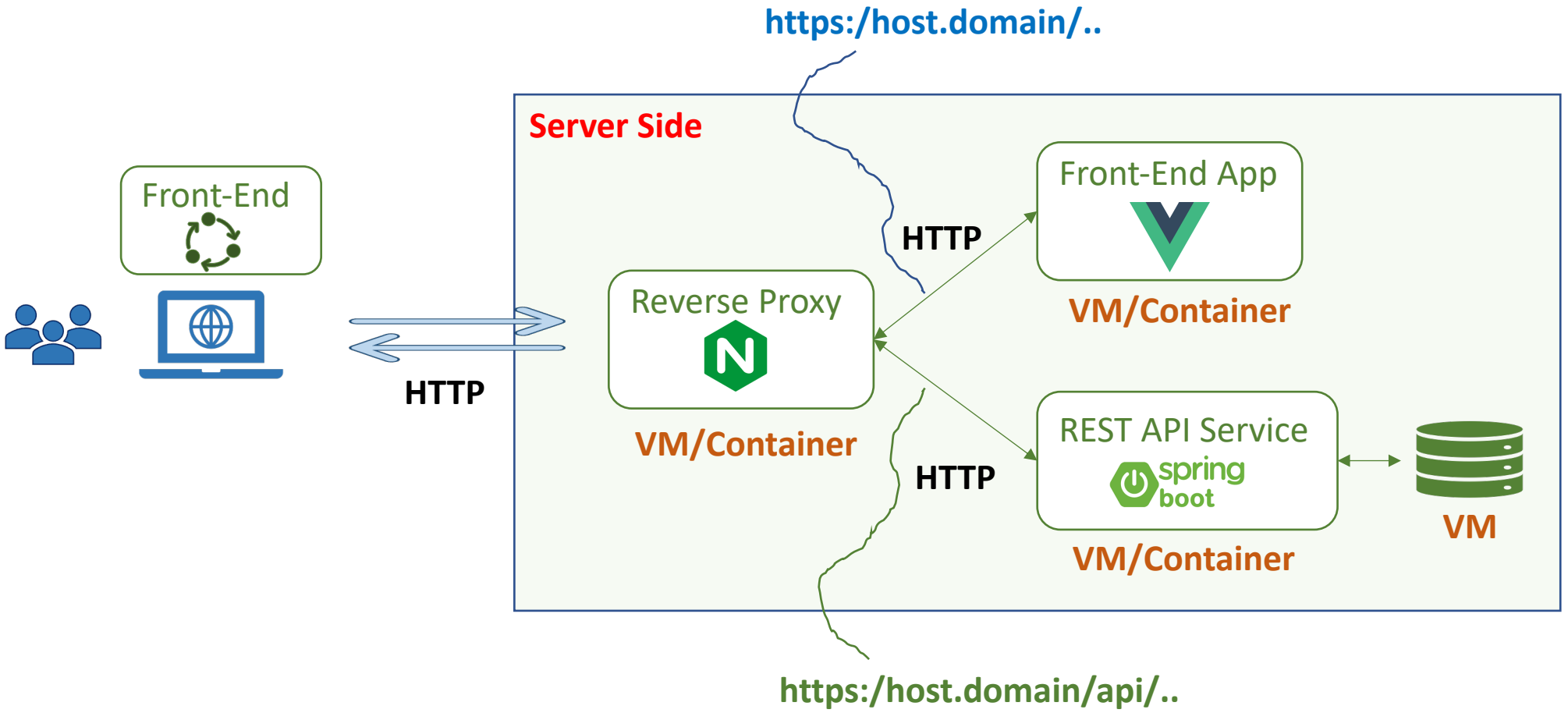
Spring RESTful API File Services

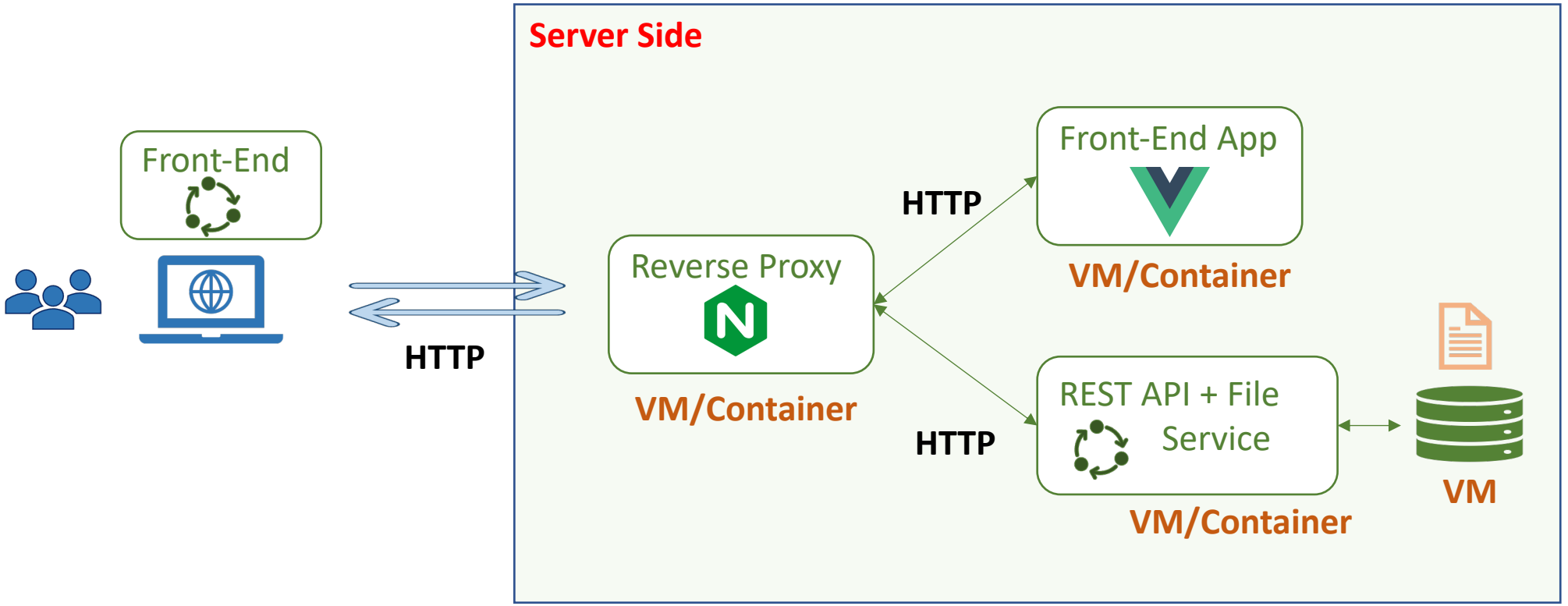
By

Pichet Limvajiranan

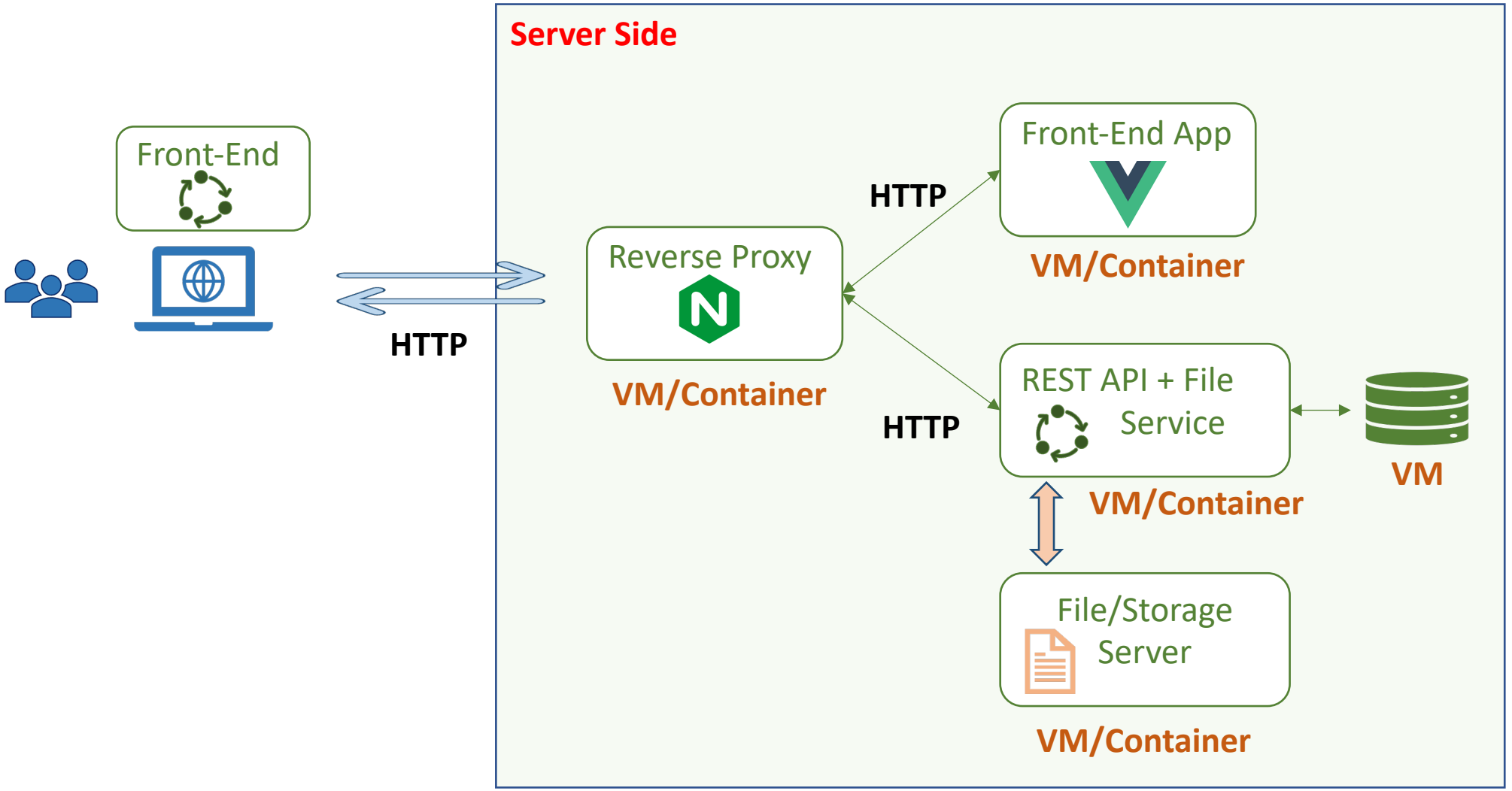
SPA Running Environment



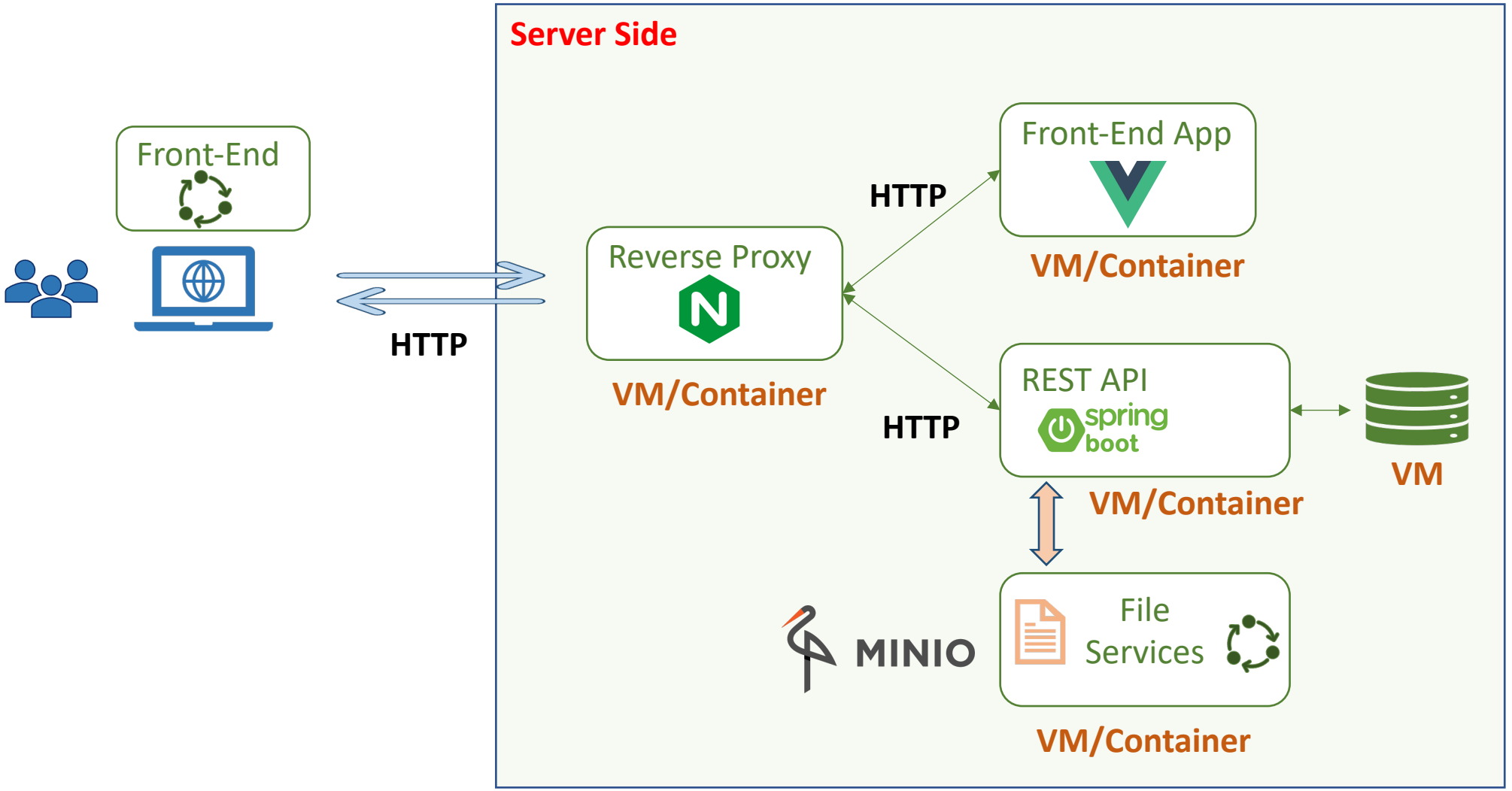
File Storage Architectures (1)



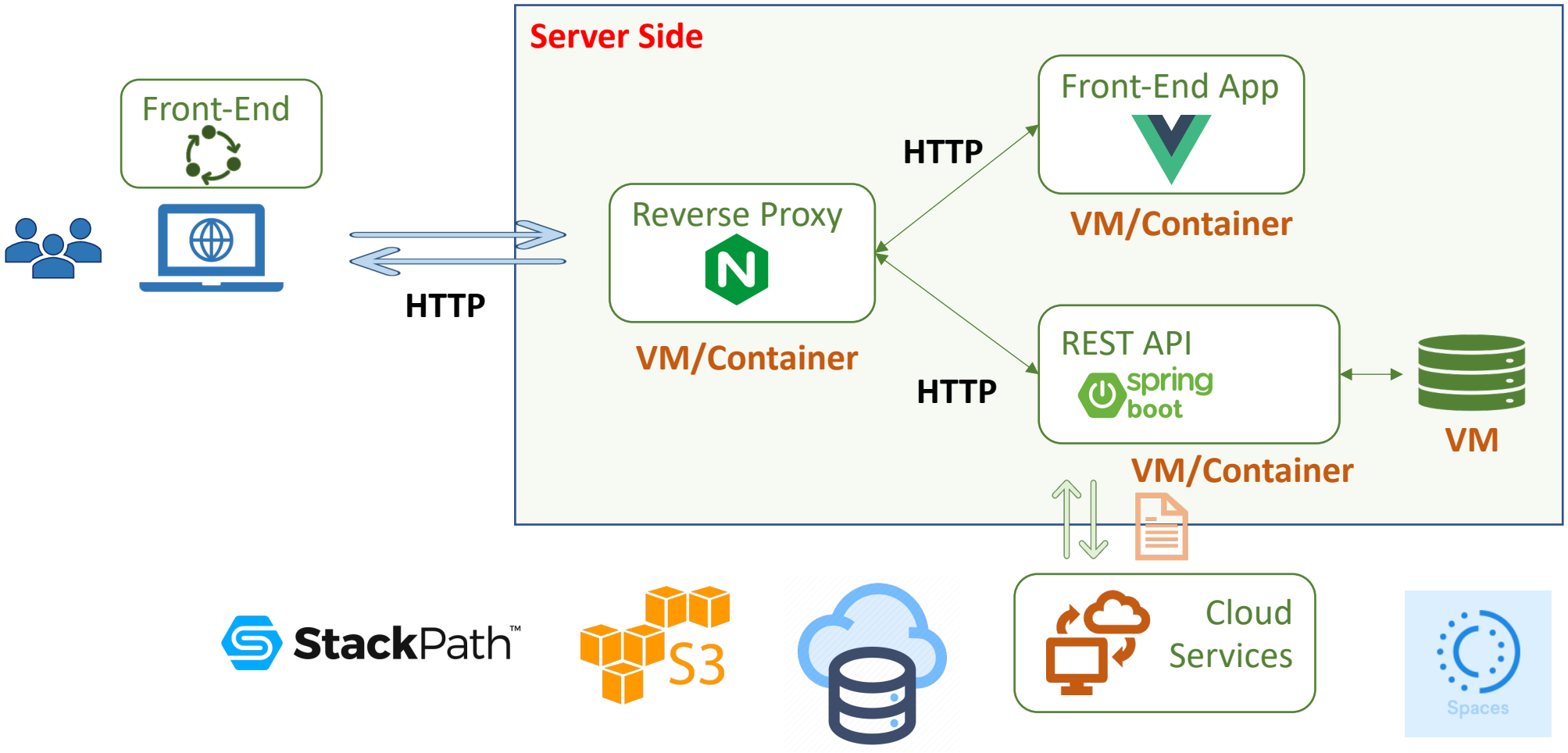
File Storage Architectures (2)



File Storage Architectures (3)



File Storage Architectures (4)



File Operation

- Upload
  - Send file – Form Data (Multipart) / File Only (FE)
  - **Read file data from Http Request (BE)**
  - **Write file to storage (BE)**
- Download
  - Return file URL / File content
- Delete
  - Read file name from Http Request
  - Remove file from storage

Configuring Server and File Storage Properties

```
## MULTIPART (MultipartProperties)
# Enable multipart uploads
spring.servlet.multipart.enabled=true

# Max file size.
spring.servlet.multipart.max-file-size=10MB

# Max Request Size
spring.servlet.multipart.max-request-size=80MB

## File Storage Properties # All files uploaded through the REST API will be stored in this directory
file.upload-dir=/public/classicmodels/uploads

# Threshold after which files are written to disk. spring.servlet.multipart.file-size-threshold=4KB

## File Storage Properties # All files uploaded through the REST API will be stored in this directory
file.upload-dir=./product-images
```

Automatically binding properties to a POJO class

- Spring Boot has an awesome feature called @ConfigurationProperties using which you can automatically bind the properties defined in the application.properties file to a POJO class.
- Let's define a POJO class called FileStorageProperties inside based-package.properties package to bind all the file storage properties.

```
file.upload-dir=./product-images
```

```
@ConfigurationProperties(prefix = "file")
@Getter
@Setter
public class FileStorageProperties {
  private String uploadDir;
}
```

```
@SpringBootApplication
@EnableConfigurationProperties({
      FileStorageProperties.class
})
public class DemoApplication {
   public static void main(String[] args) {
```

Binding properties to a POJO class

Create Configuration property class: FileStorageProperties.java (package based-package.properties

```
@ConfigurationProperties(prefix = "file")
@Getter
@Setter
public class FileStorageProperties {
  private String uploadDir;
}
```

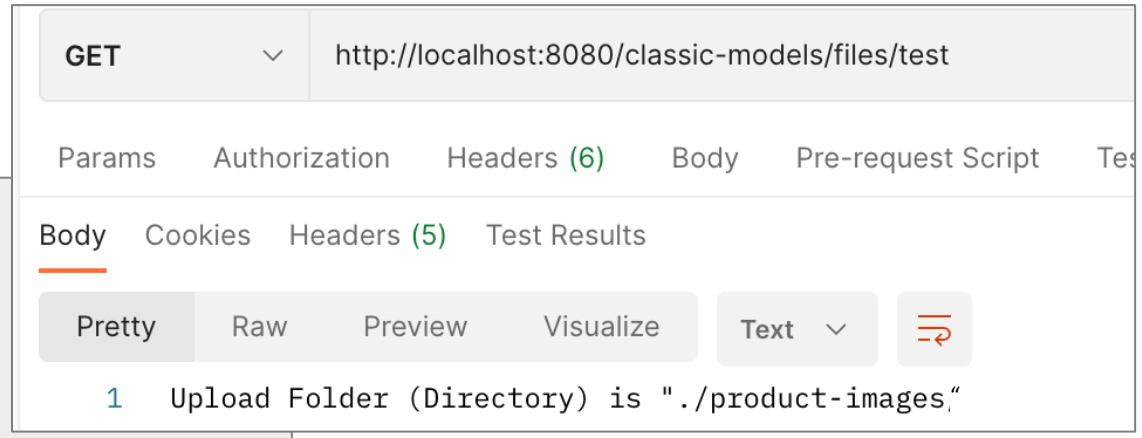
Register Configuration Property (into ApplicationConfig.java or ClassicmodelServiceApplication.java)

```
@EnableConfigurationProperties({
        FileStorageProperties.class
})
```

Test properties/POJO binding

```java
@RestController
@RequestMapping("/files")
public class FileController {
    @Autowired
    FileStorageProperties fileStorageProperties;

    @GetMapping("/test")
    public ResponseEntity<Object> testPropertiesMapping() {
        return ResponseEntity.ok("Upload Folder (Directory) is \""
                + fileStorageProperties.getUploadDir()+ "\"");
    }
}
```



FileService : Initialize

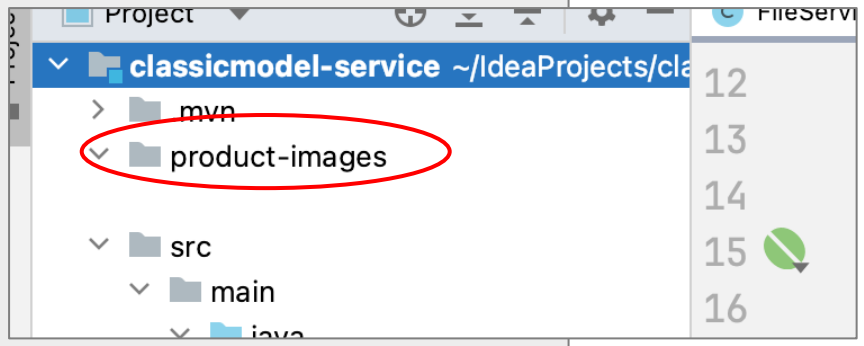
```
import java.io.IOException;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
```

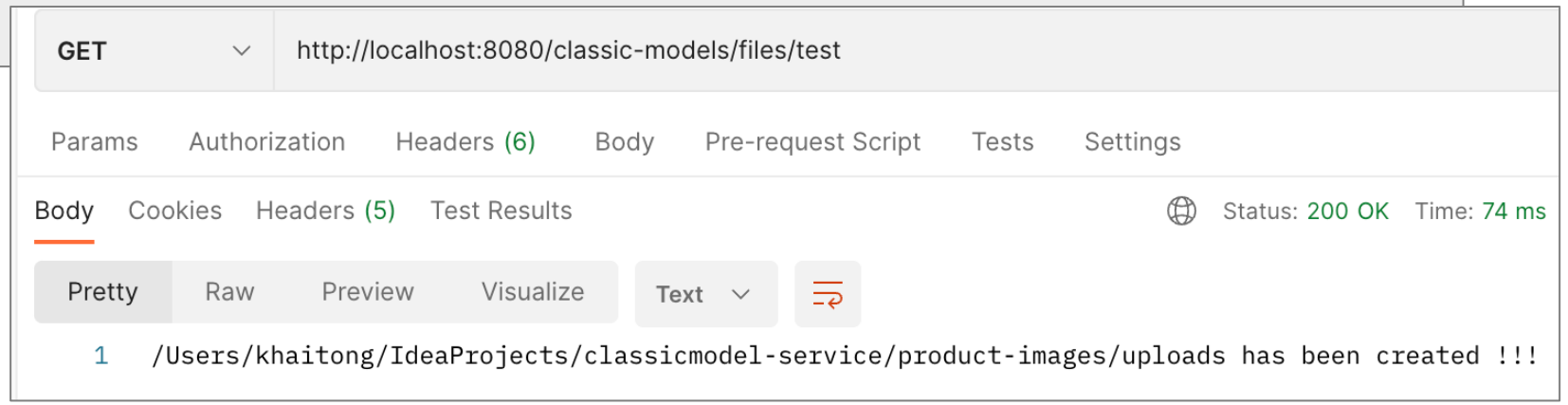
```
@Service
@Getter
public class FileService {
   private final Path fileStorageLocation;
   @Autowired
   public FileService(FileStorageProperties fileStorageProperties) {
      this.fileStorageLocation = Paths.get(fileStorageProperties
            .getUploadDir()).toAbsolutePath().normalize();
      try {
         if (! Files.exists(this.fileStorageLocation)) {
               Files.createDirectories(this.fileStorageLocation);
         }
      } catch (IOException ex) {
         throw new RuntimeException(
          "Could not create the directory where the uploaded files will be stored.", ex);
      }
   }
```

Modify FileController to Test FileService - Initialize

```java
@RestController
@RequestMapping("/files")
public class FileController {
    @Autowired
    FileService fileService;

    @GetMapping("/test")
    public ResponseEntity<Object> testPropertiesMapping() {
        return ResponseEntity.ok(fileService.getFileStorageLocation()+ " has been created !!!");
    }
}
```



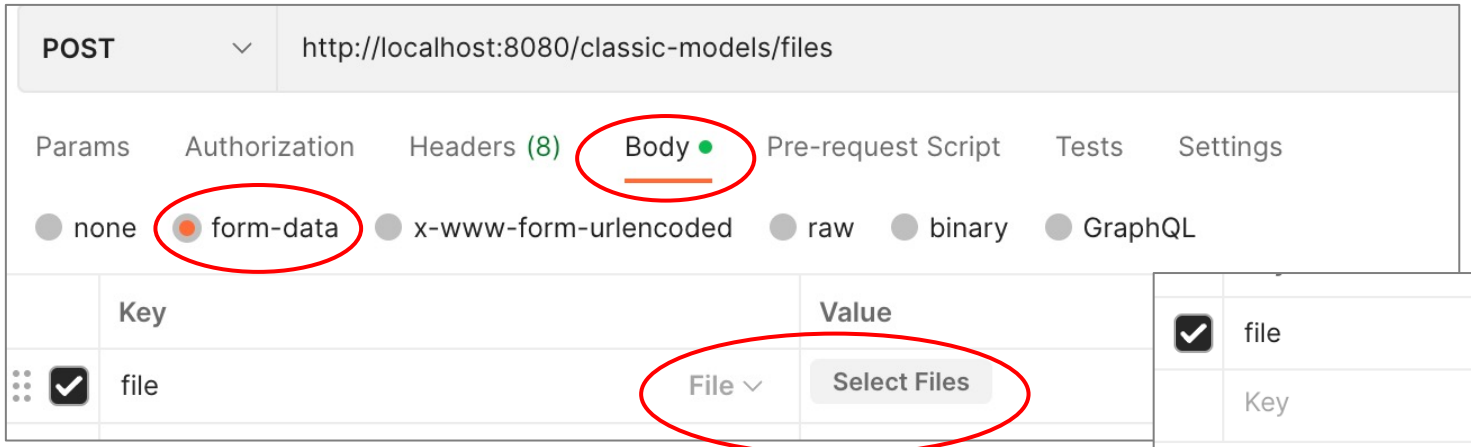


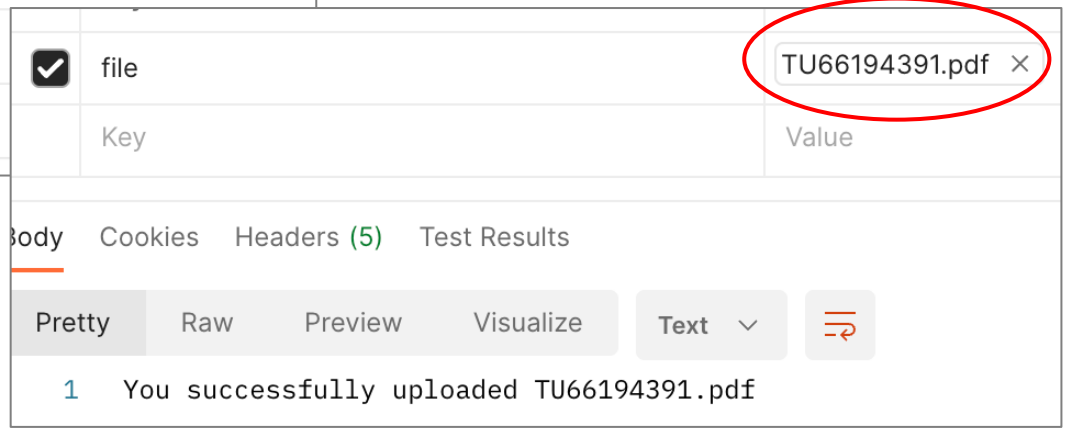
FileService: Storing File (add this method to FileService)

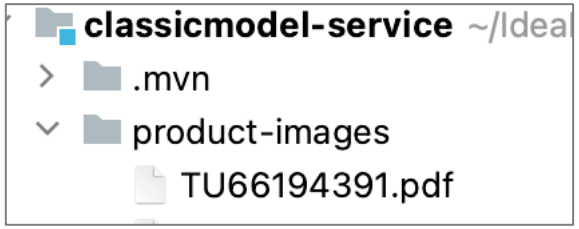
```
public String store(MultipartFile file) {
    // Normalize file name
    String fileName = StringUtils.cleanPath(file.getOriginalFilename());
    try {
        // Check if the file's name contains invalid characters
        if (fileName.contains("..")) {
            throw new RuntimeException("Sorry! Filename contains invalid path sequence " + fileName);
        }
        // Copy file to the target location (Replacing existing file with the same name)
        Path targetLocation = this.fileStorageLocation.resolve(fileName);
        Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_EXISTING);
        return fileName;
    } catch (IOException ex) {
        throw new RuntimeException("Could not store file " + fileName + ". Please try again!", ex);
    }
}
```

Add this method to FileController - Test FileService – Storing File

```
@PostMapping("")
public ResponseEntity<Object> fileUpload(@RequestParam("file") MultipartFile file) {
   fileService.store(file);
   return ResponseEntity.ok("You successfully uploaded " + file.getOriginalFilename());
}
```







FileService: LoadFile

```
public Resource loadFileAsResource(String fileName) {
    try {
        Path filePath = this.fileStorageLocation.resolve(fileName).normalize();
        Resource resource = new UrlResource(filePath.toUri());
        if (resource.exists()) {
            return resource;
        } else {
            throw new RuntimeException("File not found " + fileName);
        }
    } catch (MalformedURLException ex) {
        throw new RuntimeException("File operation error: " + fileName, ex);
    }
}
```

File Controller

```
@GetMapping("/{filename:.+}")
@ResponseBody
public ResponseEntity<Resource> serveFile(@PathVariable String filename) {
    Resource file = fileService.loadFileAsResource(filename);
    return  ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(file);
}
```