jDeploy 5.2 Release Notes

Table of Contents

Overview . . . 1
What's New in 5.2.3 . . . 1
Custom Launcher Splash Screen . . . 1
Enhanced jDeploy Installer Splash Screens . . . 3
File Descriptor Limit Configuration . . . 4
What's New in 5.2 . . . 5
Run as Administrator Feature . . . 5
Linux Installer CLI Command Support . . . 6
Linux Installation Without Desktop Environment . . . 7
Runtime Configuration File Support . . . 7
Technical Details . . . 11
Launcher Changes . . . 11
Linux Installation . . . 11
Error Handling . . . 11
Migration Guide . . . 11
Upgrading from 5.1 . . . 11
Configuring Administrator Privileges . . . 11
Configuring Runtime Configuration . . . 12
Linux CLI Installation . . . 12
Best Practices . . . 12
Known Issues . . . 13
Resources . . . 13

Overview

jDeploy 5.2 adds support for running applications with elevated privileges, includes CLI command installation in the Linux installer, fixes installation on Linux systems without a desktop environment, and introduces runtime configuration files for external argument specification.

What's New in 5.2.3

Custom Launcher Splash Screen

Applications can now provide a custom HTML splash screen that displays during installation and updates. By creating a `launcher-splash.html` file in your project root (same directory as package.json), you can customize the splash screen shown when the application and JRE are being

installed or updated.

How to Use

1. Create a file named `launcher-splash.html` in the same directory as your package.json
2. Design your splash screen using HTML and inline CSS
3. The splash screen will automatically be used during app installation and updates

Features

- Fully customizable HTML/CSS design
- Automatically displayed during installation and JRE updates
- No configuration required - just add the file and it's automatically detected
- Provides better branding and user experience during installation

Example

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            font-family: Arial, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
        }
        .content {
            text-align: center;
        }
        h1 {
            font-size: 3em;
            margin-bottom: 20px;
        }
        .spinner {
            border: 4px solid rgba(255,255,255,0.3);
            border-top: 4px solid white;
            border-radius: 50%;
            width: 50px;
            height: 50px;
            animation: spin 1s linear infinite;
            margin: 20px auto;
        }
```

```
        @keyframes spin {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }
    </style>
</head>
<body>
    <div class="content">
        <h1>My Awesome App</h1>
        <div class="spinner"></div>
        <p>Installing and updating...</p>
    </div>
</body>
</html>
```

Enhanced jDeploy Installer Splash Screens

The jDeploy installer has been enhanced to provide better branding and user experience during installation.

Improvements

Custom Splash in Installer

The installer now uses your `launcher-splash.html` file if provided, giving users a branded experience from the very first installation step.

Custom Icon in Installer

The installer now uses your custom `icon.png` file during initial installation. Previously, custom icons were only displayed after installation and during updates - now they appear from the very first installation step.

Automatic Integration

Both the custom splash and icon are automatically integrated - no special configuration in package.json is required.

What Changed

Before

- Generic jDeploy splash screen with application name during installation
- Custom icon only used after installation and during updates
- Generic jDeploy icon shown during initial installation

After

- Custom `launcher-splash.html` displayed if present (during installation and updates)
- Custom `icon.png` displayed throughout entire installation process
- Branded experience from first installation through all updates

File Descriptor Limit Configuration

Applications that need to open many files or network connections can now configure file descriptor limits on Unix-like systems (macOS and Linux) using a JVM argument.

How to Use

Add the `-Djdeploy.file.limit` system property to your package.json args:

```
{
  "jdeploy": {
    "args": [
      "-Djdeploy.file.limit=8192"
    ]
  }
}
```

Platform-Specific Configuration

You can configure different limits per platform using platform-conditional arguments:

```
{
  "jdeploy": {
    "args": [
      "-D[mac]jdeploy.file.limit=16384",
      "-D[linux]jdeploy.file.limit=8192"
    ]
  }
}
```

Or target Unix-like platforms together:

```
{
  "jdeploy": {
    "args": [
      "-D[mac|linux]jdeploy.file.limit=8192"
    ]
  }
}
```

Recommended Values

Limit Value	Use Case
4096	Light file I/O applications
8192	Moderate database/network applications

Limit Value	Use Case
16384	Heavy concurrent connection applications

Technical Details

- The limit is set before JVM startup by the launcher
- Cannot exceed your system's hard limit (check with `ulimit -Hn` on Linux/macOS)
- Windows: No-op (Windows uses a different mechanism for handle limits)
- Non-fatal: If setting the limit fails, a warning is logged and the application continues to launch
- The `-Djdeploy.file.limit` property is processed by the launcher and not passed to the JVM

Use Cases

This feature is useful for applications that:

- Handle many concurrent network connections
- Process large numbers of files simultaneously
- Use databases with connection pooling
- Experience "Too many open files" errors

What's New in 5.2

Run as Administrator Feature

Applications can now be configured to run with elevated privileges on Windows, macOS, and Linux.

Configuration Modes

Three configuration levels are available:

disabled (default)
: Applications run with normal user privileges. No elevated launchers are generated.

allowed
: Generates both standard and administrator launcher variants. Users can choose which launcher to use.

required
: All launchers are configured for elevation. No standard launchers are created.

Platform-Specific Implementation

Windows

Uses the native "Run as Administrator" option with User Account Control (UAC) prompt.

macOS

Creates a wrapper launcher app with elevated permissions. Users authenticate with their password.

Linux

Uses `pkexec` for PolicyKit-based privilege escalation.

Package.json Configuration

```
{
  "jdeploy": {
    "runAsAdministrator": "allowed"
  }
}
```

Valid values: `"disabled"`, `"allowed"`, `"required"`

Security Considerations

- Request elevation only when necessary
- Use "allowed" rather than "required" when the application can function without elevation
- Document why elevation is needed
- Validate input in elevated applications

Linux Installer CLI Command Support

The Linux installer now offers an option to install a CLI command during installation.

Functionality

During installation, users can install a symlink in `~/.local/bin`. The installer adds `~/.local/bin` to PATH if needed.

Usage

After installation with the CLI option enabled:

```
$ myapp
```

This works from any terminal session after PATH is configured.

Linux Installation Without Desktop Environment

Fixed a bug preventing installation on Linux systems with a graphical environment but no desktop environment.

What Was Fixed

- Installation now succeeds on systems without a desktop environment (e.g., programs menu, desktop icons)
- Applications can be installed in Windows Subsystem for Linux with graphics support but no desktop environment
- Systems with X11 or Wayland but no full desktop environment are now supported

Use Cases

- WSL installations with graphical support
- Minimal Linux installations with window manager only
- Custom Linux environments without traditional desktop shells

Runtime Configuration File Support

Applications can now load additional runtime arguments from external JSON configuration files.

Purpose

Runtime configuration files allow:

- Different configurations per deployment environment
- Changing runtime arguments without repackaging
- User customization of JVM memory settings and system properties
- Platform-specific settings determined at runtime

Package.json Configuration

Generic configuration file:

```
{
  "jdeploy": {
    "jar": "target/my-app.jar",
    "javaVersion": "11",
    "configFile": "{{ user.home }}/.my-app/config.json"
  }
}
```

Platform-specific configuration files:

```
{
  "jdeploy": {
    "jar": "target/my-app.jar",
    "configFileMac": "{{ app.dir }}/config/mac-config.json",
    "configFileWin": "{{ exe.dir }}\\config\\win-config.json",
    "configFileLinux": "{{ exe.dir }}/config/linux-config.json"
  }
}
```

Supported Placeholders

Placeholder	Description	Example (macOS)	Example (Windows)
`{{ user.home }}`	User's home directory	`/Users/john`	`C:\Users\john`
`{{ executable.path }}`	Full path to launcher executable	`/Applications/MyApp.app/Contents/MacOS/MyApp`	`C:\Program Files\MyApp\MyApp.exe`
`{{ executable.dir }}`	Directory containing the executable	`/Applications/MyApp.app/Contents/MacOS`	`C:\Program Files\MyApp`
`{{ app.path }}`	Path to application bundle (macOS) or executable (other platforms)	`/Applications/MyApp.app`	`C:\Program Files\MyApp\MyApp.exe`
`{{ app.dir }}`	Directory containing the app bundle (macOS) or executable (other platforms)	`/Applications`	`C:\Program Files\MyApp`

Runtime Configuration File Format

```
{
  "args": [
    "-Xmx4g",
    "-Xms1g",
    "-Dapp.mode=production",
    "-Dapp.data.dir={{ user.home }}/.my-app/data",
    "-D[mac]apple.laf.useScreenMenuBar=true",
    "-D[win]sun.java2d.d3d=false"
  ]
}
```

The `args` array can contain:

- **JVM arguments** - Memory settings, garbage collector options (`-Xmx4g`, `-XX:+UseG1GC`)
- **System properties** - Java system properties (`-Dapp.mode=production`)
- **Module arguments** - Module path and options (`-p /path/to/modules`, `--add-modules`

`javafx.controls`)

- **Program arguments** - Application-specific arguments

Platform-Conditional Arguments

Arguments can be platform-specific:

```
{
  "args": [
    "-D[mac]apple.laf.useScreenMenuBar=true",
    "-D[win,linux]file.separator=/",
    "-X[mac]dock:name=MyApp"
  ]
}
```

Supported platforms: `mac`, `win`, `windows`, `linux`

Examples

Environment-Specific Settings

Development:

```
{
  "args": [
    "-Xmx512m",
    "-Dapp.env=development",
    "-Dapp.debug=true"
  ]
}
```

+ Production:

```
{
  "args": [
    "-Xmx8g",
    "-Dapp.env=production",
    "-Dapp.debug=false",
    "-XX:+UseG1GC"
  ]
}
```

User-Specific Directories

```
{
  "args": [
    "-Dapp.data.dir={{ user.home }}/MyAppData",
```

```
    "-Dapp.logs.dir={{ user.home }}/MyAppData/logs",
    "-Dapp.cache.dir={{ user.home }}/MyAppData/cache"
  ]
}
```

Platform-Specific Libraries

```
{
  "args": [
    "-Djava.library.path={{ app.dir }}/native/mac",
    "-Dapple.laf.useScreenMenuBar=true",
    "-Xdock:name=MyApp"
  ]
}
```

Error Handling

Scenario	Behavior
Config file not found	Warning printed to stdout, execution continues
Invalid JSON	Warning printed to stderr, execution continues
Empty args array	No arguments added, execution continues
Config file property not set	No config file loaded, normal execution

System Properties

When a config file loads, the launcher sets:

```
-Djdeploy.config.file=/resolved/path/to/config.json
```

Applications can use this property to:

- Detect config file usage
- Read the config file path
- Implement custom config handling

Limitations

- Not editable in the GUI
- JSON format only
- Config file arguments are appended to package.json arguments

Technical Details

Launcher Changes

- Platform-specific privilege escalation integration
- Configuration file placeholder resolution
- Detection of desktop environment availability

Linux Installation

- Automatic PATH configuration for `~/.local/bin`
- Symlink creation with conflict detection
- Desktop environment detection without requiring full desktop installation

Error Handling

- Missing config files do not prevent launch
- Clear warnings for missing or malformed config files
- Fallback to package.json configuration

Migration Guide

Upgrading from 5.1

jDeploy 5.2 is backward compatible with 5.1. No changes are required for existing applications.

Configuring Administrator Privileges

1. Determine if elevation is needed
2. Choose "allowed" or "required"
3. Update package.json:

   ```
   {
     "jdeploy": {
       "runAsAdministrator": "allowed"
     }
   }
   ```

4. Test on each platform
5. Document the requirement for users

Configuring Runtime Configuration

1. Define which settings should be configurable
2. Choose a config file location
3. Update package.json:

```
{
  "jdeploy": {
    "configFile": "{{ user.home }}/.myapp/config.json"
  }
}
```

4. Provide a sample config file
5. Handle the `jdeploy.config.file` system property in your application if needed
6. Test with missing config files

Linux CLI Installation

The CLI installation option is available in 5.2 with no configuration required. To customize the command name, set the `bin` field in package.json:

```
{
  "name": "my-app",
  "bin": {
    "myapp": "jdeploy-bundle/jdeploy.js"
  }
}
```

Best Practices

Administrator Privileges

- Document why elevation is needed
- Test behavior when elevation fails
- Minimize code running with elevated privileges

Runtime Configuration

- Include defaults in package.json
- Document available configuration options
- Validate system properties at runtime
- Consider backward compatibility when changing config structure

Linux Deployment

- Test on systems with varying desktop environment configurations
- Detect graphical environment availability at runtime if needed

Known Issues

- Runtime config files must be created manually (no GUI editor)
- No schema validation for config files
- Only one config file per platform is supported

Resources

- **Official Website**: https://www.jdeploy.com/
- **Download**: https://github.com/shannah/jdeploy-desktop-gui/releases/latest
- **Documentation**: https://www.jdeploy.com/docs/manual/
- **GitHub Repository**: https://github.com/shannah/jdeploy
- **Support**: GitHub Issues and Discussions
- **Run as Administrator RFC**: https://github.com/shannah/jdeploy/blob/master/rfc/run-as-administrator-feature.md