Implementing a PDF Template Registry with Subcontexts and Custom Scanners

Federico Mestrone

Software Engineer and Training Consultant

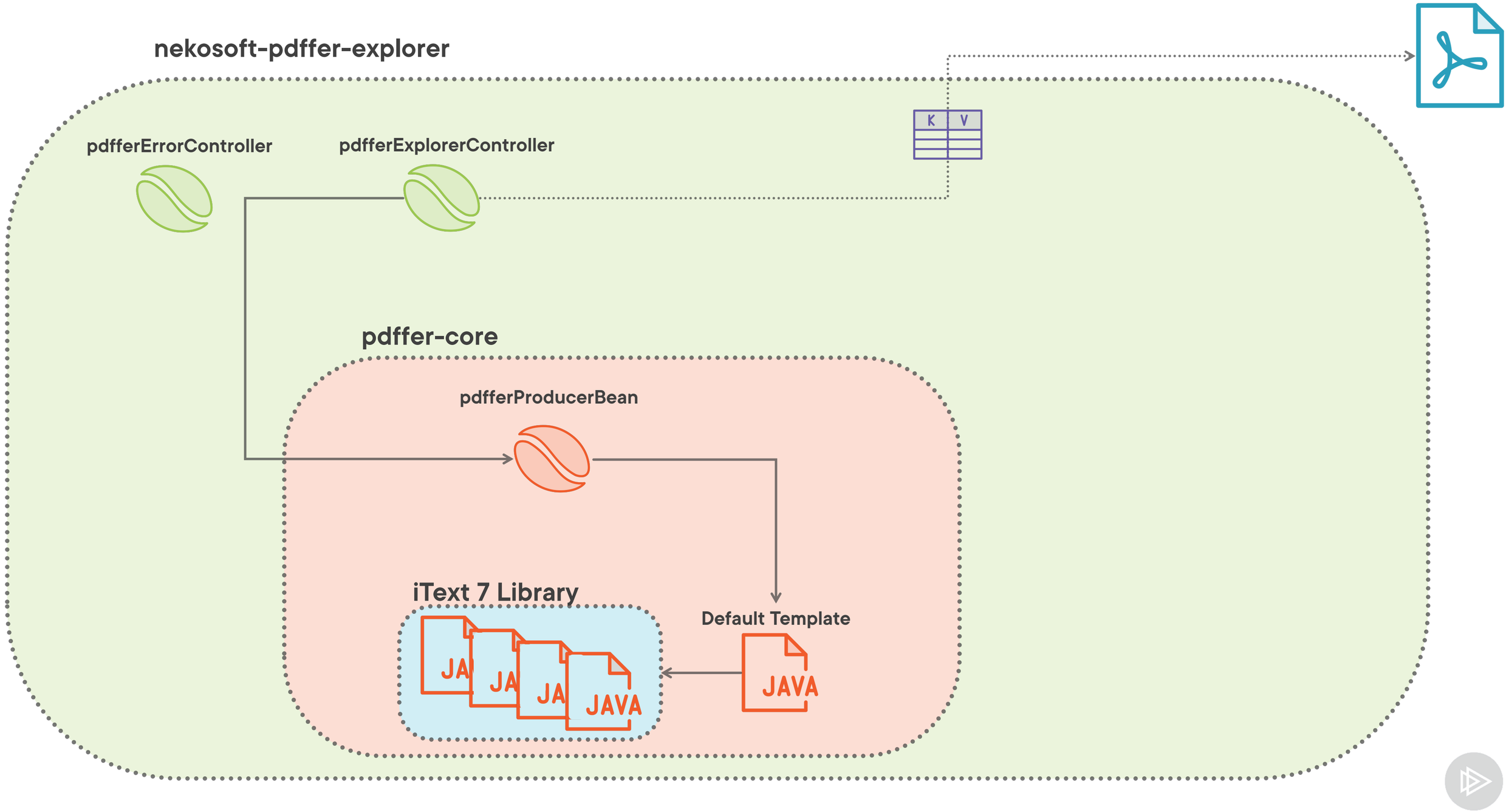
@fedmest www.federicomestrone.com

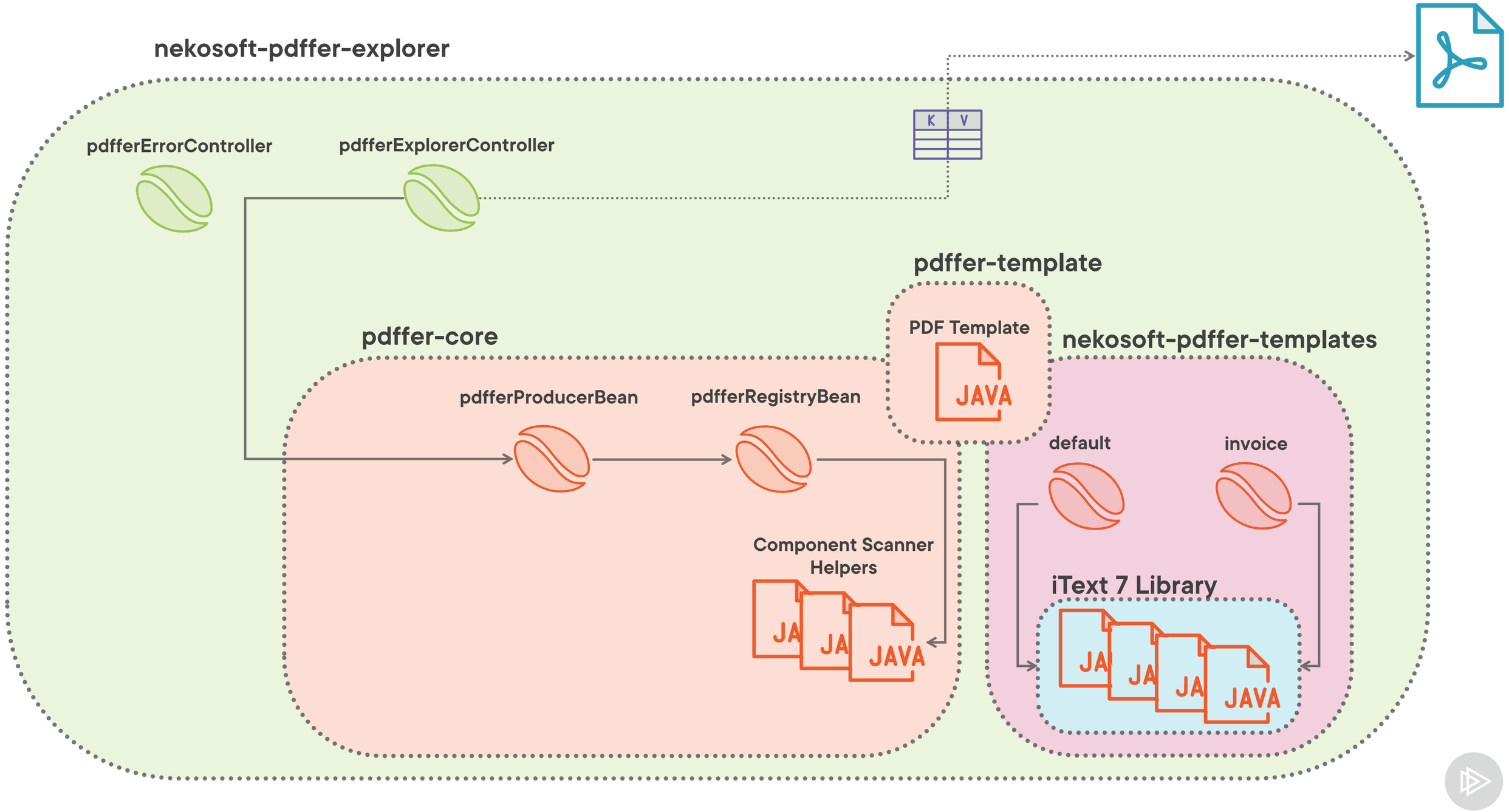
Library Overview

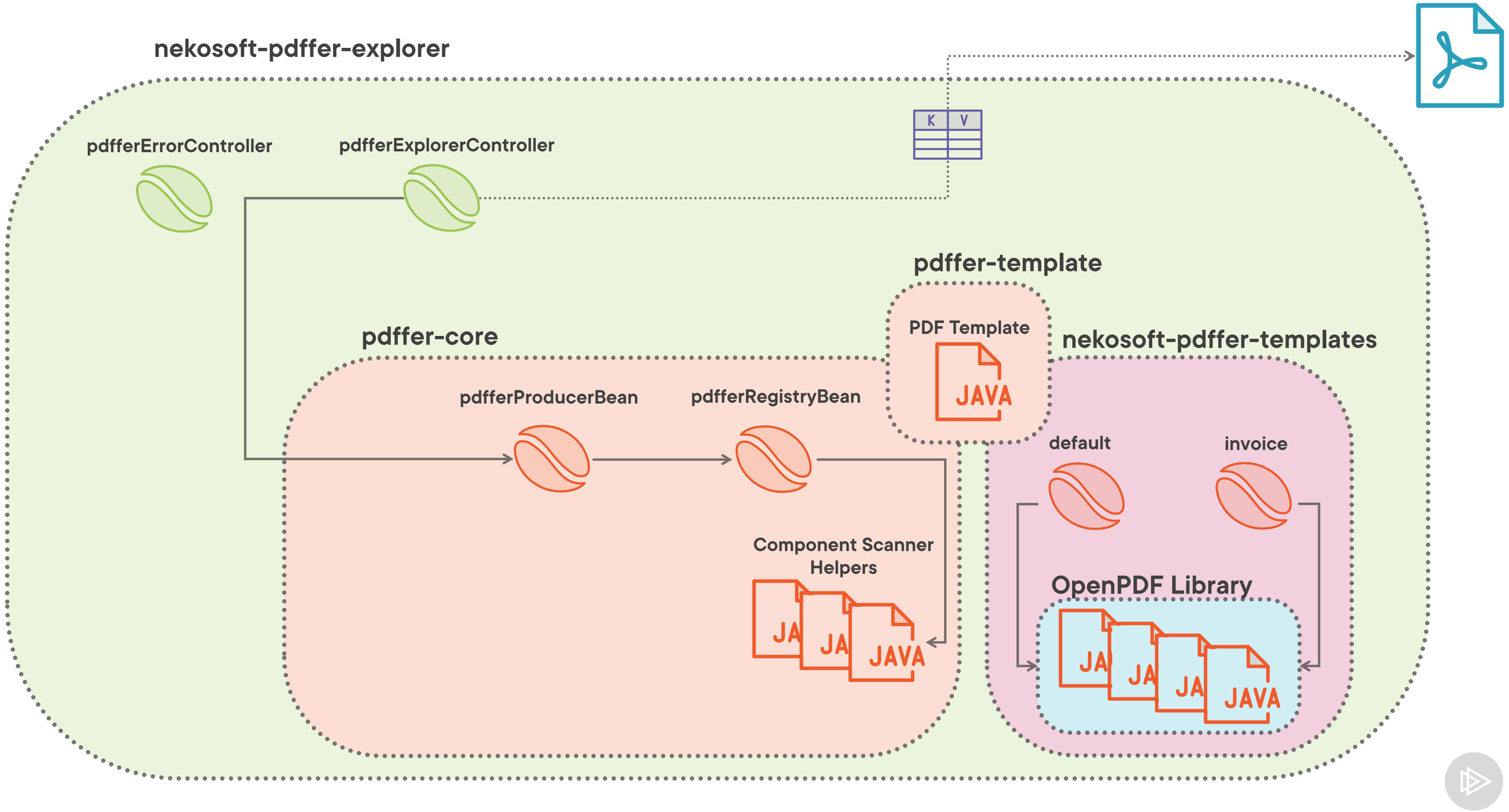
In this module:

- Implement a registry of PDF templates
- Define a couple of sample templates
- Advanced custom component scanners
  - Registry will be extensible
  - No change to original library code
  - No dependency on Spring Boot

The PDFfer Registry


nekosoft-pdffer-explorer
pdfferErrorController
pdfferExplorerController
K
V
pdffer-core
pdfferProducerBean
iText 7 Library
JA
JA
JA
JAVA
Default Template
JAVA


nekosoft-pdffer-explorer
pdfferErrorController
pdfferExplorerController
K
V
pdffer-template
PDF Template
JAVA
pdffer-core
nekosoft-pdffer-templates
pdfferProducerBean
pdfferRegistryBean
default
invoice
Component Scanner
Helpers
JA
JA
JAVA
iText 7 Library
JA
JA
JA
JAVA


nekosoft-pdffer-explorer
pdfferErrorController
pdfferExplorerController
K
V
pdffer-template
PDF Template
JAVA
pdffer-core
nekosoft-pdffer-templates
pdfferProducerBean
pdfferRegistryBean
default
invoice
Component Scanner
Helpers
JA
JA
JAVA
OpenPDF Library
JA
JA
JA
JAVA

The PdfferRegistryBean Class (1/2)

```
@Component
public class PdfferRegistryBean {

  private final ApplicationContext context;

  public PdfferRegistryBean(ApplicationContext parentContext) {
    AnnotationConfigApplicationContext c = new AnnotationConfigApplicationContext();
    c.setId("pdffer-templates");
    c.register(PdfferTemplateRegistryConfiguration.class);
    c.refresh();
    c.start();
    context = c;
  }
```

The PdfferRegistryBean Class (2/2)

```
    public List<String> listTemplates() {
        return List.of(context.getBeanNamesForAnnotation(PdfTemplateComponent.class));
    }

    public PdfTemplate findTemplate(String templateName) {
        return context.getBean(templateName, PdfTemplate.class);
    }

}
```

The PdfferProducerBean Class

```
@Component
public class PdfferProducerBean {

    private final PdfferRegistryBean registry;

    public PdfferProducerBean(PdfferRegistryBean registry) {
        this.registry = registry;
    }

    public byte[] generatePdfDocument(String templateName, Map<String, Object> data) {
        // same as before
    }

    PdfTemplate findTemplate(String templateName) {
        return registry.findTemplate(templateName);
    }

}
```

Component Scanning

Allows Spring beans to be defined by looking for specific annotations or class features in selected packages

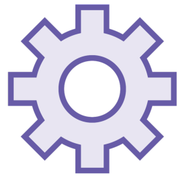
Enabling Component Scanning

```
@Configuration
@ComponentScan
public class PdfferTemplateRegistryConfiguration {

}
```

The Default Stereotypes

@Configuration
(Not a stereotype) defines beans and options for an application context



@Component
A generic Spring bean

@Service
A Spring bean representing an element of the service layer

@Repository
A Data Access Layer object with DataLayerException translation

@Controller
A web controller that responds to HTTP requests

Scanning Specific Packages

```
@Configuration
@ComponentScan(
    basePackages = "org.nekosoft.PDFferTemplates"
)
public class PdfferTemplateRegistryConfiguration {

}
```

Scanner Filters

Exclusion Filters

Applied first.
If any exclusion filter matches, the class is skipped.

Inclusion Filters

Applied if exclusion didn't match.
If any inclusion filter matches, the class is registered in the context.

Scanner Filter Types

FilterType.ANNOTATION
Matches classes that are annotated with the given annotation

FilterType.ASSIGNABLE_TYPE
Matches classes that extend or implement the given class/interface

FilterType.ASPECTJ
Uses AspectJ expressions to match classes

FilterType.REGEX
Uses regular expressions to match class names

FilterType.CUSTOM
Uses the given TypeFilter implementation to select classes

Custom Component Annotation

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface PdfTemplateComponent {

    String name();
    boolean singleton() default false;

}
```

Inclusion Filters

```
@Configuration
@ComponentScan(
    basePackages = "org.nekosoft.PDFferTemplates",
    includeFilters = @ComponentScan.Filter(
        type = FilterType.ANNOTATION, value = PdfTemplateComponent.class
    )
)
public class PdfferTemplateRegistryConfiguration {

}
```

Custom Scanner Filter

```
public class PdfferTemplateExcludeFilter implements TypeFilter {

    @Override
    public boolean match(MetadataReader reader, MetadataReaderFactory factory) {
        try {
            Class<?> c = Class.forName(reader.getClassMetadata().getClassName());
            return !PdfTemplate.class.isAssignableFrom(c);
        } catch (ClassNotFoundException e) {
            return false;
        }
    }

}
```

Exclusion Filters

```
@Configuration
@ComponentScan(
    basePackages = "org.nekosoft.PDFferTemplates",
    excludeFilters = @ComponentScan.Filter(
        type = FilterType.CUSTOM, value = PdfferTemplateExcludeFilter.class
    ),
    includeFilters = @ComponentScan.Filter(
        type = FilterType.ANNOTATION, value = PdfTemplateComponent.class
    )
)
public class PdfferTemplateRegistryConfiguration {

}
```

Disabling Default Filters

```
@Configuration
@ComponentScan(
    basePackages = "org.nekosoft.PDFferTemplates",
    useDefaultFilters = false,
    excludeFilters = @ComponentScan.Filter(
        type = FilterType.CUSTOM, value = PdfferTemplateExcludeFilter.class
    ),
    includeFilters = @ComponentScan.Filter(
        type = FilterType.ANNOTATION, value = PdfTemplateComponent.class
    )
)
public class PdfferTemplateRegistryConfiguration {

}
```

Other Scanning Features

Name generator

Bean Name Generator

```
public class PdfferTemplateBeanNameGenerator implements BeanNameGenerator {

    @Override
    public String generateBeanName(BeanDefinition def, BeanDefinitionRegistry reg) {
        try {
            Class<?> c = Class.forName(def.getBeanClassName());
            PdfTemplateComponent annotation = c.getAnnotation(PdfTemplateComponent.class);
            return annotation.name();
        } catch (ClassNotFoundException e) {
            // this should never happen, given the scanner filters in place...
            throw new BeanCreationException("Cannot find bean class");
        }
    }

}
```

Configuration of Bean Name Generator

```
@Configuration
@ComponentScan(
	basePackages = "org.nekosoft.PDFferTemplates",
	useDefaultFilters = false,
	excludeFilters = @ComponentScan.Filter(
		type = FilterType.CUSTOM, value = PdfferTemplateExcludeFilter.class
	),
	includeFilters = @ComponentScan.Filter(
		type = FilterType.ANNOTATION, value = PdfTemplateComponent.class
	),
	nameGenerator = PdfferTemplateBeanNameGenerator.class
)
public class PdfferTemplateRegistryConfiguration {

}
```

Other Scanning Features

Name generator

Scope resolver

Scope Resolver

```
public class PdfferTemplateScopeMetadataResolver implements ScopeMetadataResolver {
    @Override
    public ScopeMetadata resolveScopeMetadata(BeanDefinition definition) {
        try {
            Class<?> c = Class.forName(definition.getBeanClassName());
            PdfTemplateComponent annotation = c.getAnnotation(PdfTemplateComponent.class);
            ScopeMetadata scope = new ScopeMetadata();
            if (annotation.singleton()) {
                scope.setScopeName(BeanDefinition.SCOPE_SINGLETON);
            } else {
                scope.setScopeName(BeanDefinition.SCOPE_PROTOTYPE);
            }
            return scope;
        } catch (ClassNotFoundException e) { // this should never happen
            throw new BeanCreationException("Cannot find bean class");
        }
    }
}
```

Configuration of Scope Resolver

```
@Configuration
@ComponentScan(
	basePackages = "org.nekosoft.PDFferTemplates",
	useDefaultFilters = false,
	excludeFilters = @ComponentScan.Filter(
		type = FilterType.CUSTOM, value = PdfferTemplateExcludeFilter.class
	),
	includeFilters = @ComponentScan.Filter(
		type = FilterType.ANNOTATION, value = PdfTemplateComponent.class
	),
	nameGenerator = PdfferTemplateBeanNameGenerator.class,
	scopeResolver = PdfferTemplateScopeMetadataResolver.class
)
public class PdfferTemplateRegistryConfiguration {

}
```

Hierarchical Contexts

Hierarchical Application Contexts

ParentContextConfiguration

Parent Context

bean1

bean2

ChildContextConfiguration

Child Context

bean3

bean4

Defining a Child Context

```
childContext.setParent(parentContext);
```

Hierarchical Application Contexts

ParentContextConfiguration

Parent Context

bean1

bean2

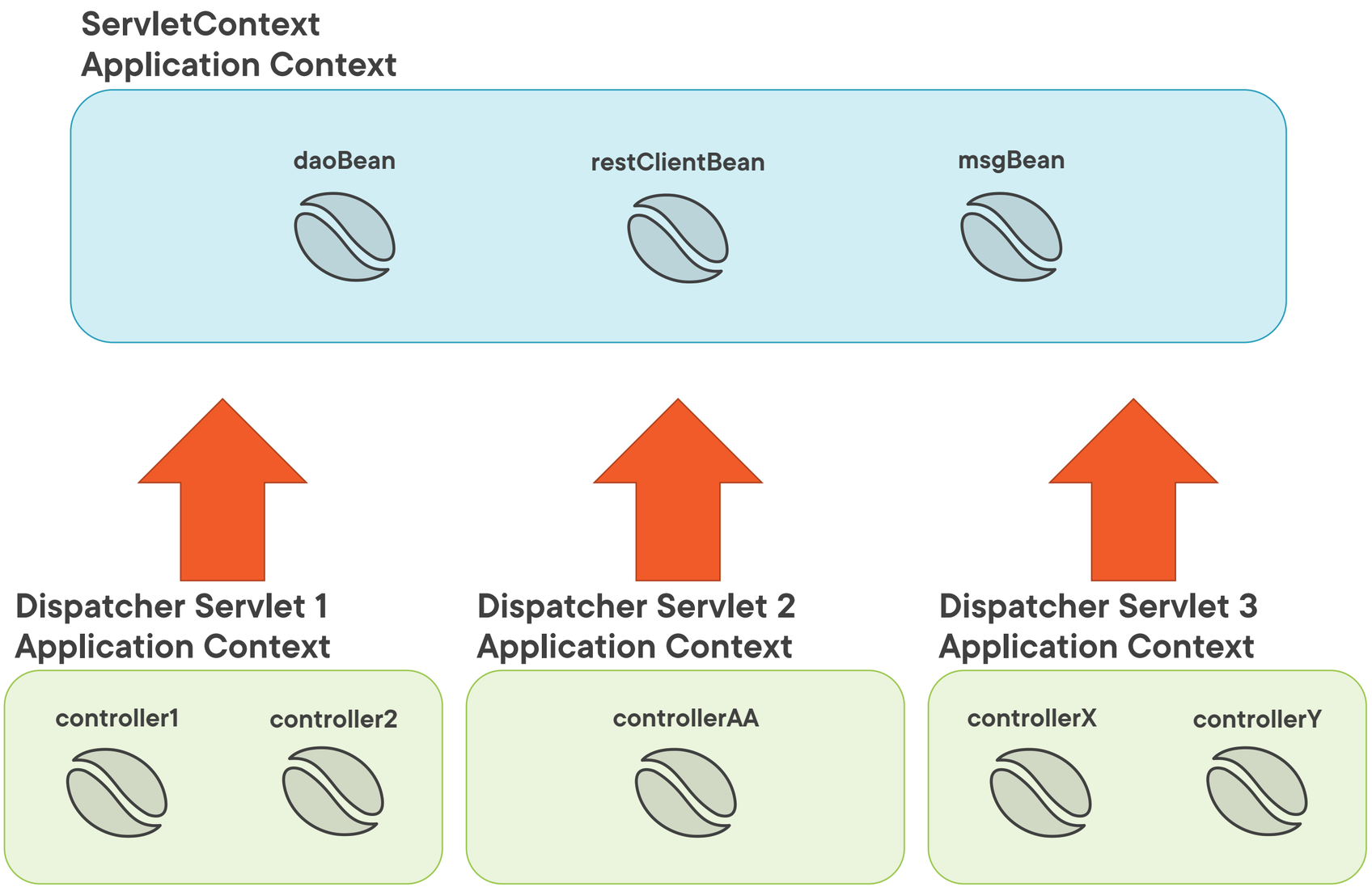
Child Context

bean3

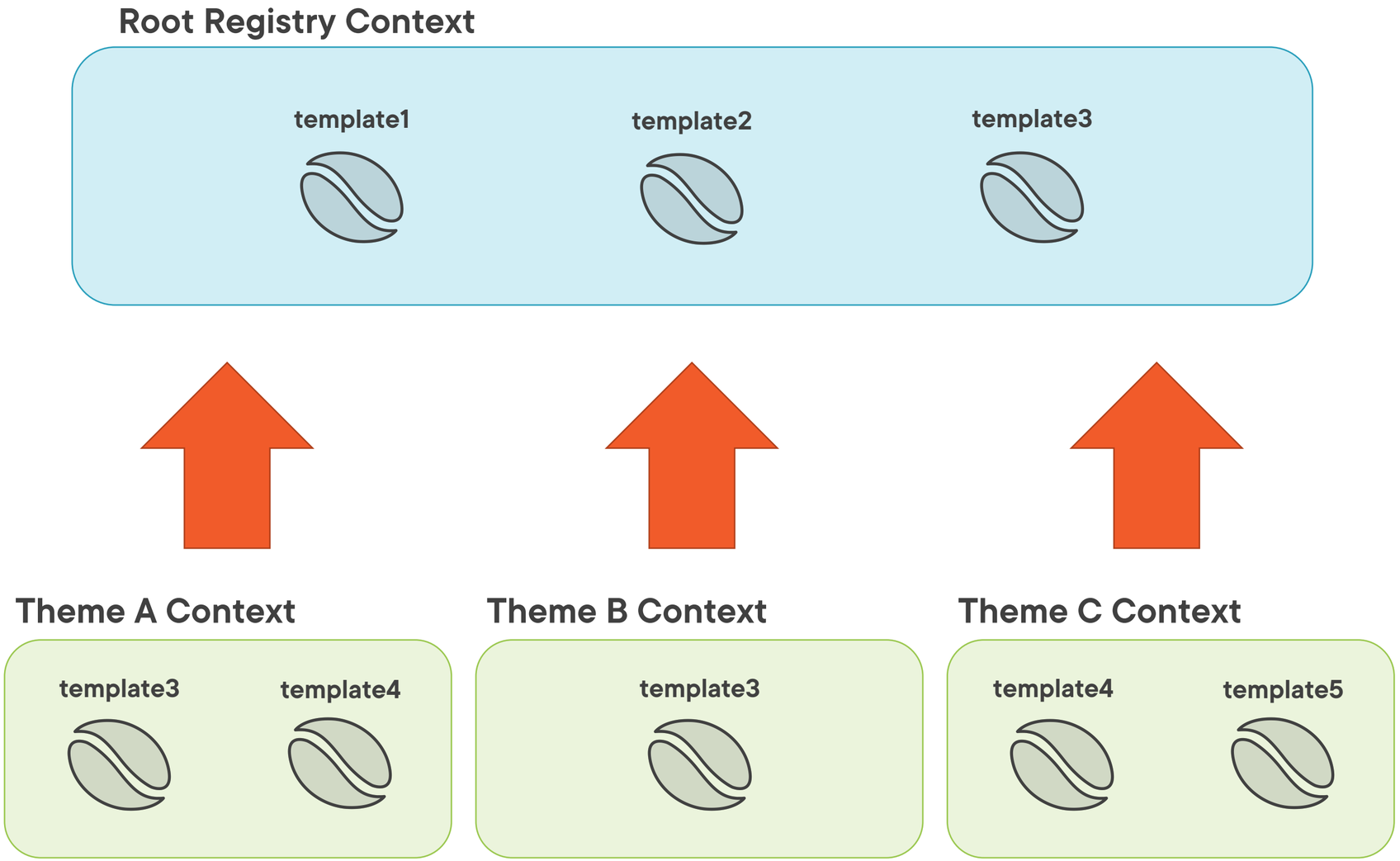
bean4

ChildContextConfiguration

Web Hierarchical Contexts



PDFfer Template Groups



Spring Boot Logging

Spring Boot Logging

Works out of the box

Default configuration

- Apache Commons Logging framework
- With Logback loggers
- And level set to INFO

Customize logging levels

Full customization with underlying logger configuration files

Summary

Effective development

- Spring Initialzr
  - From the web, console, or IDE
- Navigating Spring configurations in IDE

Summary

Effective configuration

- Spring Boot autoconfiguration
  - The spring.factories file
- Hierarchical contexts
- Advanced component scanning
  - And custom type filters

Summary

Effective deployment

- Spring Dev tools
  - Auto-reload of source code changes
- Logging settings in Spring Boot

Up Next:

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