paper called that defined them?

**Answer:** The paper is named *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*; see <https://ai.google/research/pubs/pub36356>. In Dapper, the tracing information from a complete workflow is called a trace tree, while the subparts of the tree, such as the basic units of work, are called spans. Spans can, in turn, consist of sub-spans, which form the trace tree. A correlation ID is called a `TraceId`, and a span is identified by its own unique `SpanId`, along with the `TraceId` property of the trace tree it belongs to.

## Chapter 9

1. What is required to turn a Spring Boot application created with Spring Initializr into a fully fledged Netflix Eureka Server?

**Answer:** The requirements are as follows:

- Add a build dependency for `spring-cloud-starter-netflix-eureka-server` to the build file, `build.gradle`
- Add the `@EnableEurekaServer` annotation to the application class
- Add configuration, depending on how you want to use the Eureka server, for example, as a single instance used for development, or as a high-availability configuration with multiple instances aimed at production use

2. What is required to make a Spring Boot-based microservice register itself automatically as a startup with Netflix Eureka?

**Answer:** The following are required:

- Add a build dependency for `spring-cloud-starter-netflix-eureka-client` to the build file, `build.gradle`
- Add configuration, depending, most importantly, on the `eureka.client.serviceUrl.defaultZone` property, which is used to specify where you can find the Eureka server(s)

3. What is required to make it possible for a Spring Boot-based microservice to call another microservice that is registered in a Netflix Eureka server?

**Answer:** In addition to the answer provided in relation to question number 2 in this chapter, the following is also required:

1. That the HTTP client used is made load balancer-aware. Using the reactive `WebClient` class, this can be done with the following code:

```
@Bean
@LoadBalanced
public WebClient.Builder loadBalancedWebClientBuilder() {
    final WebClient.Builder builder = WebClient.builder();
    return builder;
}
```

2. Using URL hostnames that are the same as the microservice application name, as specified in its property; that is, `spring.application.name`. This is the name that the microservice will be known by in the Eureka server.

4. Let's assume that you have a Netflix Eureka server up and running, along with one instance of microservice *A* and two instances of microservice *B*. All microservice instances register themselves with the Netflix Eureka server. Microservice *A* makes HTTP requests to microservice *B* based on the information it gets from the Eureka server. What will happen if, in turn, the following happens?
- The Netflix Eureka server crashes
  - One of the instances of microservice *B* crashes
  - A new instance of microservice *A* starts up
  - A new instance of microservice *B* starts up
  - The Netflix Eureka server starts up again

**Answer:** After a while, thanks to the resilience capabilities built into Netflix Eureka, Netflix Ribbon, and Spring Cloud, the information regarding available microservices will be in sync and the two instances of microservice *A* will be able to utilize both of the instances of microservice *B*. The time it takes to synchronize the discovery information depends on the configuration used.

## Chapter 10

1. What are the elements used to build a routing rule in Spring Cloud Gateway called?

**Answer:** Predicates, filters, the destination URI, and an ID.

2. What are they used for?

**Answer:** They are used for the following purposes:

- **Predicates** select a route based on information in the incoming HTTP request
- **Filters** can modify both the request and/or the response
- A **destination URI** describes where to send a request
- An **ID** is the name of the route

3. How can we instruct Spring Cloud Gateway to locate microservice instances through a discovery service such as Netflix Eureka?

**Answer:** Replace the `http` and `https` protocols used in the URL with `lb`. By way of a hostname in the URL, use the name of the microservice, as registered in the Eureka server; in other words, the name specified in the `spring.application.name` property of the microservice.

4. In a Docker environment, how can we ensure that external HTTP requests to the Docker engine can only reach the edge server?

**Answer:** This can be achieved by only exposing the ports in the Docker host from the edge server. This means only using ports: in the Docker Compose files for the edge server container.

5. How do we change the routing rules so that the edge server accepts calls to the product-composite service on the `http://$HOST:$PORT/api/product` URL instead of the currently used `http://$HOST:$PORT/product-composite`?

**Answer:** Using the SetPath filter, we can replace the id: product-composite routing rule with a new route, such as the following:

```
- id: api-product-composite-1
  uri: lb://product-composite
  predicates:
    - Path=/api/product/{segment}
  filters:
    - SetPath=/product-composite/{segment}
```

This will cover all calls to `http://$HOST:$PORT/api/product/{productId}`, for example, GET and DELETE requests. However, it will not match a POST on `http://$HOST:$PORT/api/product`. To also support the POST request, we can add a second route:

```
- id: api-product-composite-2
  uri: lb://product-composite
  predicates:
    - Path=/api/product
  filters:
    - SetPath=/product-composite
```

Add the route to `spring-cloud/gateway/src/main/resources/application.yml`, restart the edge server, and try out the following requests:

```
HOST=localhost
PORT=8080
# Create a minimal product with productId = 123456789
curl -i http://$HOST:$PORT/api/product \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $ACCESS_TOKEN" \
-X POST --data '{"productId": 123456789}'
# Read the product with productId = 123456789
curl -i http://$HOST:$PORT/api/product/123456789 -H "Authorization:
Bearer $ACCESS_TOKEN"
```

```
# Delete the product with productId = 123456789
curl -i http://$HOST:$PORT/api/product/123456789 -H
"Authorization:
Bearer $ACCESS_TOKEN" -X DELETE
# Try read the product again, should fail with the HTTP status
"404 Not Found"
curl -i http://$HOST:$PORT/api/product/123456789 -H
"Authorization: Bearer $ACCESS_TOKEN"
```

Expect all requests, except the last one, to return the 200 OK HTTP status; the last request should return the 404 Not Found HTTP status.

## Chapter 11

1. What are the benefits and shortcomings of using self-signed certificates?

**Answer:** The benefits and shortcomings are listed as follows:

- **Benefits:** They are convenient to use during development and tests since you can create them yourself.
- **Shortcomings:** Web browsers and mobile devices don't trust them, so they can't be used in production.

2. What is the purpose of OAuth 2.0 authorization codes?

**Answer:** Authorization codes are used in an authorization code grant flow as one-time passwords to increase security compared to an implicit grant flow. When using an authorization code grant flow, it is only the authorization code that is visible in the web browser; as soon as it is used by the backend code in exchange for an access token, it is invalidated. So, even if it is stolen in the web browser, it is of little use to an attacker. The attacker also needs access to the client secret to be able to use it.

3. What is the purpose of OAuth 2.0 scopes?

**Answer:** They can be used as time-constrained access rights.

4. What does it mean when a token is a JWT?

**Answer:** JWTs, also known as JSON Web Tokens, are an open standard (<https://tools.ietf.org/html/rfc7519>) for sending information in a token.

5. How can we trust the information that is stored in a JWT?

**Answer:** JWTs can be signed by the sender, which means that the receiver of a JWT can trust the content in the token, given that the signature is validated by the receiver.

6. Is it suitable to use the OAuth 2.0 authorization code grant flow with a native mobile app?

**Answer:** In general, no, since a native mobile app can't protect the client secret that needs to be used in an authorization code grant flow. This means that in practice, it does not provide any increased security over the simpler implicit grant flow.

However, if an OAuth 2.0 authorization code grant flow is used by a native mobile app together with RFC8252 – OAuth 2.0 for Native Apps (<https://tools.ietf.org/html/rfc8252>), the level of security is increased, so it makes sense to use an OAuth 2.0 authorization code grant flow with a native mobile app. RFC8252 – OAuth 2.0 for Native Apps relies on the use of RFC 7636: Proof Key for Code Exchange (<https://tools.ietf.org/html/rfc7636>).

7. What does OpenID Connect add to OAuth 2.0?

**Answer:** OpenID Connect enables client applications to verify the identity of users.

## Chapter 12

1. What API call can we expect from a review service to the config server during startup to retrieve its configuration?

**Answer:** Given that we are running the microservices in Docker, we can expect the following call:

```
http://${username}:${password}@config-server:8888/review/docker
```

The review service was started up using the `docker compose up -d` command.

2. What configuration information should we expect back from an API call to the config server using the following command?

```
curl https://dev-usr:dev-pwd@localhost:8443/config/application/default -ks | jq
```

**Answer:** We can expect the configuration that applies to all microservices using the default Spring profile: basically, the content in the `application.yml` file that does not belong to a specific Spring profile.

3. What types of repository backend does Spring Cloud Config support?

**Answer:** For a full list, see [https://cloud.spring.io/spring-cloud-config/reference/html/#\\_environment\\_repository](https://cloud.spring.io/spring-cloud-config/reference/html/#_environment_repository). Some of the most commonly used are:

- Git repositories
- Local filesystem
- HashiCorp Vault
- JDBC databases

Other backend integrations also exist, for example, <https://docs.aws.amazon.com/spring-cloud-aws/docs/current/reference/html/index.html>.

4. How can we encrypt sensitive information on disk using Spring Cloud Config?

**Answer:** The configuration server supports encryption of configuration information when stored on disk. The config server supports the use of both symmetric and asymmetric keys. Asymmetric keys are more secure but harder to manage. The configuration server provides endpoints for encrypting and decrypting information: `/config/encrypt` and `/config/decrypt`. Access to these endpoints must be locked down before being used in production.

5. How can we protect the config server API from misuse?

**Answer:** By using HTTPS to prevent eavesdropping, and HTTP Basic authentication to ensure that the API user is a known client.

6. Mention some pros and cons for clients that connect to the config server first as opposed to those that connect to the discovery server first.

**Answer:** The pros and cons are as follows.

Connecting to the config server first:

- **Pros:** This is the default behavior and easy to configure. It requires clients to have a local configuration for how to connect to the configuration server. The remainder of the configuration information will be retrieved from the configuration server, including a configuration for how to communicate with the discovery server. Also, the discovery server can store its configuration in the config server.

- **Cons:** Only partial support for the high-availability configuration of multiple configuration servers. Clients can be configured with multiple URLs in their connect URL for the configuration server, supporting basic failover. But there will be no surveillance of the health of the configuration servers, unlike when using a discovery server or in a container orchestrator, as we will see in *Chapter 15, Introduction to Kubernetes*. This means that clients can call a configuration server that is unhealthy and will return an error (for example, a 500 – Internal Server Error), and the client will not be able to retrieve its configuration.

Connecting to the discovery server first:

- **Pros:** This scenario supports the high-availability configuration of multiple configuration servers. Multiple configuration servers can register in the discovery server, and the discovery server will monitor the health of the registered configuration servers, as with any registered service.
- **Cons:** This scenario requires a more complex local configuration of the clients. The clients must have both a local configuration for how to connect to the discovery server and some configuration (except for URLs) for how to connect to the configuration server.

## Chapter 13

1. What are the states of a circuit breaker and how are they used?

**Answer:** The states are as follows:

- **Closed:** This is the state for normal operation; all requests are sent through the circuit breaker.
- **Open:** Too many failures have been detected and no requests are allowed. The circuit breaker will perform fast failure logic. This means that it doesn't wait for a new fault, for example, a timeout, to happen on subsequent requests. Instead, it redirects the call directly to a fallback method.
- **Half-open:** The circuit breaker allows a new request to be sent through to see whether the issue that caused the failures has disappeared.



2. How can we handle timeout errors in the circuit breaker?

**Answer:** The circuit breaker in Resilience4j is triggered by exceptions, not by a timeout itself. Therefore, we have to add code that generates an exception after a timeout. Using the `WebClient` class, we can do that by using its `timeout(Duration)` method.

3. How can we apply fallback logic when a circuit breaker fails fast?

**Answer:** We can catch the exception, `CircuitBreakerOpenException`, that is thrown by the circuit breaker when it is open and call a fallback method.

4. How can a retry mechanism and a circuit breaker interfere with one another?

**Answer:** If a circuit breaker is configured to open the circuit too soon, the retry mechanism might not get a chance to perform all the retries it is configured to perform before giving up.

5. Provide an example of a service that you can't apply a retry mechanism to.

**Answer:** A service that is not idempotent (by which we mean, calling the service one or many times with the same request parameters) does not give the same result. This could, for example, be a service for creating orders. If an order creation service doesn't have logic for detecting retry attempts, it may create multiple orders instead of just one.

## Chapter 14

1. What configuration parameter is used to control how trace information is sent to Zipkin?

**Answer:** `spring.zipkin.sender.type`. It takes the `web`, `rabbit`, and `kafka` values, where `web` is the default value.

2. What is the purpose of the `spring.sleuth.sampler.probability` configuration parameter?

**Answer:** It is used to limit the number of spans that are sent to Zipkin. By default, it is set to `0.1`; in other words, 10% of all spans are sent to Zipkin. This can be a sensible value in production environments to avoid flooding Zipkin but, for development, we often want all spans to be sent to Zipkin by setting this value to `1.0`.

3. How can you identify the longest-running request after executing the `test-em-all.bash` test script?

**Answer:** By setting the Sort parameter to Longest First in the Zipkin UI.

4. How can we find requests that have been interrupted by the timeout introduced in *Chapter 13, Improving Resilience Using Resilience4j*?

**Answer:** If you set the Sort parameter to Longest First, you will find them at the top of the list.

However, you can also be a bit more precise in your search and specify the following:

- Service Name: A11
- Annotation Query: error=CANCELLED
- Duration: 2s

Note that it is the second `product-composite` span that has the error field set to CANCELLED, so it is important that you don't search for the gateway service only!

5. What does the trace look like for an API request when the circuit breaker introduced in *Chapter 13, Improving Resilience Using Resilience4j*, is open?

**Answer:** We expect a short response time when the circuit breaker is open; in other words, when fast fallback to the fallback method has been applied. Therefore, perform a search with the following parameter settings:

- Service Name: A11
- Annotation Query: error=CANCELLED
- Sort: Shortest First

The first two traces in the response should come from a request with the circuit breaker open. Click on one of them. They should appear as follows:

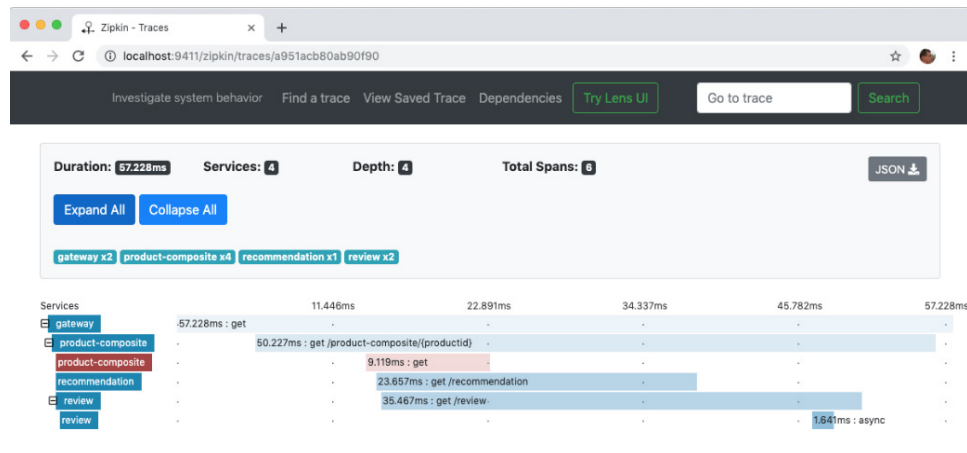


Figure 1: Observing a trace

Note that the call to the product service failed, due to the open circuit breaker, but the request succeeded thanks to the fallback method applied by the product-composite service!

- How can we locate APIs that failed due to the caller not being authorized to perform the request?

**Answer:** Search for `error=403` in the Annotation Query parameter. You will find an HTTP DELETE request that failed due to the user not having enough privileges – in other words, the expected OAuth Scope.

## Chapter 15

- What happens if you run the same `kubectl create` command twice?

**Answer:** The command fails and returns an `AlreadyExists` error.

- What happens if you run the same `kubectl apply` command twice?

**Answer:** The command succeeds and returns information to the effect that the affected objects are unchanged.

- In terms of questions 1 and 2, why do they act differently the second time they are run?

**Answer:** The `create` command is imperative. This means that it tells Kubernetes exactly what to do. The `apply` command is declarative; this means that it is up to Kubernetes to find out what to do if anything at all needs to be done, as in this case.

4. What is the purpose of a ReplicaSet, and what other resource creates a ReplicaSet?

**Answer:** A ReplicaSet is used to ensure that a specified number of Pods are running at all times. If a Pod is deleted, it will be replaced with a new Pod by ReplicaSet. A ReplicaSet can be created by a Deployment object.

5. What is the purpose of etcd in a Kubernetes cluster?

**Answer:** It is used to store the configuration of the Kubernetes cluster.

6. How can a container find out the IP address of another container that runs in the same Pod?

**Answer:** Containers in the same Pod share the same network namespace, meaning that they have the same IP address and port space. They can communicate with each other using the localhost hostname, or the IP address 127.0.0.1. They have to ensure that they don't use the same ports.

7. What happens if you create two Deployments with the same name but in different namespaces?

**Answer:** This works perfectly fine. For example, two developers can run their own versions of a group of Deployments and Pods in two different namespaces in the same Kubernetes cluster.

8. What configuration of two services with the same name can make them fail, even if they are created in two different namespaces?

**Answer:** If the services are of the NodePort type and try to allocate the same node port, they will conflict and the creation of the second service will fail.

## Chapter 16

1. Why did we remove the Eureka server from the microservice landscape when deploying it on Kubernetes?

**Answer:** Kubernetes comes with a built-in discovery service. This makes it unnecessary to deploy a separate discovery service such as Netflix Eureka.

2. What did we replace the Eureka server with and how was the source code of the microservices affected by this change?

**Answer:** The Eureka server was replaced by built-in Kubernetes Service objects and the kube-proxy runtime component.

The source code in the microservices themselves was not affected by this change. However, the build dependencies and configuration were changed on account of the following:

- We removed the dependency on `spring-cloud-starter-netflix-eureka-client` in all the build files; that is, `build.gradle`
- Netflix Eureka-specific configuration has been removed from the configuration, in the `config-repo` repository
- We removed the `eureka.client.enabled=false` property setting, which was no longer required, from all integration tests

3. What's the purpose of liveness and readiness probes?

**Answer:** Their purposes are as follows:

- Liveness probes are used by Kubernetes to determine whether a container needs to be restarted
- Readiness probes are used by Kubernetes to determine whether a container is ready to accept incoming requests

4. How is Spring Boot's mechanism for graceful shutdown useful?

**Answer:** When applying graceful shutdown, a microservice stops accepting new requests and waits for a configurable time for active requests to complete before it shuts down the application. Requests that take a longer time to complete than the shutdown wait period will be aborted.

5. What is the purpose of the following Helm template directives?

```
{{- $common := dict "Values" .Values.common -}}
{{- $noCommon := omit .Values "common" -}}
{{- $overrides := dict "Values" $noCommon -}}
{{- $noValues := omit . "Values" -}}
{{- with merge $noValues $overrides $common -}}
```

**Answer:** It implements an override mechanism so that a parent chart can override values from a subchart that it uses.

6. Why would the following named Helm template fail?

```
{{- define "common.secrets" -}}
{{- range $secretName, $secretMap := .Values.secrets }}
  apiVersion: v1
  kind: Secret
  metadata:
    name: {{ $secretName }}
```

```
labels:
  app.kubernetes.io/name: {{ $secretName }}
type: Opaque
data:
{{- range $key, $val := $secretMap }}
  {{ $key }}: {{ $val | b64enc }}
{{- end }}
{{- end -}}
{{- end -}}
```

**Answer:** The template will iterate over the `.Values.secrets` variable and create a Secret manifest for each element in the variable. To produce a valid YAML output, each manifest declaration must be separated with `---`. A line containing `---` must be added before the two last end directives. For a working code example, see `$BOOK_HOME/Chapter16/kubernetes/helm/common/templates/_secrets.yaml`.

7. Why would the following manifests not work together?

```
apiVersion: v1
kind: Service
metadata:
  name: review
  labels:
    app.kubernetes.io/name: review
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
  selector:
    app.kubernetes.io/pod-name: review
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: review
  labels:
    app.kubernetes.io/name: review
spec:
  replicas: 1
```

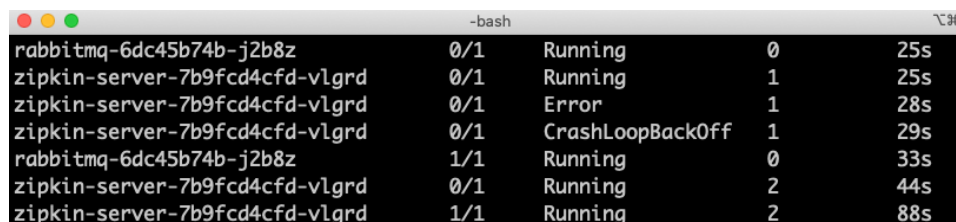
```

selector:
  matchLabels:
    app.kubernetes.io/name: review
template:
  metadata:
    labels:
      app.kubernetes.io/name: review
  spec:
    containers:
      - name: review
        image: "hands-on/review-service:latest"
        ports:
          - containerPort: 80
            name: http-port
            protocol: TCP

```

**Answer:** Each Pod that will be created based on the Deployment manifest's template definition will be labeled with a label named `app.kubernetes.io/name`, which will have the value `review`. The Service manifest declares its selector based on a label named `app.kubernetes.io/pod-name`. This mismatch in label names will result in the Service object not being able to select any of the Pods created based on the Deployment manifest. To correct this error, the label name used by the Service object's selector should be changed to `app.kubernetes.io/name`.

8. What is going on here?



The image shows a terminal window with a dark background. At the top, there are three colored circles (red, yellow, green) and the text '-bash'. Below this, there is a table of pod status information. The table has five columns: pod name, restarts, phase, container ID, and age. The rows show the following data:

Pod Name	Restarts	Phase	Container ID	Age
rabbitmq-6dc45b74b-j2b8z	0/1	Running	0	25s
zipkin-server-7b9fcd4cfd-vlgrd	0/1	Running	1	25s
zipkin-server-7b9fcd4cfd-vlgrd	0/1	Error	1	28s
zipkin-server-7b9fcd4cfd-vlgrd	0/1	CrashLoopBackOff	1	29s
rabbitmq-6dc45b74b-j2b8z	1/1	Running	0	33s
zipkin-server-7b9fcd4cfd-vlgrd	0/1	Running	2	44s
zipkin-server-7b9fcd4cfd-vlgrd	1/1	Running	2	88s

Figure 2: What is going on here?

**Answer:** The output tells us that the following happened:

1. The first two lines show that Pods running RabbitMQ and Zipkin are starting up
2. The two following lines show that the Zipkin Pod crashes before it is up and running
3. On the fifth line, the RabbitMQ Pod reports as up and running

4. The last two lines show that the Zipkin server has started up again, and this time, the startup succeeds

What is probably going on here is that the Zipkin server requires access to RabbitMQ during its start up phase. If it fails to connect to RabbitMQ during startup, it simply crashes. Kubernetes will detect this and restart it. Once the Zipkin service has been restarted, the RabbitMQ service has become available, so the Zipkin server will start up with no problems this time.

## Chapter 17

1. How was the Spring Cloud Config Server replaced by Kubernetes resources?

**Answer:** It was replaced by the built-in Kubernetes secrets and config-maps objects.

2. How was the Spring Cloud Gateway replaced by Kubernetes resources?

**Answer:** It was replaced by a built-in Kubernetes Ingress object managed by a Nginx-based Ingress controller. The Ingress controller was deployed using the `minikube addons enable ingress` command in *Chapter 15, Introduction to Kubernetes*; refer to the *Creating a Kubernetes cluster* section.

3. What is required to make the cert-manager automatically provision certificates for an Ingress object?

**Answer:** At least one issuer must be declared, and the ingress manifest must be annotated with `cert-manager.io/issuer`, which specifies which issuer to use.

4. How can the retention time of a certificate be checked and updated?

**Answer:** By inspecting and updating the certificate object created by the cert-manager. To inspect a certificate, the `kubectl describe certificate` command can be used. To update it, the `kubectl patch certificate` command can be used.

5. Where is the actual certificate stored?

**Answer:** It is stored as a TLS Secret.

6. Why did we run the tests using Docker Compose?

**Answer:** To verify that the microservices work without Kubernetes, in other words, that the microservice source code isn't locked into Kubernetes.



## Chapter 18

1. What is the purpose of a proxy component in a service mesh?

**Answer:** The proxy component is injected into each Pod that is part of the service mesh and controls its incoming and outgoing traffic. The proxy component also collects telemetry data.

2. What's the difference between a **control plane** and a **data plane** in a service mesh?

**Answer:** The **data plane** consists of proxy components together with gateway components that handle external traffic to and from the service mesh. The **control plane** manages the components in the data plane and also collects metrics and telemetry data used for monitoring and observation.

3. What is the `istioctl kube-inject` command used for?

**Answer:** It is used to inject Istio's proxy components into Pods – in other words, to add an Istio proxy component to a Pod as a sidecar.

4. What is the `minikube tunnel` command used for?

**Answer:** It is used to simulate the load balancer of a Minikube instance, and to expose the services of the LoadBalancer type. On macOS, it can also be used to resolve internal DNS names for Kubernetes services on the host where the Minikube instance runs.

5. What tools is Istio integrated with for observability?

**Answer:** Kiali, Jaeger, and Grafana.

6. What configuration is required to make Istio protect communication within the service mesh using mutual authentication?

**Answer:** Both the client and server sides need to agree on using mutual authentication. The server side is controlled by a `Policy` object that can be set to be either `STRICT` or `PERMISSIVE` (in other words, allowing both plaintext and HTTPS with mutual authentication). Clients are configured using `DestinationRule` objects, where the TLS mode can be set to `ISTIO_MUTUAL`.

7. What can the `abort` and `delay` elements in a virtual service be used for?

**Answer:** They can be used to test resilience in deployed microservices by injecting faults into Istio's proxy components.

8. What configuration is required to set up a blue/green deploy scenario?

**Answer:** The configuration is as follows:

- VirtualService objects, which define weight distributions between requests sent to different subsets of a service
- DestinationRule objects, which define the version of the underlying Pods that a subset is defined by – in other words, to which the traffic is routed

## Chapter 19

1. A user searched for ERROR log messages in the hands-on namespace for the last 30 days using the search criteria shown in the following screenshot, but none were found. Why?

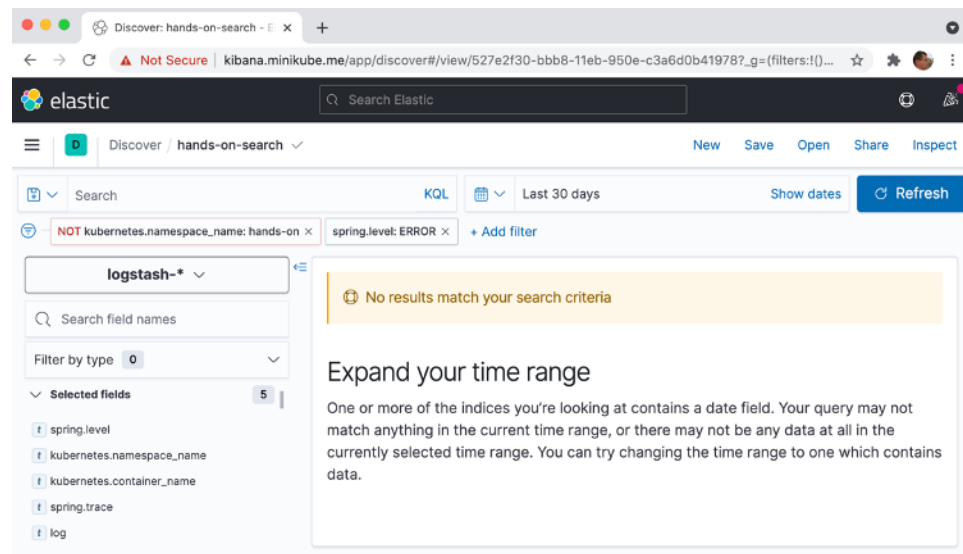


Figure 3: Kiali web UI – Not showing the expected records

**Answer:** The user has, by mistake, chosen to exclude the hands-on namespace instead of including it.

2. A user has found a log record of interest. How can the user find related log records from this and other microservices, for example, that come from processing an external API request?

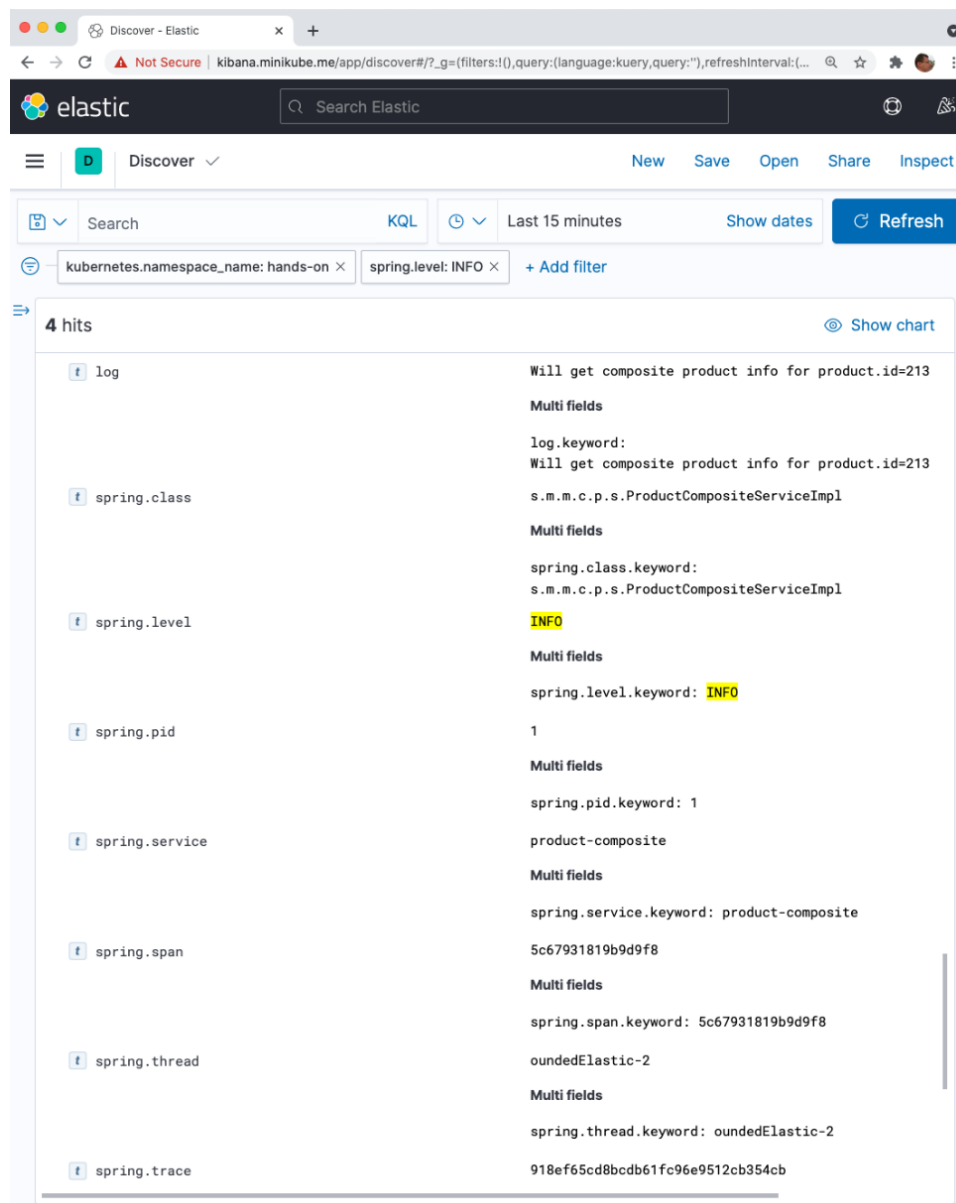


Figure 4: Kiali web UI - How do we find related log records?

**Answer:** Use the `spring.trace` field in the log record, which contains the trace ID, as a correlation ID to search for other log records that also contain the same value in their `spring.trace` field.

3. A user has found a log record that seems to indicate the root cause of a problem reported by an end user. How can the user find the stack trace that shows where the error occurred in the source code?



Figure 5: Kiali web UI – How do we find the root cause?

**Answer:** Click on the **View surrounding documents** link to find log records with stack traces reported in near time to the log record.

4. Why doesn't the following Fluentd configuration element work?

```
<match kubernetes.**hands-on*>
  @type rewrite_tag_filter
  <rule>
    key log
    pattern ^(.*)$
    tag spring-boot.${tag}
  </rule>
</match>
```

**Answer:** The match pattern, `kubernetes.**hands-on*`, is incorrect. It must end with `**` to match zero, one, or more parts of the tag field in the log record after `hands-on`. `*` will only match precisely one tag part.

5. How can you determine whether Elasticsearch is up and running?

**Answer:** Run the following command:

```
curl https://elasticsearch.minikube.me -sk | jq -r .tagline
```

Expect the response `You Know, for Search`.

6. You suddenly lose connection to Kibana from your web browser. What could have caused this problem?

**Answer:** It may be that the `minikube tunnel` command has stopped running. Restart it if required.

## Chapter 20

1. What changes did we need to make to the source code in the microservices to make them produce metrics that are consumed by Prometheus?

**Answer:** The following changes were applied to the source code:

- A build dependency for the micrometer library was added to the build.gradle build file:

```
implementation("io.micrometer:micrometer-registry-  
prometheus")
```

- Annotations were added to the Pod templates in the Deployment objects to let Prometheus know where to find the metrics:

```
annotations:  
  prometheus.io/scrape: "true"  
  prometheus.io/port: "4004"  
  prometheus.io/scheme: http  
  prometheus.io/path: "/actuator/prometheus"
```

2. What is the management.metrics.tags.application config parameter used for?

**Answer:** It is used to tag the Prometheus metrics with the name of the microservice. The following configuration was added to the application.yml file in the configuration repository, config-repo:

```
management.metrics.tags.application: ${spring.application.name}
```

3. If you want to analyze a support case regarding high CPU consumption, which of the dashboards in this chapter would you start with?

**Answer:** The dashboard named **JVM (Micrometer)**, and look for high CPU consumption. The **Istio Mesh Dashboard** can also be of help in determining whether any specific service currently has longer response times than usual. It could indicate what services you should start to investigate. Also, the Kiali graph view may be of considerable help in getting an overview of the situation from an application view.

4. If you want to analyze a support case regarding slow API responses, which of the dashboards in this chapter would you start with?

**Answer:** Start with the **Istio Mesh Dashboard** to get an overview of what services appear to be slow. Follow up with the **Istio Service Dashboard** and **JVM (Micrometer)** dashboards to ascertain further details. Also, the Kiali graph view can be of considerable help in obtaining an overview of the situation from an application perspective.

5. Name a problem with counter-based metrics, such as Resilience4j's retry metrics. What can be done so that we can monitor them in a useful way?

**Answer:** Only the counter-based metric value has increased, making it hard to identify trends in a dashboard. The rate function in Grafana can be used to convert a counter metric into a *counts per second* metric, making it much easier to see changes in how the increased rate goes up or down.

6. What is going on here?

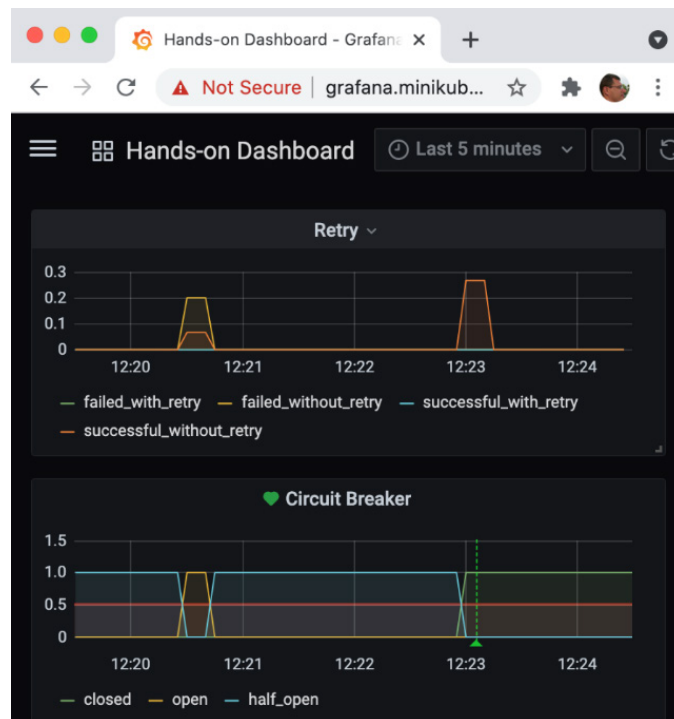


Figure 6: What is going on here?

**Answer:**

1. The circuit breaker is half open due to a problem detected earlier.
2. During the first burst of requests, the problem still remains. The circuit breaker therefore goes to the open state after some requests, and the last requests are processed immediately by the fallback logic (also known as **failing fast**). The circuit breaker goes back to the half open state after a while.
3. During the second burst of requests, the circuit breaker is closed. This means that the problem has been fixed and that requests are processed as normal again.

## Chapter 23

1. How are the Spring Native and GraalVM projects related to each other?

**Answer:** The Spring Native project uses the native image compiler provided by the GraalVM project.

2. How is the tracing agent used?

**Answer:** The tracing agent is used to simplify the creation of the configuration required by the native image compiler. It can observe a running Java application's behavior and based on the use of, for example, reflection and dynamic proxies, the tracing agent can create configuration files.

3. What is the difference between JIT and AOT compilation?

**Answer:** A Just in Time (JIT) compiler creates binary code from Java byte code at runtime. An Ahead of Time (AOT) compiler creates the binary code at build time.

4. What is a native hint?

**Answer:** It is a Java annotation provided by the Spring Native project that the Spring Native AOT plugin uses at build time to create configuration for the native compiler.

5. How are memory and startup times affected by native compiling Java code?

**Answer:** Startup times are significantly reduced, and memory usage is lowered (at least, the initial memory usage).

6. How does a Kubernetes Deployment object differ from a StatefulSet object?

**Answer:** A StatefulSet supports the use of **distributed stateful workloads**, where each Pod needs to have its own identity in terms of its own DNS name. If disks are attached to Pods of a StatefulSet using a PersistentVolumeClaim, each Pod gets its own disk (PersistentVolume).