Add instructor notes here.

Objective

- What is Actuator
- Endpoints
- Monitoring and Management over HTTP
- Profiles-Dev & Prod

Spring Boot-Actuator

- Spring Boot includes a number of additional features to help you monitor and manage your application when we push it to production. We can choose to manage and monitor our application by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to our application.
- The spring-boot-actuator module provides all of Spring Boot's production-ready features.
- To add the actuator to a Maven based project, add the following 'Starter' dependency:

```
<dependencies> <dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-actuator</artifactId>
</dependency> </dependencies>
```

Features-Actuator

- Endpoints Actuator endpoints allow us to monitor and interact with our application. Spring Boot includes a number of built-in endpoints and we can also add our own. For example the health endpoint provides basic application health information. Run up a basic application and look at /actuator/health.
- management.endpoints.web.base-path=/ add this properties to make it only /health,/info,/logger
- Metrics Spring Boot Actuator provides dimensional metrics by integrating with Micrometer.
- Audit Spring Boot Actuator has a flexible audit framework that will publish events to an AuditEventRepository. Once Spring Security is in play it automatically publishes authentication events by default. This can be very useful for reporting, and also to implement a lock-out policy based on authentication failures.

Actuator-Endpoints

- Actuator endpoints let us monitor and interact with our application. Spring Boot includes a number of built-in endpoints and lets us add our own. For example, the health endpoint provides basic application health information.
- Each individual endpoint can be enabled or disabled. This controls whether or not the endpoint is created and its bean exists in the application context.
- To be remotely accessible an endpoint also has to be exposed via JMX or HTTP.
- Most applications choose HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

Actuator-Endpoints

ID	Description
auditevents	Exposes audit events information for the current application. Requires an AuditEventRepository bean.
beans	Displays a complete list of all the Spring beans in your application.
caches	Exposes available caches.
conditions	Shows the conditions that were evaluated on configuration and auto-configuration classes and the reasons why they did or did not match.
configprops	Displays a collated list of all @ConfigurationProperties.
env	Exposes properties from Spring's ConfigurableEnvironment.
flyway	Shows any Flyway database migrations that have been applied. Requires one or more Flyway beans.
health	Shows application health information.
httptrace	Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges). Requires an HttpTraceRepository bean.
info	Displays arbitrary application info.
integrationgraph	Shows the Spring Integration graph. Requires a dependency on spring-integration-core.
loggers	Shows and modifies the configuration of loggers in the application.
liquibase	Shows any Liquibase database migrations that have been applied. Requires one or more Liquibase beans.
metrics	Shows 'metrics' information for the current application.
mappings	Displays a collated list of all @RequestMapping paths.
scheduledtasks	Displays the scheduled tasks in your application.
sessions	Allows retrieval and deletion of user sessions from a Spring Session-backed session store. Requires a Servlet-based web application using Spring Session.
shutdown	Lets the application be gracefully shutdown. Disabled by default.
threaddump	Performs a thread dump.

The following technology-agnostic endpoints are available:
IDDescriptionheapdump
Returns an hprof heap dump file.
jolokia
Exposes JMX beans over HTTP (when Jolokia is on the classpath, not available for WebFlux). Requires a dependency on jolokia-core.
logfile
Returns the contents of the logfile (if logging.file.name or logging.file.path properties have been set). Supports the use of the HTTP Range header to retrieve part of the log file's content.
prometheus
Exposes metrics in a format that can be scraped by a Prometheus server. Requires a dependency on micrometer-registry-prometheus.

Exposing Endpoints

- Endpoints may contain sensitive information, careful consideration should be given about when to expose them.
- To change which endpoints are exposed, use the following technology-specific include and exclude properties:

Property	Default
management.endpoints.jmx.exposure.exclude	
management.endpoints.jmx.exposure.include	*
management.endpoints.web.exposure.exclude	
management.endpoints.web.exposure.include	info, health

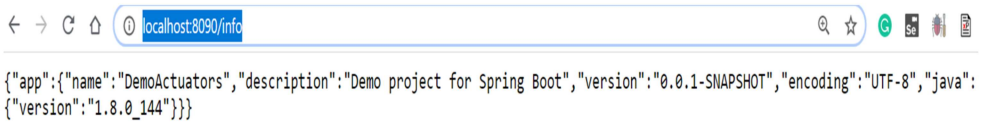
The include property lists the IDs of the endpoints that are exposed.
The exclude property lists the IDs of the endpoints that should not be exposed.
The exclude property takes precedence over the include property.
Both include and exclude properties can be configured with a list of endpoint IDs.
* can be used to select all endpoints. For example, to expose everything over HTTP except the env and beans endpoints, use the following properties:
management.endpoints.web.exposure.include=*
management.endpoints.web.exposure.exclude=env,beans

EndPoints

http://localhost:8090/info

```
#Actuators
management.endpoints.web.base-path=/
management.endpoints.web.exposure.include=*
# INFO ENDPOINT CONFIGURATION
info.app.name=@project.name@
info.app.description=@project.description@
info.app.version=@project.version@
info.app.encoding=@project.build.sourceEncoding@
info.app.java.version=@java.version@
```

Properties has to add in application.properties



{"app":{"name":"DemoActuators","description":"Demo project for Spring Boot","version":"0.0.1-SNAPSHOT","encoding":"UTF-8","java":{"version":"1.8.0_144"}}}

The info endpoint displays arbitrary information about your application. It obtains build information from META-INF/build-info.properties file, Git information from git.properties file. It also displays any information available in environment properties under the key info.

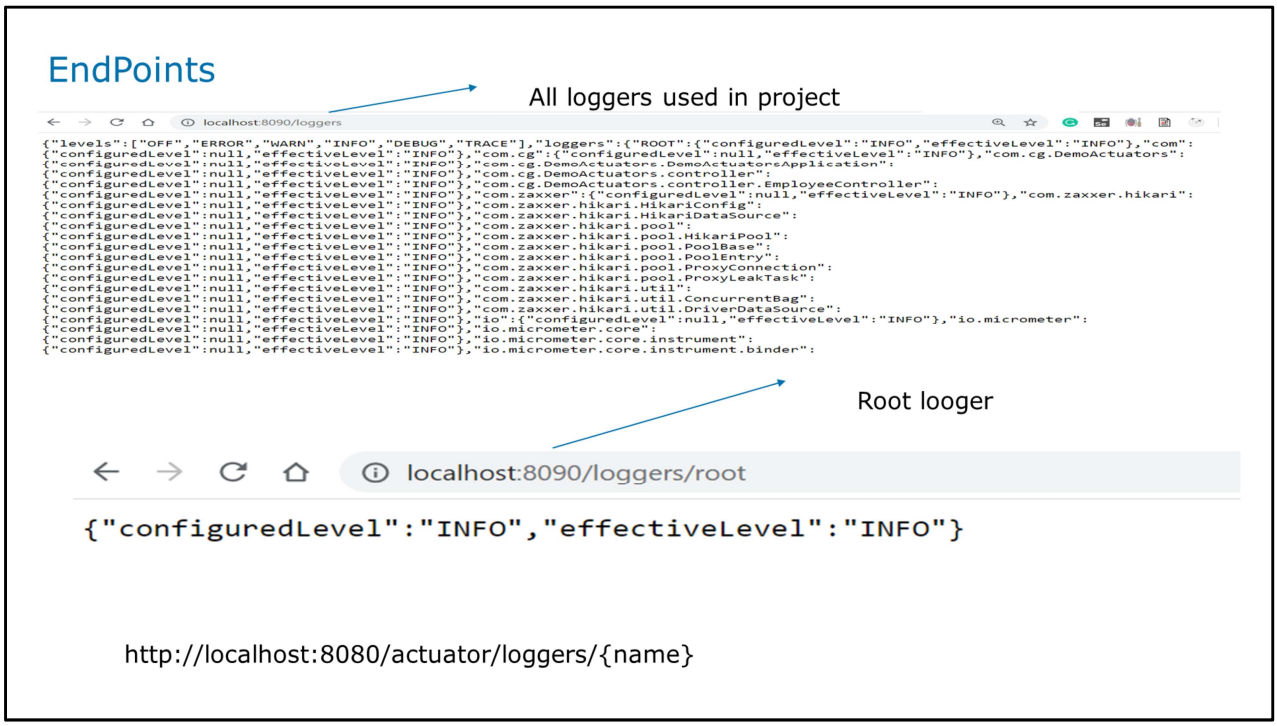
Note that, I'm using Spring Boot's Automatic property expansion feature to expand properties from the maven project.

EndPoints

http://localhost:8090/health


```
#Actuators
management.endpoints.web.base-path=/
management.endpoints.web.exposure.include=*
# HEALTH ENDPOINT
management.endpoint.health.show-details=always
```

The health endpoint checks the health of your application by combining several health indicators.
Spring Boot Actuator comes with several predefined health indicators like DataSourceHealthIndicator, DiskSpaceHealthIndicator, MongoHealthIndicator, RedisHealthIndicator, CassandraHealthIndicator etc. It uses these health indicators as part of the health check-up process.
For example, If your application uses Redis, the RedisHealthIndicator will be used as part of the health check-up. If your application uses MongoDB, the MongoHealthIndicator will be used as part of the health check-up and so on.
You can also disable a particular health indicator using application properties like so - management.health.mongo.enabled=false
But by default, all these health indicators are enabled and used as part of the health check-up process.
The health endpoint only shows a simple UP or DOWN status. To get the complete details including the status of every health indicator that was checked as part of the health check-up process, add the following property in the application.properties file -The health endpoint now includes the details of the DiskSpaceHealthIndicator which is run as part of the health checkup process.



The loggers endpoint, which can be accessed at http://localhost:8080/actuator/loggers, displays a list of all the configured loggers in your application with their corresponding log levels.

The loggers endpoint also allows you to change the log level of a given logger in your application at **runtime**.

For example, To change the log level of the root logger to DEBUG at runtime, make a POST request to the URL http://localhost:8080/actuator/loggers/root

Demo

- DemoActuators

Spring Profiles

- Spring Profiles provide a way to segregate parts of our application configuration and make it be available only in certain environments. Any @Component, @Configuration or @ConfigurationProperties can be marked with @Profile to limit when it is loaded.
- We can use a spring.profiles.active Environment property to specify which profiles are active.
- The spring.profiles.active property follows the same ordering rules as other properties: The highest PropertySource wins. This means that you can specify active profiles in application.properties.
- Profile-specific variants of both application.properties (or application.yml) and files referenced through @ConfigurationProperties are considered

Spring Profiles

application.properties

spring.profiles.active=prod

Profiles can be **prod or dev** if its prod it will load application-prod.properties & if dev it will load application-dev.properties

application-dev.properties

```
server.port=9091
logging.level.root=WARN
logging.level.com.cg=INFO
# H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2
# Datasource
spring.datasource.url=jdbc:h2:file:~/test
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
```

On Dev mode different port number, different logger, different databases

Spring Profiles

application-prod.properties

```
server.port=9092
logging.level.root=WARN
logging.level.com.cg=TRACE

spring.datasource.url =jdbc:mysql://localhost:3306/mumbai
spring.datasource.username =root
spring.datasource.password =123456


## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

For prod it will run on different port,database & level

Spring Profiles

@Profile("dev")

By using @Profile we can set profile bean will create at time of dev

```
@Profile("dev")
public class DevlopmentConfig {

@Bean
public String getDataDev() {

return "for Devlopment";

}

}
```

Spring Profiles

@Profile(“prod")

By using @Profile we can set profile bean will create at time of prod

```
@Profile("prod")
public class ProductionConfig {
@Bean
public String getData() {

return "for production";

}

}
```

Demo

- DemoConfigurationManagement

Lab

Lab3

- What is Actuator
- Endpoints
- Monitoring and Management over HTTP
- Profiles-Dev & Prod