

Adding an HTTP API and Email Capabilities with Conditional Configuration and Beans



Federico Mestrone
Software Engineer and Training Consultant

@fedmest www.federicomestrone.com



Library Overview



In this module:

- We will add web endpoints
 - For an HTTP-based PDF API
- We will add email capabilities
 - To send PDF attachments
- But only if the right dependencies, classes or configuration are in place
 - Using Spring Conditionals



nekosoft-pdffer-explorer

pdfferErrorController

pdfferExplorerController

pdffer-core

pdfferProducerBean

pdfferRegistryBean

pdffer-template

PDF Template

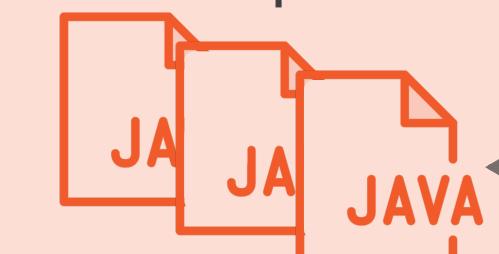


nekosoft-pdffer-templates

default

invoice

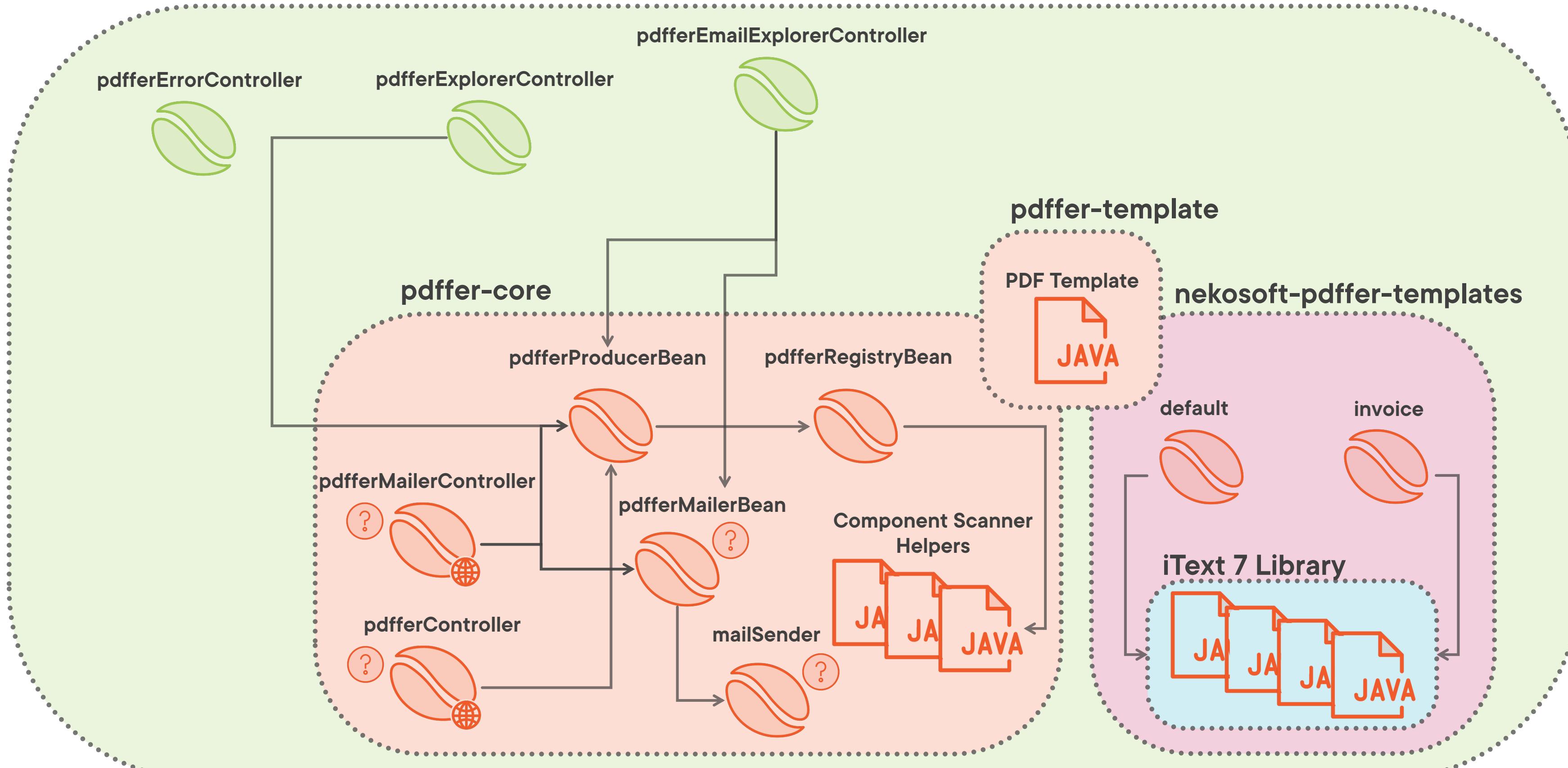
Component Scanner
Helpers



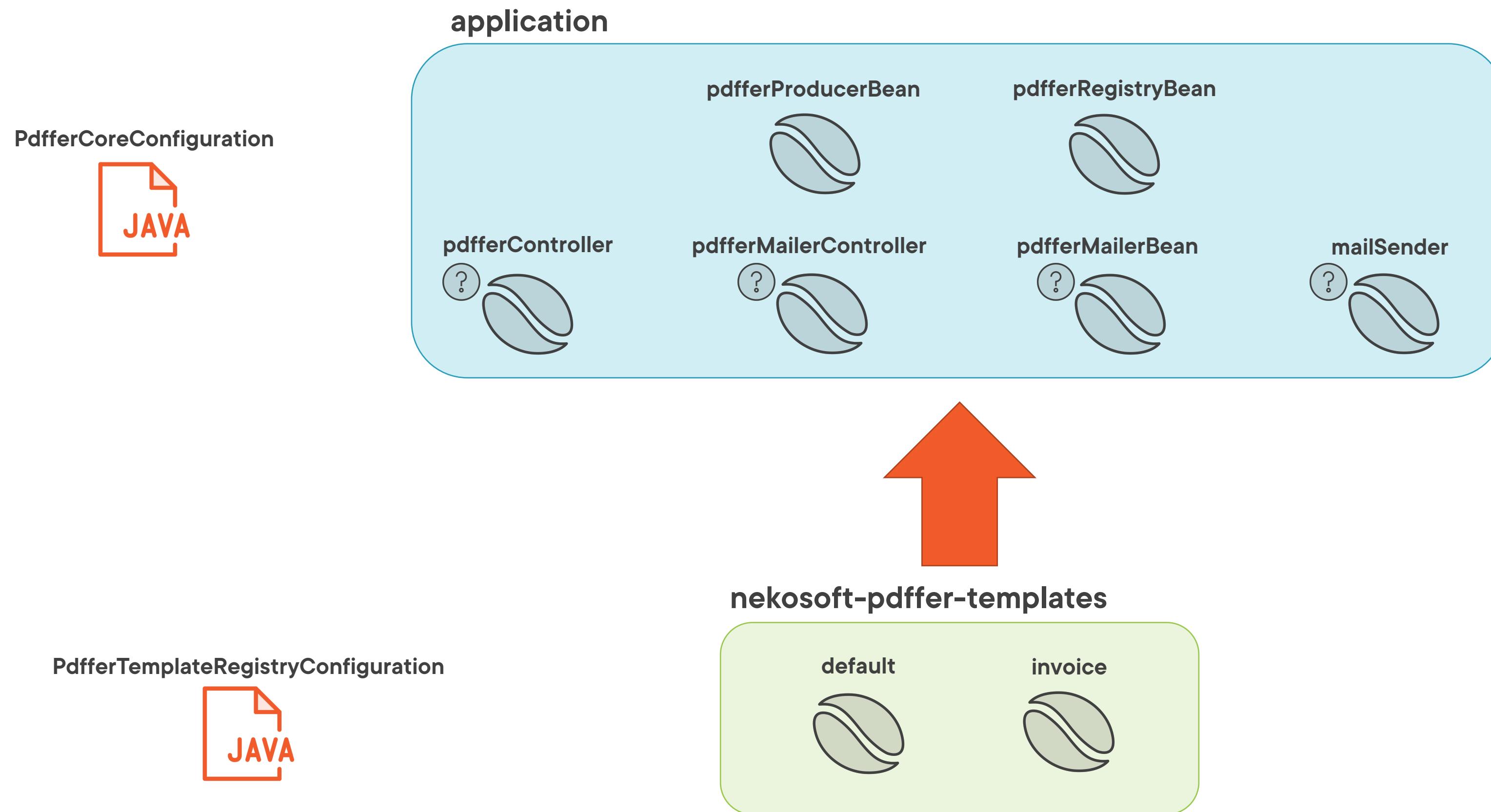
iText 7 Library



nekosoft-pdffer-explorer



Updated PDFfer Context



Conditional Beans



Allow you to define beans that
will be included only if the
environment offers what they
need in order to run properly





Conditionals can be used whenever you like

- But they make a lot of sense with Spring Autoconfiguration

You don't know anything about the final application

- Is it a web application?
- Is it a command line application?
- Does it include certain libraries?

Therefore you want to be able to add beans based on what you find at run-time





Conditions on beans

- `@ConditionalOnBean`

Conditions on files and classes

- `@ConditionalOnClass`
- `@ConditionalOnResource`

Conditions on environment and set-up

- `@ConditionalOnWebApplication`
- `@ConditionalOnJava`



More on Conditionals

Other conditions

There are several other built-in conditional annotations

Where to apply

Conditions can be applied to an entire configuration class, or to individual Spring beans



Conditional Configuration and Beans

```
@Configuration
@ConditionalOnResource(resources = "customlog.config")
public class CustomLoggerConfiguration {
    @ConditionalOnBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")
    @Bean String loggerCustomBeanName() {
        return CustomLoggerConfiguration.class.getCanonicalName();
    }
    @ConditionalOnMissingBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")
    @Bean Logger loggerCustomBean() {
        return Logger.getLogger(CustomLoggerConfiguration.class.getCanonicalName());
    }
    @ConditionalOnClass(name = "org.nk.CoolClass")
    @Bean CoolBean coolBean() {
        return new CoolBean();
    }
}
```



Conditional Configuration and Beans

```
@Configuration
@ConditionalOnResource(resources = "customlog.config")
public class CustomLoggerConfiguration {
    @ConditionalOnBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")
    @Bean String loggerCustomBeanName() {
        return CustomLoggerConfiguration.class.getCanonicalName();
    }
    @ConditionalOnMissingBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")
    @Bean Logger loggerCustomBean() {
        return Logger.getLogger(CustomLoggerConfiguration.class.getCanonicalName());
    }
    @ConditionalOnClass(name = "org.nk.CoolClass")
    @Bean CoolBean coolBean() {
        return new CoolBean();
    }
}
```



Conditional Configuration and Beans

```
@Configuration  
@ConditionalOnResource(resources = "customlog.config")  
public class CustomLoggerConfiguration {  
    @ConditionalOnBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")  
    @Bean String loggerCustomBeanName() {  
        return CustomLoggerConfiguration.class.getCanonicalName();  
    }  
    @ConditionalOnMissingBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")  
    @Bean Logger loggerCustomBean() {  
        return Logger.getLogger(CustomLoggerConfiguration.class.getCanonicalName());  
    }  
    @ConditionalOnClass(name = "org.nk.CoolClass")  
    @Bean CoolClass coolClass() {  
        return new CoolClass();  
    }  
}
```



Conditional Configuration and Beans

```
@Configuration  
@ConditionalOnResource(resources = "customlog.config")  
public class CustomLoggerConfiguration {  
    @ConditionalOnBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")  
    @Bean String loggerCustomBeanName() {  
        return CustomLoggerConfiguration.class.getCanonicalName();  
    }  
    @ConditionalOnMissingBean(name = "loggerBean", type = "org.nk.CustomLoggingConfig")  
    @Bean Logger loggerCustomBean() {  
        return Logger.getLogger(CustomLoggerConfiguration.class.getCanonicalName());  
    }  
    @ConditionalOnClass(name = "org.nk.CoolClass")  
    @Bean CoolClass coolClass() {  
        return new CoolClass();  
    }  
}
```

CoolClass does not exist
= Configuration class cannot be loaded



Conditional Beans

```
@Configuration  
@ConditionalOnClass(name = "org.nk.CoolClass")  
public class CoolClassLoggerConfiguration {  
  
    @Bean  
    CoolClass coolClass() {  
        return new CoolClass();  
    }  
}
```



Conditional Beans

```
@Component
@ConditionalOnJava(
    range = ConditionalOnJava.Range.EQUAL_OR_NEWER,
    value = JavaVersion.NINE
)
public class CustomLoggingConfig {
    /* ... */
}
```



Conditional Beans

```
@Component
@ConditionalOnJava(
    range = ConditionalOnJava.Range.OLDER_THAN,
    value = JavaVersion.NINE
)
public class LegacyCustomLoggingConfig {
    /* ... */
}
```



Conditional Controllers and Controller Methods

```
@ConditionalOnWebApplication
@RestController
public class CustomLoggingController {

    @GetMapping("loggerName")
    public String getLoggerNameEndpoint() {
        return CustomLoggerConfiguration.class.getCanonicalName();
    }

    @GetMapping("coolLogger")
    public CoolBean getCoolLoggerEndpoint(CoolBean coolBean) {
        return coolBean;
    }
}
```



More on Conditionals

Other conditions
There are several other built-in conditional annotations

Where to apply
Conditions can be applied to an entire configuration class, or to individual Spring beans

Custom conditions
You can also create your own custom condition classes and use them with the `@Condition` annotation



Custom Conditions

```
public class SecurityTokenCondition implements Condition {  
  
    @Override  
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {  
        Resource r = context.getResourceLoader().getResource("classpath:sectoken");  
        try {  
            if (r.exists() && r.isFile() && r.isReadable()) {  
                try (Reader d = new InputStreamReader(r.getInputStream())) {  
                    return "NEKOPDFFER".equals(FileCopyUtils.copyToString(d));  
                }  
            }  
            return false;  
        } catch (IOException e) {  
            return false;  
        }  
    }  
}
```



Using a Custom Condition

```
@Configuration  
@ComponentScan  
@Conditional(SecurityTokenCondition.class)  
public class SecureConfiguration {  
}
```



Gradle Features



Gradle Features

```
java {  
    registerFeature('web') {  
        usingSourceSet(sourceSets.main)  
    }  
    registerFeature('email') {  
        usingSourceSet(sourceSets.main)  
    }  
}  
  
dependencies {  
    // ...  
    webApi 'org.springframework.boot:spring-boot-starter-web'  
    emailApi 'org.springframework.boot:spring-boot-starter-mail'  
}
```



Using Gradle Features

```
dependencies {  
    // ...  
    implementation project(':pdffer-core')  
  
    implementation(project(':pdffer-core')) {  
        capabilities {  
            requireCapability('org.nekosoft.pdffer:pdffer-core-web')  
        }  
    }  
}  
}
```



Conditions Evaluation Report

A detailed list of all conditional beans that were evaluated in the application context, with the result of the evaluation and the reason for that result.



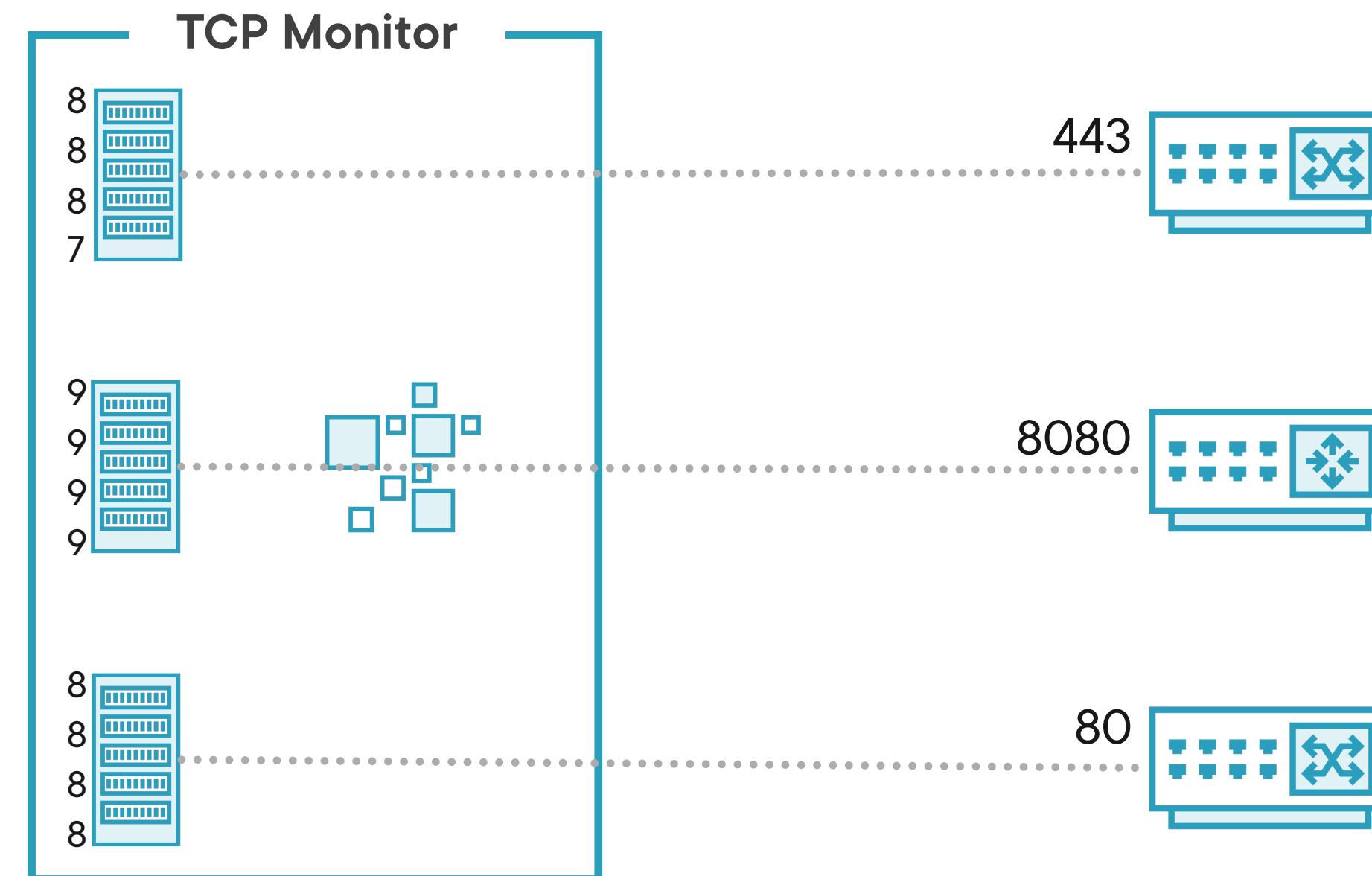
Conditions Evaluation Report

A detailed list of all conditional beans that were evaluated in the application context, with the result of the evaluation and the reason for that result.

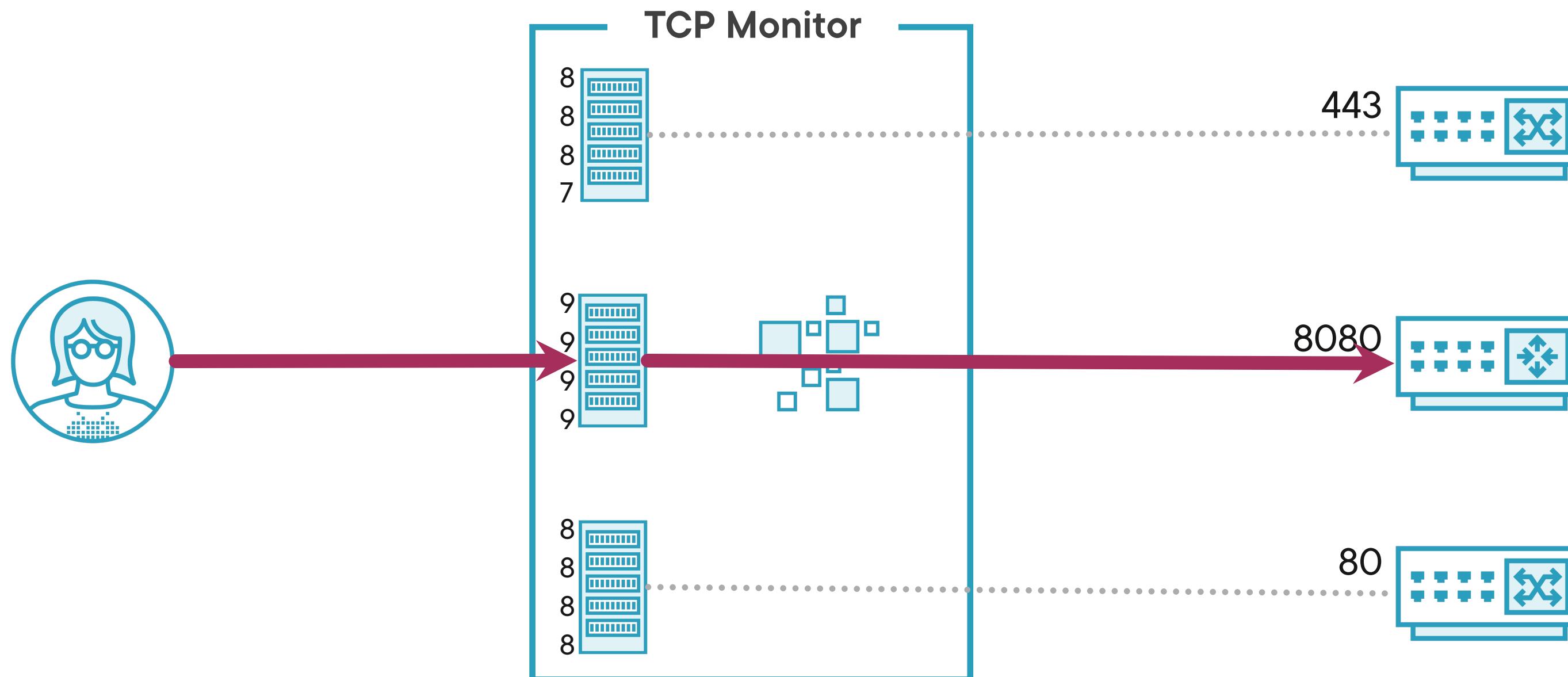
It is enabled with the --debug flag and it will appear in the application log.



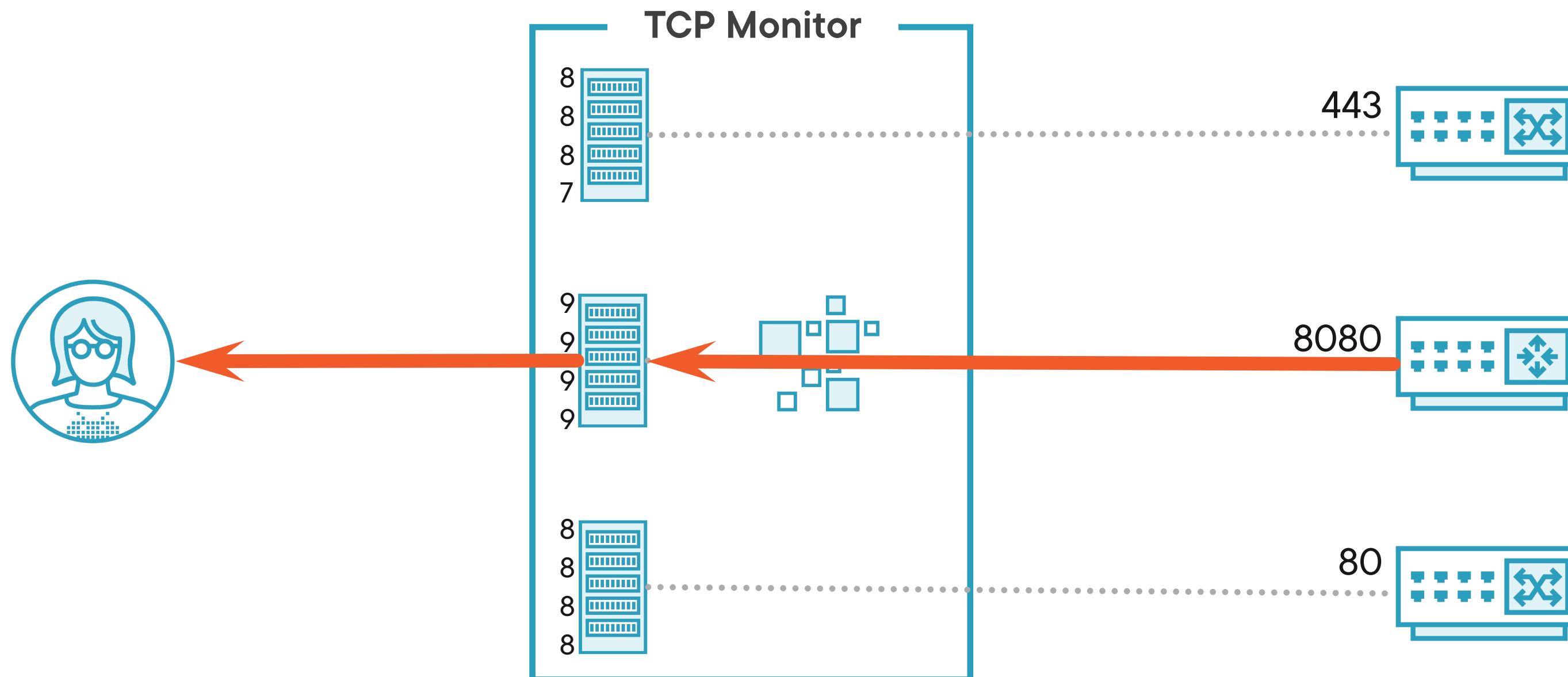
Eclipse TCP Monitor



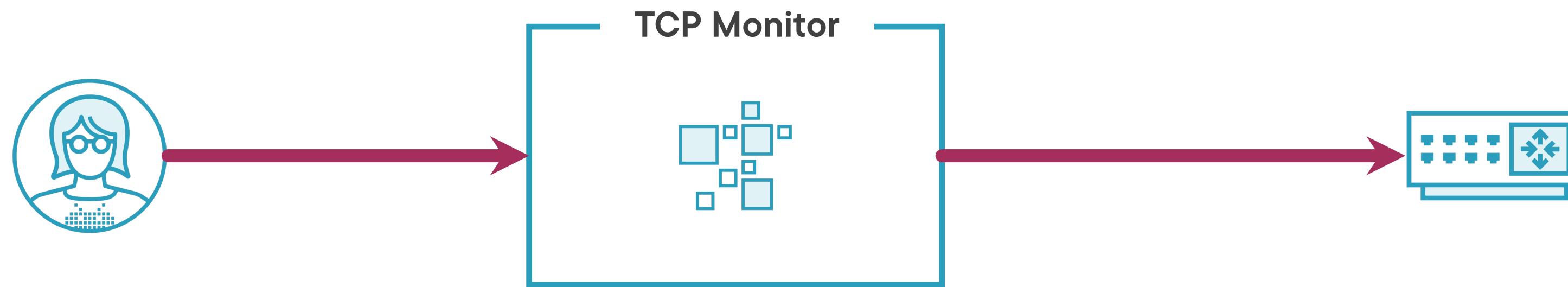
Eclipse TCP Monitor



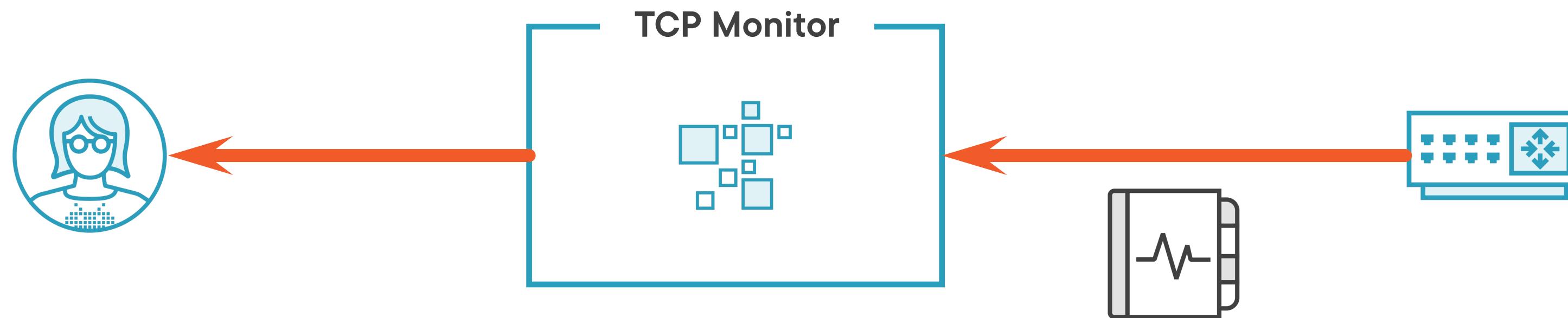
Eclipse TCP Monitor



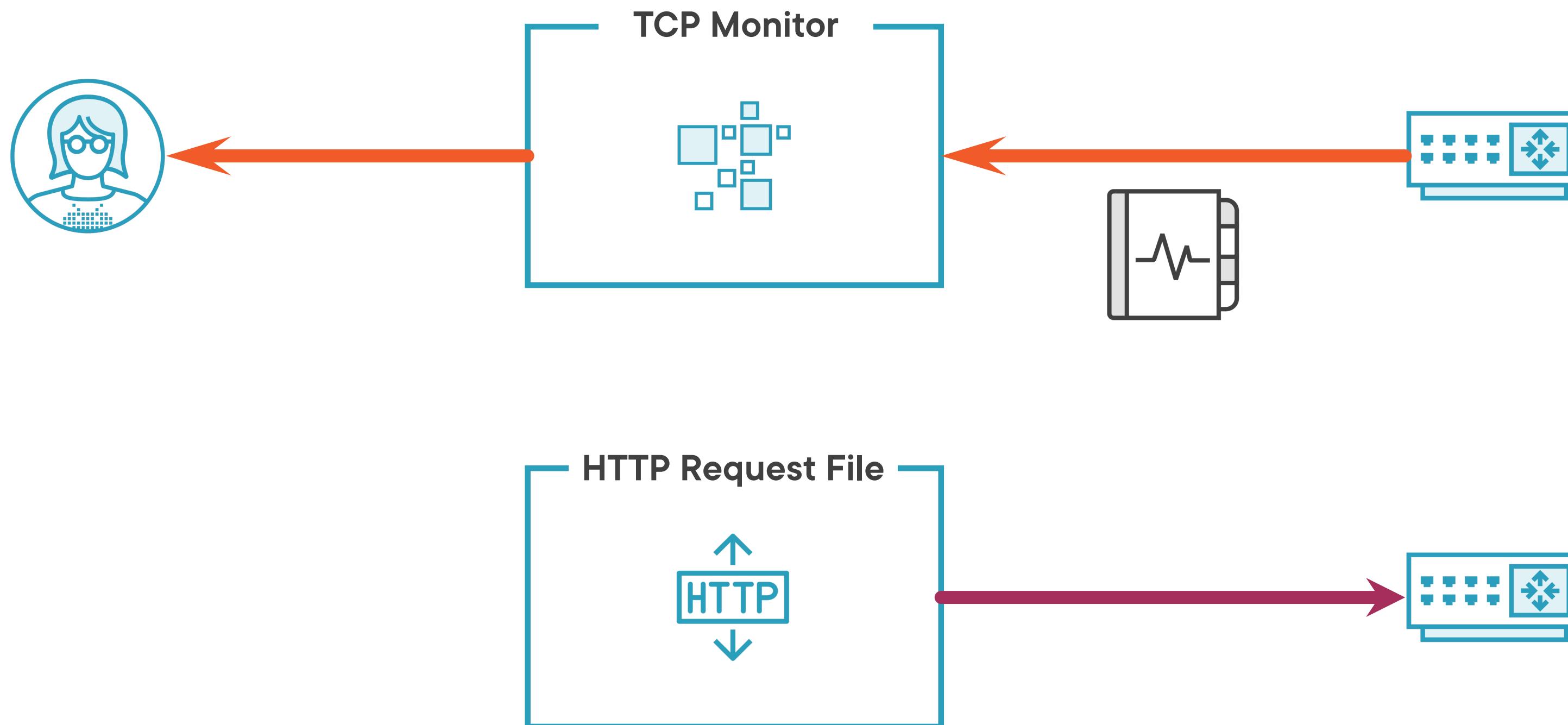
IntelliJ HTTP Request File vs Eclipse TCP Monitor



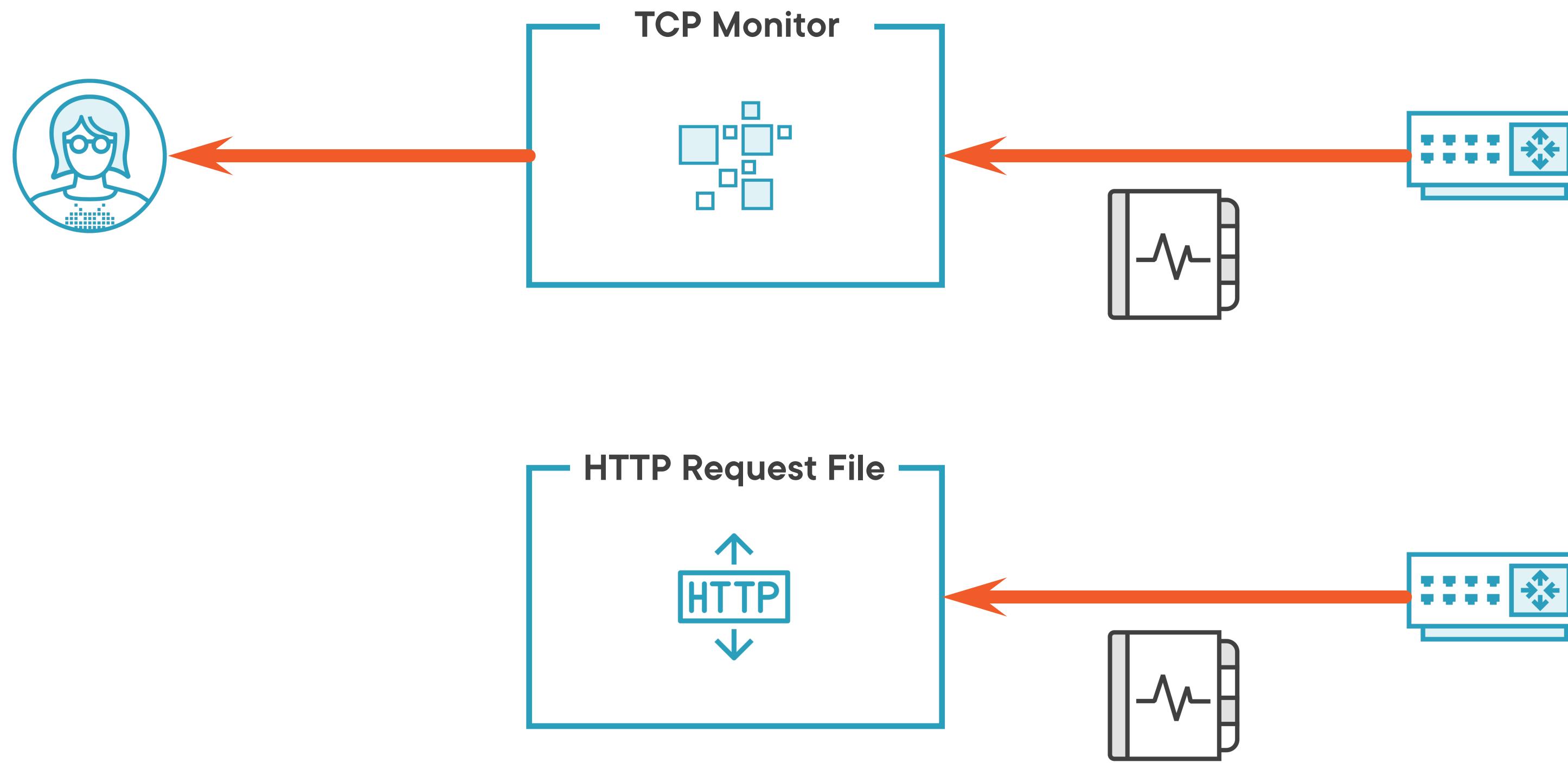
IntelliJ HTTP Request File vs Eclipse TCP Monitor



IntelliJ HTTP Request File vs Eclipse TCP Monitor



IntelliJ HTTP Request File vs Eclipse TCP Monitor



Summary



Effective development

- Spring Initializr
 - From the web, console, or IDE
- Navigating Spring configurations in IDE
- Testing HTTP endpoints



Summary



Effective configuration

- Spring Boot autoconfiguration
 - The `spring.factories` file
- Hierarchical contexts
- Advanced component scanning
 - And custom type filters
- Built-in and custom conditional beans



Summary



Effective deployment

- Spring Dev tools
 - Auto-reload of source code changes
- Logging settings in Spring Boot
 - Conditions Evaluation Report
- Optional dependencies in Maven
 - And features in Gradle



Up Next:
Externalizing Configuration with Properties
and YAML Files

