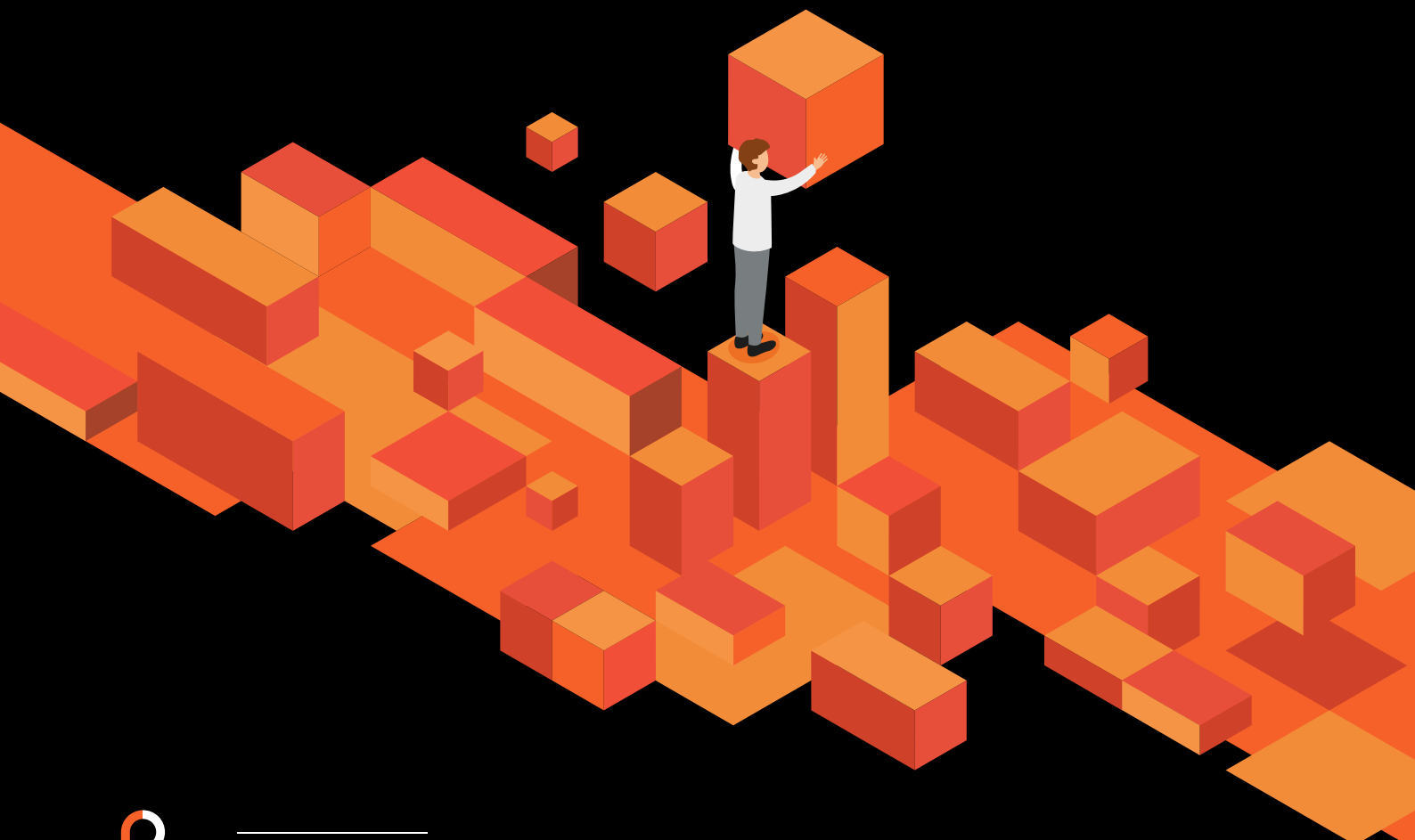


Technical Whitepaper

# GRASPING WEBSPHERE LIBERTY

—

A deep dive into  
IBM WebSphere Liberty  
and its possibilities



Platinum  
Business  
Partner



# TABLE OF CONTENTS

## Chapter 1 - Liberty as we know it

---

<b>1. History</b>	1.1 Software family	p4
	1.2 Introduction	p4

## Chapter 2 - Liberty at its core

---

<b>2. Core concepts and capabilities</b>	2.1 Liberty's purpose	p9
	2.2 Evolution to an open project	p9
	2.3 Flexible and modular configuration	p10
	2.4 Stability	p16
	2.5 Agility and reactivity	p17
	2.6 Security	p19
	2.7 Upgrades and migrations	p19
	2.8 Cloud-native	p20
	2.9 Troubleshooting	p22

## Chapter 3 - Liberty today

---

<b>3. Use cases</b>	3.1 Containers	p28
	3.2 Pak'ed up	p29
	3.3 Buildpacks	p30
	3.4 Pipeline integrations	p31
	3.5 Integration with existing software	p31

## Chapter 4 - Closing notes

---

<b>4. Closing notes</b>	4.1 Notes on the future of Java and Liberty	p33
	4.2 About the author	p34



# CHAPTER 1

-

# LIBERTY AS WE KNOW IT

# 1. History

## 1.1 Software family

IBM WebSphere, first introduced in 1988, is a suite of enterprise products enabling businesses to create, manage and integrate enterprise applications. Over time this software family has evolved beyond its original scope through ever-evolving business needs and iterations of improvements.

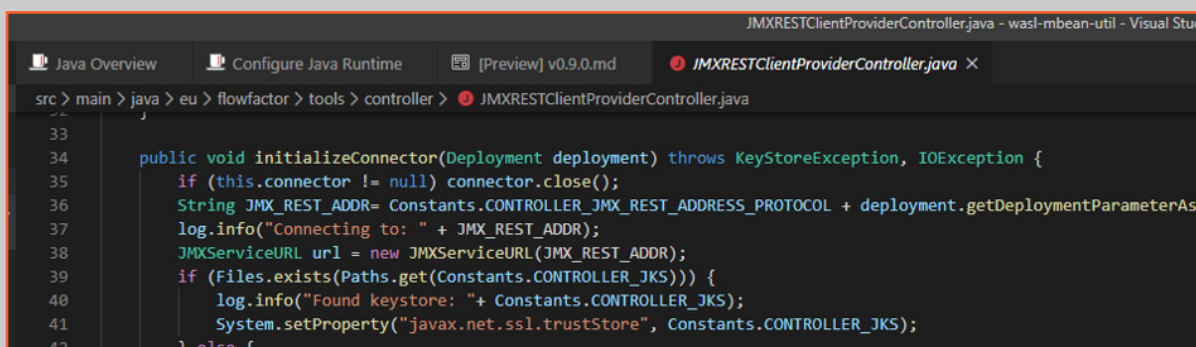
This whitepaper will mostly focus on **WebSphere Liberty** with some references to related/origin products such as WebSphere Application Server (WAS) and some utilities built around application migration and modernization.

## 1.2 Introduction

### 1.2.1 Prologue

Today, Java is still one of the most prominent programming languages used by businesses world-wide. In fact, **PYPL (PopularitY of Programming Language)** which is based on data provided by Google Trends shows that it is the second most popular programming language (bested by Python due to its wide range of application in the fields of Operating Systems, Artificial Intelligence, web development and general scripting).

Java is usually written in Integrated Development Environments (IDEs), tools which exist to simplify development. They provide syntax highlighting, code completion, integration with several plugins and development tools and much more. Commonly used IDEs for Java are Eclipse, IntelliJ, Visual Studio Code.



```
JMXRESTClientProviderController.java - wasl-mbean-util - Visual Studio Code
Java Overview  Configure Java Runtime  [Preview] v0.9.0.md  JMXRESTClientProviderController.java X
src > main > java > eu > flowfactor > tools > controller > JMXRESTClientProviderController.java
33
34     public void initializeConnector(Deployment deployment) throws KeyStoreException, IOException {
35         if (this.connector != null) connector.close();
36         String JMX_REST_ADDR= Constants.CONTROLLER_JMX_REST_ADDRESS_PROTOCOL + deployment.getDeploymentParameterAs
37         log.info("Connecting to: " + JMX_REST_ADDR);
38         JMXServiceURL url = new JMXServiceURL(JMX_REST_ADDR);
39         if (Files.exists(Paths.get(Constants.CONTROLLER_JKS))) {
40             log.info("Found keystore: "+ Constants.CONTROLLER_JKS);
41             System.setProperty("javax.net.ssl.trustStore", Constants.CONTROLLER_JKS);
42         } else {
```

Image 1:  
Example of  
Visual Studio  
Code

After writing code (.java files), it needs to be compiled to bytecode (.class files) so a JVM (Java Virtual Machine) can interpret and execute it. Fortunately for us, Java compilers exist for just such a purpose. IDEs usually provide a compiler, often via the Java Development Kit (JDK) and most also have a way to run and test the application with an embedded application server or servlet container.

Once compiled, the application can be packaged along with all necessary resources into a single file or archive. Several archive types exist, each a subset of the next:

- ▶ **JAR (A standard Java archive)**
- ▶ **WAR (Web archive - Java EE Web Profile specification)**
- ▶ **EAR (Enterprise archive - Java EE Full Profile specification)**

Typically, you choose an archive type based on your choice of application server or servlet container as well as your needs for EE functionalities and support.

It is however important to note that due to the evolution to microservices, the complexities and features of a monolithic application are nowadays broken down into smaller, more manageable parts. Since less features are required to create your application, you may find more JARs in a microservices landscape.

## — 1.2.2 Application server versus servlet container

A common misconception is that an application server (for example IBM's WebSphere Liberty) and a servlet container (for example Apache's Tomcat) are identical.

The biggest difference is that an application server is usually Java EE compliant, which means it has been approved for running enterprise applications and that it is able to provide all the EE standardized specifications and functions which have been reviewed meticulously via JSRs (Java Specification Requests) by the Java community. These standardized components are also rigorously tested to make sure they can work in parallel or collectively.

Whereas a servlet container (also known as a webcontainer) exists simply to run an application, it only implements the web component of the EE specified architecture and cannot provide the other components without having the developers implement them manually by providing third party libraries.

## 1.2.3 WebSphere flavors and supporting software

**WebSphere Application Server (Traditional or Network Deployment)**

**WebSphere Application Server Liberty Core**

**OpenLiberty**

**WebSphere Liberty**

The Websphere family is not limited to the products above, but these are the ones I will focus on in order to differentiate on their use cases.

The traditional WebSphere Application Server is typically installed on virtual machines as a standalone or distributed platform. In its distributed form it can span across multiple virtual machines, collectively referred to as a WebSphere Cell, providing functions like high availability and such. You can manage all the instances using a Deployment Manager (dmgr) and deploy applications and configuration to one or several nodes. Operating it may prove difficult without the necessary training or guides. Assemble a team to govern and maintain distributed clusters on a wide server landscape.

WebSphere Application Server Liberty Core is a lightweight Liberty-based edition. With WebSphere Application Server Liberty Core, you can rapidly build and deliver web applications that do not require the full Java EE stack, since it contains functionalities to support the Java EE Web Profile which is a smaller subset.

Finally, we arrive at WebSphere Liberty, which is a fully featured application server with a very small footprint, ideal for developers because it is fast (less than 2 seconds start-up time thanks to a modular approach) and easy to use. Production-ready because it supports the Java EE full specification, has high availability, scaling and many other features.

Many quality-of-life features have been added to provide easier integrations with other development tools. WebSphere Liberty is free to use for development purposes, running it as a production application server requires a license.



## — 1.2.4 Accelerating and evolving

Harder, better, faster, stronger: this is how tomorrow's applications are built. My apologies for (mis)quoting Daft Punk but they were spot on. The demands for building applications which are robust, secure, integrated and fully featured are increasing whilst time-to-market should be decreased when possible. Crudely put, this goal is simply unattainable when using the development tools and strategies most companies have been using for the last decade. Hence, we must accelerate and evolve along with our tools. Liberty can be a part of that solution and you will find out why in the next chapters.

I highly recommend reading the **twelve-factor app** methodology as well. Keep in mind that such changes are not realized in a day, it should be a common and longer-term goal to work towards. Development, security, operations and all involved parties have to work together towards it.

# CHAPTER 2

-

# LIBERTY AT ITS CORE





# 2. Core concepts and capabilities

## 2.1 Liberty's purpose

We already touched base on Liberty's purpose in the previous chapter; providing an EE full profile application server which can help improve agility whilst maintaining the quality one needs and deserve during the development process.

IBM has worked tirelessly on bringing this technology up to the highest standards, testing it thoroughly and providing it to corporations and consumers.

I have mentioned Liberty is modular, stable and agile and it is long overdue for me to explain why, in-depth and concrete. Also, try Liberty for yourself to find out if it fits your use case.

## 2.2 Evolution to an open project

As it became more apparent that Liberty was the way to go forward, IBM open sourced the base source code making everyone able to improve and expand upon it. Open source is not the route they have chosen for their products in the past. This shows that IBM itself is also evolving to accelerate. Most products in their catalog have now also been containerized, even the large software suites.

Download OpenLiberty from <https://openliberty.io/> or the WebSphere Liberty development kit from <https://developer.ibm.com/wasdev/websphere-liberty/>, the latter built upon the core OpenLiberty provides.

The difference between OpenLiberty and WebSphere Liberty can be shown with a very detailed graph, or I could simply say that WebSphere Liberty has mechanisms to set up clustering, health management and monitoring and introduces a couple of advanced authentication features such as SAML, OpenID and similar.

Here is said graph either way:

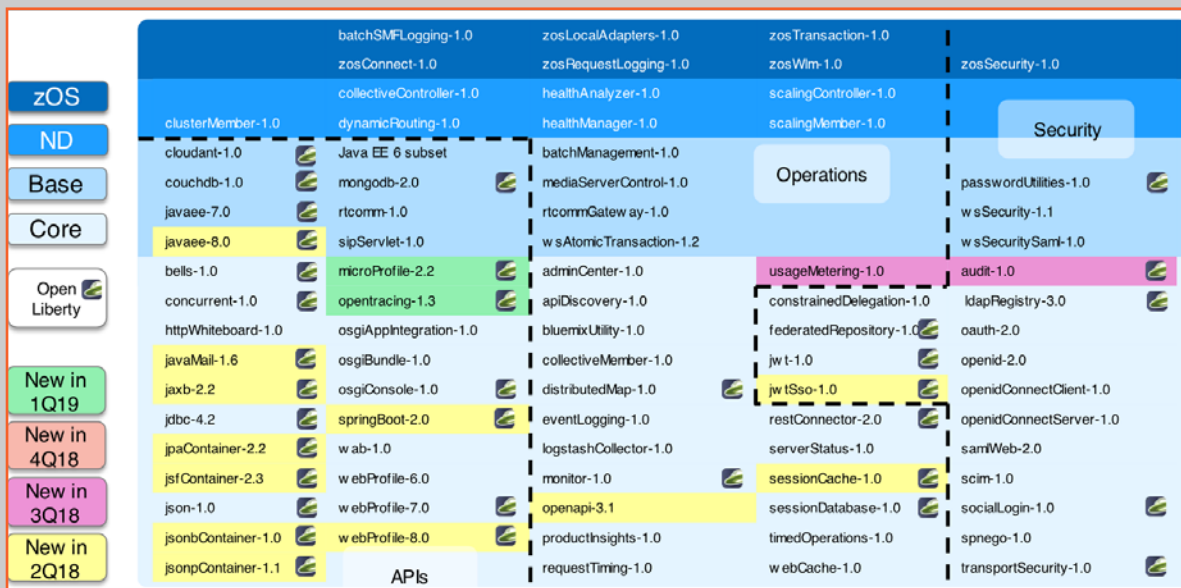


Image 2: Difference OpenLiberty and Websphere-Liberty

## 2.3 Flexible and Modular configuration

### 2.3.1 Features

Liberty offers modularity in the form of features (shown in previous graph). Each Liberty server has a **server.xml** file which can be used to completely configure the environment.

```

3      <!-- Enable features -->
4      <featureManager>
5          <feature>adminCenter-1.0</feature>
6          <feature>batchManagement-1.0</feature>
7          <feature>j2eeManagement-1.1</feature>
8          <feature>jca-1.7</feature>
9          <feature>jms-2.0</feature>
10         <feature>jndi-1.0</feature>
11         <feature>localConnector-1.0</feature>
12         <feature>mdb-3.2</feature>
13         <feature>jsp-2.3</feature>
14         <feature>wasJmsClient-2.0</feature>
15     </featureManager>
    
```

Image 3: server.xml to configure Liberty server

Some of these features may look familiar because they are in fact part of the official EE specifications. For example, **jms-2.0**:

```

Version history [edit]
• JMS 1.0[5]
• JMS 1.0.1 (October 5, 1998)[6]
• JMS 1.0.1a (October 30, 1998)[6][7]
• JMS 1.0.2 (December 17, 1999)[8]
• JMS 1.0.2a (December 23, 1999)[9]
• JMS 1.0.2b (August 27, 2001)[10]
• JMS 1.1 (April 12, 2002)[11]
• JMS 2.0 (May 21, 2013)[12][13]
• JMS 2.0a (March 16, 2015)[14][15]

JMS 2.0 is currently maintained under the Java Community Process as JSR 343.[16]
JMS 3.0 is under early developmentⓘ as part of Jakarta EEⓘ.
    
```

Image 4: Wikipedia source JMS versions



Other optional features such as the **adminCenter-1.0** have been added by IBM to facilitate certain operations or extend upon the core features. In this case the adminCenter adds a user interface capable of stopping and starting or even configuring the servers.

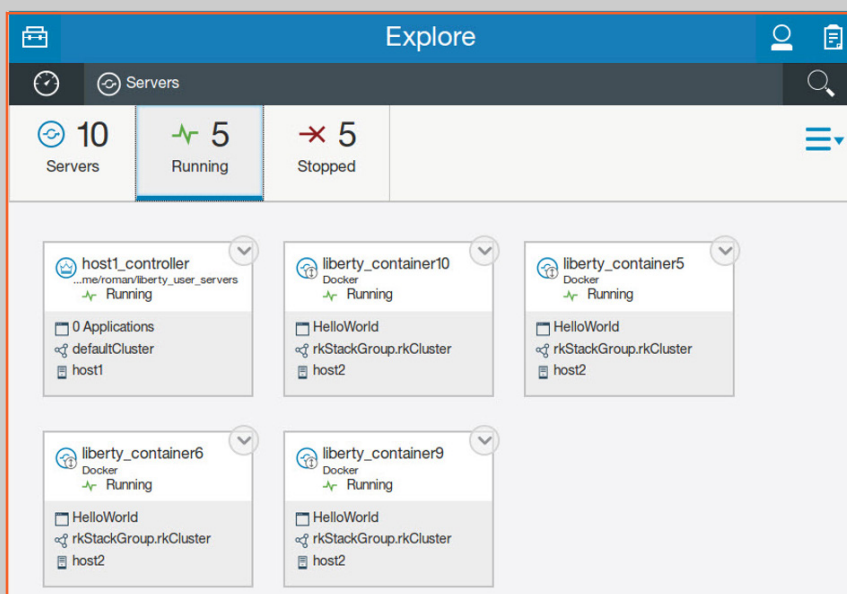


Image 5: adminCenter-1.0 at work.

Each IBM product has a Knowledge Center and the full and up-to-date feature list is available [here](#)

After adding a feature it stands to reason that one might want to configure some of these features. This is possible with the server.xml. After adding **jdbc-4.2** some configurations are required: a connection pool to set a minimum and maximum amount of connections as well as a datasource to connect to. In this example we are connecting to an Oracle database, so we are also adding an Oracle JDBC driver as library.

```
<!-- ConnectionManager -->
<connectionManager agedTimeout="-1" connectionTimeout="3m" id="dbConnectionPoolManager" maxIdleTime="5m" maxPoolSize="20" minPoolSize="1" purgePolicy="EntirePool" reapTime="3m"/>
<!-- DataSource -->
<dataSource connectionManagerRef="dbConnectionPoolManager" id="dbNameDS" jndiName="jdbc/dbnameDS" statementCacheSize="10">
  <jdbcDriver libraryRef="OracleLib"/>
  <properties.oracle URL="jdbc:oracle:thin:@/dbhost.example.com:1529/dbname" password="password" user="user"/>
</dataSource>
<!-- DataSource Library configuration -->
<library id="OracleLib">
  <file name="{shared.resource.dir}/oracle/jdbc-12.1.0.2.jar"/>
</library>
```

Image 6: Code example: connecting to an Oracle database

If the server.xml is getting rather large, simply split the configuration using **XML includes**:

```
<!-- FEATURES -->
<include optional="true" location="features.xml" />

<!-- LOGCONFIG -->
<include optional="true" location="logconfig.xml" />

<!-- CONNECTIONMANAGERS -->
<include optional="true" location="connectionmgrs.xml" />
```

Image 7:  
Code example:  
Configuration  
split

This keeps the configurations clean and gives a grouped overview.

Another important thing to keep in mind is that the configuration loaded by the XML files is **dynamically loaded**. This means that changing configuration will immediately take effect, even if the application is running. Adding a new feature will also install it and update the server configuration on the fly.

```
[INFO] [AUDIT ] CWWKG0016I: Starting server configuration update.
[INFO] [AUDIT ] CWWKG0017I: The server configuration was successfully updated in 0,414 seconds.
[INFO] [AUDIT ] CWWKF0012I: The server installed the following features: [localConnector-1.0].
[INFO] [AUDIT ] CWWKF0008I: Feature update completed in 0,456 seconds.
```

Image 8:  
Code example:  
Configuration  
update

## 2.3.2 Structure

After downloading a WebSphere Liberty development kit, whether it be the full EE8 platform or a stripped-down kernel version, there will be a standard folder structure. It helps to familiarize with this structure a bit to know where to find things, troubleshoot or customize the installation.

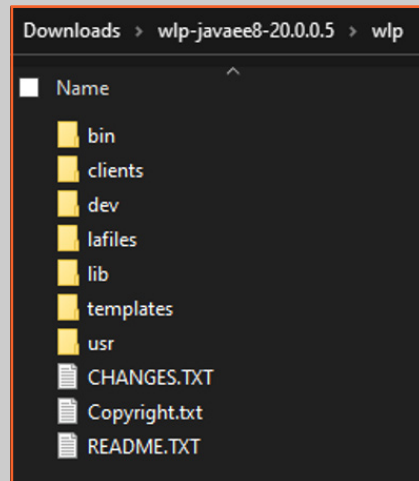


Image 9:  
Standard folder  
structure  
WebSphere  
Liberty

The **bin** folder contains a set of binaries or tools used to manage the installation, encode passwords, create a server or cluster of servers or download additional features.

```
..-20.0.0.5/wlp
[18:29][faan@DESKTOP-ONGRMTV:wlp]$ bin/productInfo featureInfo
appClientSupport-1.0 [1.0.0]
appSecurity-2.0 [1.0.0]
appSecurity-3.0 [1.0.0]
appSecurityClient-1.0 [1.0.0]
batch-1.0 [1.0.0]
beanValidation-2.0 [1.0.0]
cdi-2.0 [1.0.0]
collectiveMember-1.0 [1.0.0]
concurrent-1.0 [1.0.0]
distributedMap-1.0 [1.0.0]
ejb-3.2 [1.0.0]
ejbHome-3.2 [1.0.0]
ejbLite-3.2 [1.0.0]
ejbPersistentTimer-3.2 [1.0.0]
ejbRemote-3.2 [1.0.0]
el-3.0 [3.0.0]
FederatedRegistry-1.0 [1.0.0]
j2eeManagement-1.1 [1.1.0]
jacc-1.5 [1.0.0]
jakartaee-8.0 [8.0.0]
jaspic-1.1 [1.0.0]

[18:29][faan@DESKTOP-ONGRMTV:wlp]$ bin/securityUtility encode supersecret_password
xor}Lc0v0i0s0jw0isALz4sLCgwLTs=
```

Image 10:  
Bin Folder in  
Websphere  
Liberty folder  
structure

The **clients** folder contains client connectors which can be used to connect to the running server. For example, if you enable the restConnector-1.0 feature you can use the restconnector.jar in combination with a jmx:rest client such as VisualVM to get information about the JVM or trigger management operation such as maintenance mode.

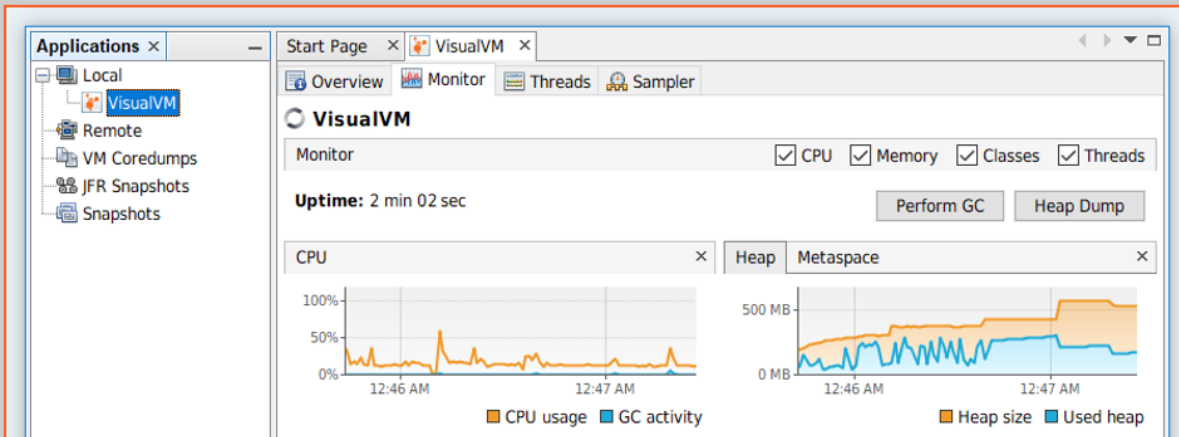


Image 11: VisualVM

Add a truststore to the configuration to make this work.

The **dev** folder contains JAR files and Javadoc which relates to features in Liberty. The JAR files are provided only for compilation of applications and features.

The **lfiles** folder contains License Agreement information.

The **lib** folder contains the actual platform runtime environment files.

The **templates** folder contains a few very basic templates for creating a server or client.

The **usr** folder is where the magic happens.

Subfolder **shared** is used for resources which should be shared between all servers.

Subfolder **servers** is where the actual servers will be. Initially this folder will be empty until the first server is created by using `bin/server create <servername>`. Start your server by using `bin/server start <servername>`.

A folder will be created which corresponds to the chosen name, with following structure:

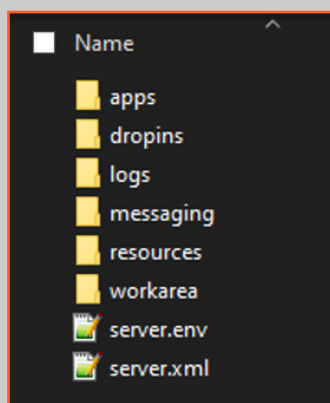


Image 12: Folder structure of the usr folder

**Apps** should contain the packaged application (JAR/WAR/EAR).

**Dropins** allows to literally “drop in” a new version of an application or configuration file.

**Logs** serves a pretty obvious purpose, **messages.log** usually being the most interesting one. We will go deeper into the topic of logging in the troubleshooting chapter.

**Messaging** contains a couple of files which are used as storage for persistent JMS messages. These are configurable in the server configuration. Best not to touch any of the files in this folder unless you want to delete them to create an entirely new message store.

**Resources** contains security-based files. Server identity files and trusted servers are generally separated using a **keystore** (identity) and a **truststore** (trusted servers). There will also be a file called `ltpa.keys` here, which the server uses to distribute LTPA tokens. After creating a Liberty Collective (it is possible to simulate having a clustered setup in a devkit!) there will also be separate security files for collective identity and collective trust verification.

The **workarea** is where the server’s work is being done. Best not to touch any of the files here.

The **server.env** file can be used to set environment variables.

We discussed the **server.xml** file before, this is where almost everything can be configured: which features to use, which configuration the features should use, port bindings ...

There are 2 files which don’t exist by default, but it might be useful to create them:

- jvm.options** ▶ allows setting JVM options such as min/max heap size or a custom garbage collector.
- bootstrap.properties** ▶ can contain attributes that affect the configuration and initialization of the runtime core. You may want to disable a specific engine in order to provide your own. These properties are not dynamic and are only loaded on server start.



## 2.4 Stability

In a quickly evolving development landscape, there needs to be a way to logically separate bleeding edge features which one might want to test, from the stable, production ready application builds and application servers.

I have already hinted at most of the pillars providing stability for WebSphere Liberty, but I will summarize them here and add a few extras.

- ▶ EE Liberty feature versions correspond to the Java EE specification, so there is no chance of suddenly using a different version of an implementation.
- ▶ IBM subjects the code base to thorough testing.
- ▶ WebSphere Liberty is built upon the OpenLiberty core, this means any changes to the application server's code have already been tested and used in OpenLiberty first.
- ▶ The Liberty repository (<https://developer.ibm.com/wasdev/downloads/>) holds a collection of feature versions, snippets and tools one can inject into a project. Subscribe to the beta repository of features to test the latest features while they are still under development. By testing beta features and providing feedback or code fixes, you yourself can contribute to the stability of the product.
- ▶ Feedback from developers around the world contributes to the expansion and general stability of the product.
- ▶ Feedback from enterprise consumers also reaches IBM via the official IBM support channels. If any defects are found, fixes are slipstreamed into the OpenLiberty core (if applicable) and/or the WebSphere Liberty code base in the next version.  
<https://www.ibm.com/support/pages/fix-list-ibm-websphere-application-server-liberty-continuous-delivery>

IBM has a very long track record of creating and maintaining large enterprise application suites and they aim at providing the same level of premium support for Liberty.





## 2.5 Agility and reactivity

Agility comes in many forms and it is not one but the combination of many principles that makes a development process agile. Having Liberty or OpenLiberty in your toolbox can help significantly but keep in mind to invest in the processes around it as well.

Intuitive issue tracking, code review and code quality tools, automated unit and integration tests (and other), security and vulnerability checks, pipelines which facilitate the build process, semantic versioning, release and deploy processes are all very important in today's agile approach.

Liberty starts up within 2 seconds which is great, but not enough to make our process agile and reactive, so let's take a look at what we can do to speed up our development process.

IBM has built several plugins and extensions for IDEs out there. At their core, all of them allow running freshly developed code on Liberty or OpenLiberty from within the IDE, instantly allowing access to an application and to see the results of changes made.

In the last years, some of these plugins have gained some amazing features. The Open Liberty Tools extension in Visual Studio code for example can now run tests and debug an application while running the application on an embedded server. Whether it be a more traditional JEE application or a microservice does not matter, support for both is built-in.

Creating additional tests will automatically compile the new test and it will be available for the next test run. Add the `-DhotTests=true` parameter to enable **hot testing** and run tests automatically after every change.

Similarly, enable **hot debugging** which, after putting breakpoints in the code, allows to pausing when a specific line of code is reached in the application running inside the embedded Liberty. This way values can be inspected step by step.

```
16 @Override
17 public HealthCheckResponse call() {
18     MemoryMXBean memBean = ManagementFactory.getMemoryMXBean();
19     long memUsed = memBean.getHeapMemoryUsage().getUsed();
20     long memMax = memBean.getHeapMemoryUsage().getMax();
```

Image 13: Add a breaking point

```
.vscode > {} launch.json > Launch Targets > {} Debug (Attach)
1  {
2      // Use IntelliSense to learn about possible a
3      // Hover to view descriptions of existing att
4      // For more information, visit: https://go.m
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "type": "java",
9              "name": "Debug (Attach)",
10             "request": "attach",
11             "hostname": "localhost",
12             "port": "7777"
13         },
```

Image 14: Attaching the debugger to the embedded server

There is also a Maven plugin, called `liberty-maven-plugin`, which can start up an embedded Liberty in dev mode, enabling to live test changes. Under the hood the plugin listens for changes made in a Java or configuration file and updates or recompiles it. Pressing 'Enter' while in dev mode can also run tests on demand. Meanwhile, since the server supports dynamic loading, changing or adding features or even making changes to Maven dependencies imply works without having to restart the embedded server.

## 2.6 Security

Common Vulnerabilities and Exposures (CVE) which impact Liberty's security, are reviewed and patched in the next version. There is a **[maintained list to track patched CVE's](#)** online.

Next to this, you can subscribe to **[IBM's security bulletins](#)**. Stay informed, stay safe.

**Passwords** in the XML configuration **can be encoded using the securityUtility** provided in each Liberty server's bin folder.

## 2.7 Upgrades and migrations

Starting 2016 WebSphere Liberty changed from a traditional release model which was linked to WebSphere Application Server (Traditional or Network Deployment) to a decoupled continuous release model (initially version 16.0.0.2) with OpenLiberty at its core. The first number of each version corresponds to the year of the release, while the last number used to correspond to the release quarter. Nowadays the continuous model releases even more swiftly and the last number corresponds to the month of the release. (year.0.0.month).

- ▶ **WebSphere Liberty fix list:**  
**<https://www.ibm.com/support/pages/fix-list-ibm-websphere-application-server-liberty-continuous-delivery>**  
The fix list includes which bugs and CVEs have been patched.
- ▶ **WebSphere Liberty changelog:**  
**[https://www.ibm.com/support/knowledgecenter/SSEQTP\\_liberty/com.ibm.websphere.wlp.doc/ae/rwlp\\_newinrelease.html](https://www.ibm.com/support/knowledgecenter/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_newinrelease.html)**  
The changelog includes which new features have been added.

## 2.8 Cloud native

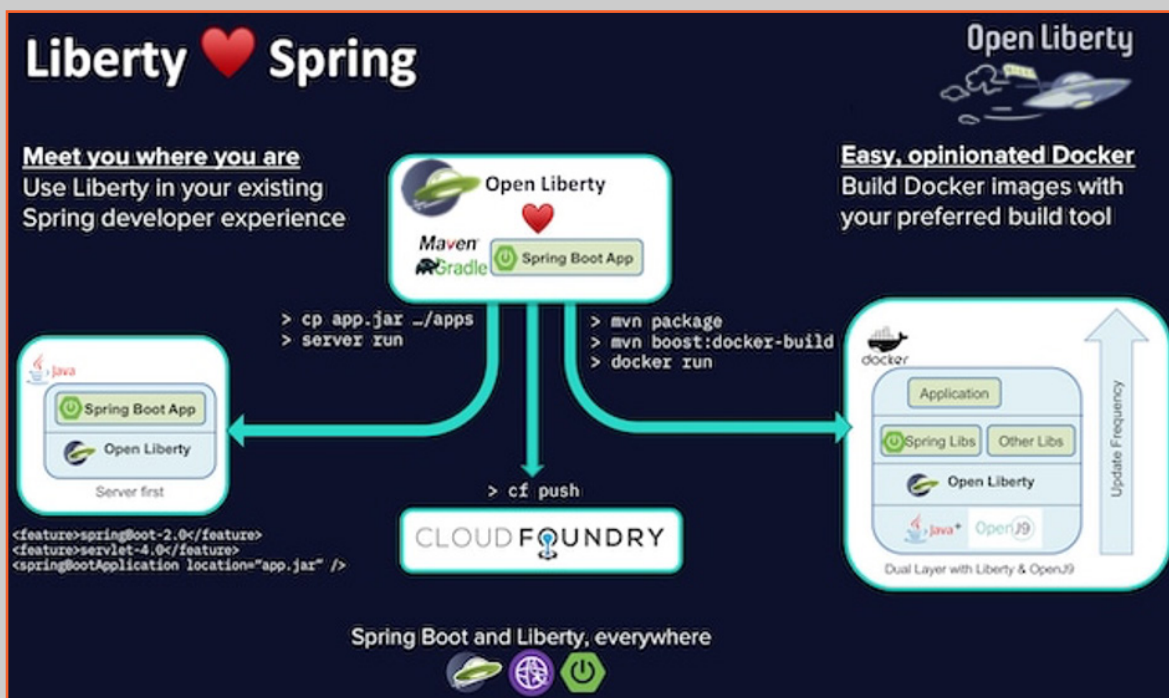
Liberty has a rich set of features which have been added to support cloud native development:

- ▶ `jakartaee-8.0` ([notes on the future of Java](#))
- ▶ **microProfile: The Eclipse MicroProfile (MP)** project is an open collaboration between developers, the community, and vendors to create a programming model for microservice applications which is complementary to JEE.
  - **mpConfig:** The MicroProfile Config API enables application configuration properties from multiple sources to be amalgamated into a single set of configuration properties and accessed by using a single client API. Different sources of configuration properties can be given different priorities. Higher priority sources can then override property values from lower priority sources. Allows for a library or application developer to package code with associated configuration settings that can then be overridden during application assembly, installation, or at runtime in response to events.
  - **mpFaultTolerance:** Aimed at increasing robustness for the application and others communicating with it. Contains the following mechanisms:
    - Timeout: Define a duration for timeout
    - Retry: Define a criteria on when to retry
    - Fallback: provide an alternative solution for a failed execution.
    - CircuitBreaker: offer a way of fail fast by automatically failing execution to prevent the system overloading and indefinite wait or timeout by the clients.
    - Bulkhead: isolate failures in part of the system while the rest part of the system can still function.
  - **mpHealth:** Allows easily adding Liveness and Readiness probes, which are specifications needed to ensure the health and operability of a microservice.
  - **mpOpenTracing:** Enable and customize tracing of JAX-RS and non-JAX-RS methods. When fed to a distributed tracing system, a complete picture of each call can be traced from start to finish: how long it took, which endpoints were used (across multiple microservices implementing this specification), diagnose problems and assess load distribution across a microservice landscape.
  - **mpOpenAPI:** OpenAPI provides a language-agnostic interface for both humans and computers. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and has many other use cases.
  - **mpMetrics:** Provides a `/metrics` REST interface. Developers can add their own custom metrics using the MicroProfile metrics API alongside the metrics provided by Liberty.

- **mpRestClient:** Create type-safe REST services
- **mpJwt:** Build and consume JSON Web Token (JWT) tokens, which can be used to propagate user identity or tokens.
- **mpReactiveStreams:** Reactive Streams Operators provide flow control and elegant error handling when subscribing to and processing streams of events (Publisher, Subscriber, Subscription and Processor interfaces).
- **mpReactiveMessaging:** Reactive Messaging provides an easy way to send and receive messages within and between microservices using Kafka message brokers.
- **mpContextPropagation:** CompletionStage and CompletableFuture (SE8) enable chaining together pipelines of dependent actions, where execution of each dependent stage is triggered by the completion of the stage(s) upon which that stage depends. Context Propagation enhances the concurrency capabilities for dependent stages that run with predictable thread context regardless of which thread the completion stage action ends up running on.

For these and other features exists a bunch of how-to guides with code snippets:  
<https://openliberty.io/guides/>

IBM has also shown love for developers who use Spring Boot to create their applications. With the springBoot-2.0 feature, developers can easily embed OpenLiberty instead of the default Tomcat and build a Docker container.



▶ Image 15:  
 IBM's  
 springBoot-2.0  
 feature

## 2.9 Troubleshooting

### 2.9.1 Logging

Logs are the first things to verify when things go wrong. Inside the logs folder there are multiple files so let's go over them.

**console.log** - This file contains the redirected standard output and standard error streams from the JVM process. The logging component writes major events to the standard output stream and errors to the standard error stream when using the default `consoleLogLevel` configuration. The standard output and standard error streams always contain messages that are written directly by the JVM process, such as `-verbose:gc` output.

**messages.log** - This file contains all messages that are written or captured by the logging component. All messages that are written to this file contain additional information such as the message timestamp and the ID of the thread that wrote the message. This file does not contain messages that are written directly by the JVM process.

**Note:** since both `console.log` and `messages.log` will both capture standard output and error, one might struggle with disk space when running the server for a long time with a high amount of logging. Especially because `console.log`, which is more of a system log, cannot be configured as a rotating log. `Messages.log` however can be configured much more fine grained, plus it contains timestamps for standard output/error which is nice while analyzing them. One way to mitigate this, is to set `copySystemStreams` to `false`. This way, important JVM logs will still be kept in `console.log`, but it will not grow very large over time due to standard out/error logging and there will still be standard output/error in `messages.log`.

**trace.log** - This file contains all messages that are written or captured by the logging component and any additional trace. This file is created only after enabling additional trace. This file does not contain messages that are written directly by the JVM process.

Within this folder there could also be an **ffdc** (First Failure Data Capture) folder, this feature preserves the information that is generated from a processing failure and returns control to the affected engines. FFDC entries are datetime-stamped and only hold non-applicative errors. An FFDC entry can then be provided to a tech-savvy person or to IBM support for analysis.

**Configuring specific traces** can be done via the `server.xml` (or `includes`) as such:

```
<logging traceSpecification="com.example.package.*=finest"/>
```

The configurable log levels are `off`, `fatal`, `severe`, `warning`, `audit`, `info`, `config`, `detail`, `fine`, `finer`, `finest`, `all`.

You can specify **multiple traces** by separating them using `“:”` (colon) as delimiter:

```
<logging traceSpecification="com.example.package.model.*=fine: com.example.package.service.login.*=audit"/>
```

One can enable **audit logging**. These logs are encrypted and signed and can be decrypted and unsigned using the **auditUtility** in combination with the correct keys.

## 2.9.2 Slow JDBC calls

The **timedOperations-1.0** feature can help troubleshoot JDBC operations running in the application server. This feature periodically (configurable) generates a report that contains the 10 longest JDBC calls. Using a server dump (see 2.1.9.5) will also generate this report.

## 2.9.3 Context requests

The **eventLogging-1.0** feature can show all the application context requests and total time it took to receive, execute and respond. Each request has a unique request ID and the context information that helps the user to understand the request-specific data. Enabling this feature can generate overhead because it can be a lot of extra information that needs to be fetched and logged.

```
TaIVDTO_AAAAAAAAAAK request ID and TradeWeb context:
# contextInfo=TradeWeb | TradeScenarioServlet
# contextInfo=TradeWeb | TradeAppServlet
# contextInfo=TradeWeb | /quote.jsp
# contextInfo=TradeWeb | /displayQuote.jsp
cuteQuery # contextInfo=jdbc/TradeDataSource | select * from quoteejb q where q.symbol=?
teQuery # contextInfo=jdbc/TradeDataSource | select * from quoteejb q where q.symbol=? # duration=12.537ms
contextInfo=TradeWeb | /displayQuote.jsp # duration=22.171ms
# contextInfo=TradeWeb | /displayQuote.jsp
cuteQuery # contextInfo=jdbc/TradeDataSource | select * from quoteejb q where q.symbol=?
teQuery # contextInfo=jdbc/TradeDataSource | select * from quoteejb q where q.symbol=? # duration=4.968ms
contextInfo=TradeWeb | /displayQuote.jsp # duration=12.569ms
contextInfo=TradeWeb | /quote.jsp # duration=152.752ms
contextInfo=TradeWeb | TradeAppServlet # duration=154.616ms
contextInfo=TradeWeb | TradeScenarioServlet # duration=158.283ms
```

▶ Image 16:  
eventLogging-  
1.0 feature

## 2.9.4 Slow and hung requests

The **requestTiming-1.0** feature can detect requests which are slow or stuck.

Slow requests: When a request runs for longer than it was configured (default 10s), a warning message is written in the messages log file.

```
[12/1/14 11:58:09:629 IST] 0000001d com.ibm.ws.request.timing.SlowRequestTimer
W TRAS0112W: Request AABjnS+1In0_AAAAAAAAAAb has been running on thread 00000021 for at least 10003.571ms.
The following stack trace shows what this thread is currently running.
```

```
at java.util.HashMap.getEntry(HashMap.java:516)
at java.util.HashMap.get(HashMap.java:504)
at org.apache.derby.iapi.store.access.BackingStoreHashtable.get(Unknown Source)
at org.apache.derby.impl.sql.execute.HashScanResultSet.getNextRowCore(Unknown Source)
at org.apache.derby.impl.sql.execute.NestedLoopJoinResultSet.getNextRowCore(Unknown Source)
at org.apache.derby.impl.sql.execute.ProjectRestrictResultSet.getNextRowCore(Unknown Source)
at org.apache.derby.impl.sql.execute.DMLWriteResultSet.getNextRowCore(Unknown Source)
at org.apache.derby.impl.sql.execute.DeleteResultSet.setup(Unknown Source)
at org.apache.derby.impl.sql.execute.DeleteResultSet.open(Unknown Source)
at org.apache.derby.impl.sql.GenericPreparedStatement.executeStmt(Unknown Source)
at org.apache.derby.impl.sql.GenericPreparedStatement.execute(Unknown Source)
at org.apache.derby.impl.jdbc.EmbedStatement.executeStatement(Unknown Source)
at org.apache.derby.impl.jdbc.EmbedPreparedStatement.executeStatement(Unknown Source)
at org.apache.derby.impl.jdbc.EmbedPreparedStatement.executeUpdate(Unknown Source)
at com.ibm.ws.rsadapter.jdbc.WSJdbcPreparedStatement.executeUpdate(WSJdbcPreparedStatement.java:626)
```

▶ Image 17:  
requestTiming-  
1.0 feature



**Hung request:** If the request exceeds the configured threshold value (default 10m), details about the request and events that made up the request are captured. For a hung request detection, a series of three thread dumps (javacores) is taken with a 1-minute delay between them. The following log message sample shows the log messages for a request that crossed the hung request detection threshold.

```
[WARNING ] TRAS0114W: Request AAA7W1pP717_AAAAAAAAAA was running on thread 00000021 for at least 240001.015ms. The following table shows the events that have run during this request.
Duration      Operation
240001.754ms + websphere.servlet.service | TestWebApp | TestServlet?sleepTime=480000
0.095ms       websphere.session.setAttribute | mCzBMyzMvAEjMJjX9zQYIw | userID
0.007ms       websphere.session.setAttribute | mCzBMyzMvAEjMJjX9zQYIw | visitCount
```

▶ Image 18:  
hung  
request

## 2.9.5 Server dumps

Server dumps can be very helpful in analyzing problems otherwise not easily identified, such as memory leaks, deadlocks, hung threads and similar problems. IBM support might also request one for analysis.

There are different kinds of information collection sets (=dumps) available. When a server crashes, it might already create a relevant one in the server directory using default settings. IBM also provides a command line option to generate one on demand.

### 2.9.5.1 Types and categories

Firstly, Java has specific types of dumps available:

A **thread dump** collects information about the state of all threads that are part of the running process. The “stack” of each thread at the time of the snapshot will be collected in a readable format. Some threads will be those of the running application(s), some will be those of the JVM or internal components. Note that Liberty has a **self-tuning threadpool** and in most cases there is no need to customize it. Gather this information when experiencing deadlocks, performance issues or hung threads.

A **heap dump** collects information about the Java memory heap. This snapshot contains information on the objects in the stack like object types, object sizes, their values and memory allocations. A heap dump is automatically generated when running out of memory. Gather this information when experiencing OutOfMemory exceptions, memory leaks or abnormal object/heap growth.



**Be advised:** There are some risks involved for your environment when creating a heap dump so it is important to understand what happens: assuming there are memory problems, chances are that the maximum Java heap (-Xmx) has been reached. To create a snapshot of this memory, the operating system must load it in memory (outside the JVM heap) in order to write it to disk. Depending on the size of the application's max heap (-Xmx), the OS will need (apart from its usual functions) sufficient memory to process this and sufficient disk space to create the file.

- ▶ If there is not enough memory for the OS's core functions, it might decide to kill a process which consumes a large amount of memory, like the application server or the process performing the heap dump.
- ▶ If there is not enough disk space, many processes will stop functioning correctly.

In short, provide extra memory and disk space scaling equally with Xmx to create heap dumps.

A **system dump**, also known as a **core dump**, typically contains information about the system itself. In case of Liberty, information about the kernel and runtime, all its components, configurations, libraries and resources are included. A core dump is also automatically generated when the process ends abnormally.

**Be advised:** system dumps can be very large on disk because they include the complete packaged runtime. If there is not enough disk space, many processes will stop functioning correctly.

## .....2.9.5.2 Generating server dumps

IBM has chosen to split the creation of dump files on demand into 2 categories:

1. The **JVM snapshot** creates a javacore file. Javacore files are specifically created and formatted for debugging purposes and mostly contain information to mitigate specific types of problems such as 100% CPU usage, crashes, performance problems or memory problems. Javacores are typically not very large but contain useful information about the JVM, the environment, loaded libraries, locks and garbage collection.

They can be generated automatically or generated:

```
root(faan) /mnt/c/DevOps/Liberty/wlp/bin 16:50:09
> ./server javadump defaultServer

Dumping server defaultServer.
Server defaultServer dump complete in /mnt/c/DevOps/Liberty/wlp/usr/servers/defaultServer/javacore.20200613.165013.505.0003.txt.
```

▶ Image 19:  
Code example:  
generating  
Javacores

Include heap and/or system dumps with the --include=heap,system option.

The **server snapshot** is mostly used for server-related problems because it contains server configuration, logs, includes the server's work area and some information about the deployed application.

```
root(faan) /mnt/c/DevOps/Liberty/wlp/bin 17:01:41
> ./server dump defaultServer

Dumping server defaultServer.
Server defaultServer dump complete in /mnt/c/DevOps/Liberty/wlp/usr/servers/defaultServer/defaultServer.dump-20.06.13_17.01.44.zip
```

▶ Image 20:  
Code example:  
server dump

Include thread, heap and/or system dumps with the --include=thread,heap,system option.

### .....2.9.5.3 Analysis

Analyzing the generated files can be difficult depending on the type and knowledge of the subject.

To analyze thread dumps, it helps to have knowledge of how threads and executors work. To analyze heap dumps, it helps to know how object hierarchy works and how they interact.

In addition, for both of the above cases, knowing the application helps a lot. Specifically, knowing which threads and objects are created under which circumstances, their places in the hierarchy and expected values can help identify problems faster.

Heap dumps can be loaded into external tools such as **IBM's Heap analyzer** or **Eclipse's Memory Analyzer** for further inspection.

To analyze system or core dumps, provide them to IBM support and they will provide feedback on the encountered issues.

### 2.9.6 Error codes

The default error codes can be found on **this page**. Most of the time additional messages will be shown along with the error codes in the console log so there will be no need to search for them here.





# CHAPTER 3

## - LIBERTY TODAY

# 3. Liberty today

## 3.1 Use cases

There are several ways to use Liberty, OpenLiberty or WebSphere Liberty:

- ▶ embedded in an IDE to speed up the development process
- ▶ on standalone or distributed (=Liberty Collective, this is a paid feature) servers.
- ▶ in docker containers
- ▶ on container platforms
- ▶ on any Cloud provider
- ▶ using build packs
- ▶ in pipelines
- ▶ ...

Several integrations are compatible or built for WebSphere Liberty or OpenLiberty. In this chapter, we will discuss the most common use cases.

### 3.1.1 Containers

Liberty, due to its quick startup time, dynamic loading capabilities and low overhead is an ideal application server to run in containers.

On Dockerhub there are [container images for OpenLiberty](#) as well as [container images for WebSphere Liberty](#) (Ubuntu + IBM jdk). The latter has specific arguments to enable enterprise capabilities (for which a license is needed). If one aims at creating enterprise software to run on a production container platform, I advise using the [container images for WebSphere Liberty on the Red Hat Universal Base Image](#) (UBI).

Instructions on how to build an application container with these images are included:

<https://github.com/openliberty/ci.docker#building-an-application-image>

The [OpenLiberty guides](#) are also a great way of learning how to leverage certain cloud-native features within Liberty.

Ideally a container runs on top of a Kubernetes certified container platform. IBM generally recommends using RedHat Openshift (OCP) with an additional Cloud Pak installed.

## 3.2 Pak'ed up

IBM has introduced several products called Cloud Paks for large enterprises wanting to move to a Kubernetes based cloud, on-premise or hybrid solution. Using Red Hat Openshift as core Kubernetes platform, these Paks can be installed on top of the container platform and enable additional tools, operators or interfaces. Licenses for running IBM software on OpenShift are covered through the Paks.

### **IBM Cloud Pak for Multicloud Management**

Helps to provide consistent visibility, automation, and governance across a range of hybrid, multicloud management capabilities such as event management, infrastructure management, application management, multicluster management, edge management and integration with existing tools and processes.

Improve cloud visibility, governance and automation →

### **IBM Cloud Pak for Integration**

Helps support the speed, flexibility, security and scale required for all of your integration and digital transformation initiatives and comes pre-integrated with a set of capabilities including API lifecycle, application and data integration, messaging and events, high speed transfer and integration security.

Integrate app, data, cloud services and APIs →

### **IBM Cloud Pak for Automation**

Helps you deploy on your choice of clouds anywhere Kubernetes is supported, with low-code tools for business users and real-time performance visibility for business managers. Customers can migrate their automation runtimes without application changes or data migration, and automate at scale without vendor lock-in.

Transform business process, decisions and content →

### **IBM Cloud Pak for Applications**

Helps to accelerate the build of cloud-native apps by leveraging built-in developer tools and processes, including support for microservices functions and serverless computing. Customers can quickly build apps on any cloud, while existing IBM middleware clients gain the most straightforward path to modernization.

Build, deploy and run applications →

### **IBM Cloud Pak for Data**

Helps to unify and simplify the collection, organization and analysis of data. Enterprises can turn data into insights through an integrated cloud-native architecture. IBM Cloud Pak for Data is extensible, easily customized to unique client data and AI landscapes through an integrated catalog of IBM, open source and third-party microservices add-ons.

Collect, organize and analyze data →

### **IBM Cloud Pak for Security**

IBM Cloud Pak for Security is a platform that helps you uncover hidden threats, make more informed risk-based decisions and prioritize your team's time. Connect to your existing data sources to generate deeper insights. Securely access IBM and third-party tools to search for threats across any cloud or on-premises location. Quickly orchestrate actions and responses to those threats.

Gain security insights, respond faster to threats →

At the time of writing, 2 Cloud Pak Systems (actual server racks) exist as well:

### **IBM Cloud Pak System**

An integrated system of hardware and software optimized to build, deploy and manage Cloud Paks and Kubernetes workloads in your data center. Configure in less than a day.

Five IBM Cloud Paks cover key workloads on your journey to the cloud. For rapid deployment, use the **IBM Cloud Pak System**

### **IBM Cloud Pak System for Data**

True plug-and-play for enterprise data and AI in hours, right out of the box

Accelerate your journey to AI to transform how your business operates with an open, extensible platform that runs on any cloud →

Depending on your needs, one or more of these Cloud Paks/Systems are sure to cover them. The Cloud Pak for Applications would be the most appropriate one for standard Java EE on Liberty.

DISCOVER IBM CLOUD PAKS

## **3.3 Buildpacks**

Unrelated to Cloud Paks, a buildpack is a technology that provides framework and runtime support for applications. Buildpacks typically inspect an application and determine which dependencies and which integrations are necessary to communicate with services.

Developer oriented PaaS solutions like **Cloud Foundry** have been offering buildpacks for quite some time and they have been accepted as an official cloud native technology since 2018

<https://buildpacks.io/>

I mention them because IBM uses Cloud Foundry inside IBM Cloud and **a buildpack to run WebSphere Liberty on Cloud Foundry** is generally available. Thanks to the integration with Cloud Foundry and buildpacks, simply "cf push" an application to IBM Cloud and watch it unfold, making it excellent at accelerating development. Infrastructure is abstracted in this model and while this may be a blessing during development, it could be a curse when more control or testing is needed from an infrastructure perspective.

## 3.4 Pipeline integrations

The concept of pipelines aims at creating an end-to-end software building and delivery system. Other aspect can easily be included in this same process: code review, security vulnerability checking, automated testing, automatically applying release models and much more.

The liberty-maven-plugin could help automate certain tasks before or after running a build. For example, the process might include deploying packaged code to a development server and running integration or smoke tests. The aforementioned plugin in combination with others readily available in most pipeline software should be able to help automate this.

## 3.5 Integration with existing software

On top of the usual integrations you may find within application servers such as datasource compatibilities, IBM has created several other features outside of the EE specification which can make life easier. Want to centralize your logging? Use the Logstash collector feature, part of the ELK stack (Elasticsearch – Logstash – Kibana) to send it to an existing cluster.

Most necessary features are readily available but you can also **extend or develop features** from the ground up.



# CHAPTER 4

-

# CLOSING NOTES



# 4. Closing notes

## 4.1 Notes on the future of Java and Liberty

Lots of interesting stuff on the horizon in the Java landscape. We are heading into an era with a new kind of EE specification, namely JakartaEE (which is named after the capital of Indonesia, same as Java was named after coffee which originated from Indonesia), which is now open source and will focus on cloud native development. Serving as a baseline, the JakartaEE Platform 8 has already been released and Platform 9 is under development.

The goal of the Jakarta EE 9 release is to deliver a set of specifications functionally like Jakarta EE 8 but in the new Jakarta EE 9 namespace jakarta.\*.

In addition, the Jakarta EE 9 release removes specifications from Jakarta EE 8 that were old, optional, or deprecated in order to reduce the surface area of the APIs to ensure that it is easier for new vendors to enter the ecosystem – as well as reduce the burden on implementation, migration, and maintenance of these old APIs.

My personal assumption is that in the Jakarta spec, we will also find a faster paced release model making the effort of moving to newer EE specifications smaller and less intrusive for development teams. I am also confident that the **OpenJDK (+ OpenJ9)** will become the new standard development kit instead of the long running Oracle JDK (+ hotspot) due to a plethora of reasons.

Despite or because of its decline in the list of top programming languages we are seeing many changes which may once again make, or enforce making, Java our weapon of choice.

With a continuous release model, cloud native support, Java EE and JakartaEE certifications, Liberty is a very good platform to use for our Java applications. Due to the stacked open source contributions with IBM enterprise support I think it can only grow faster and stronger.

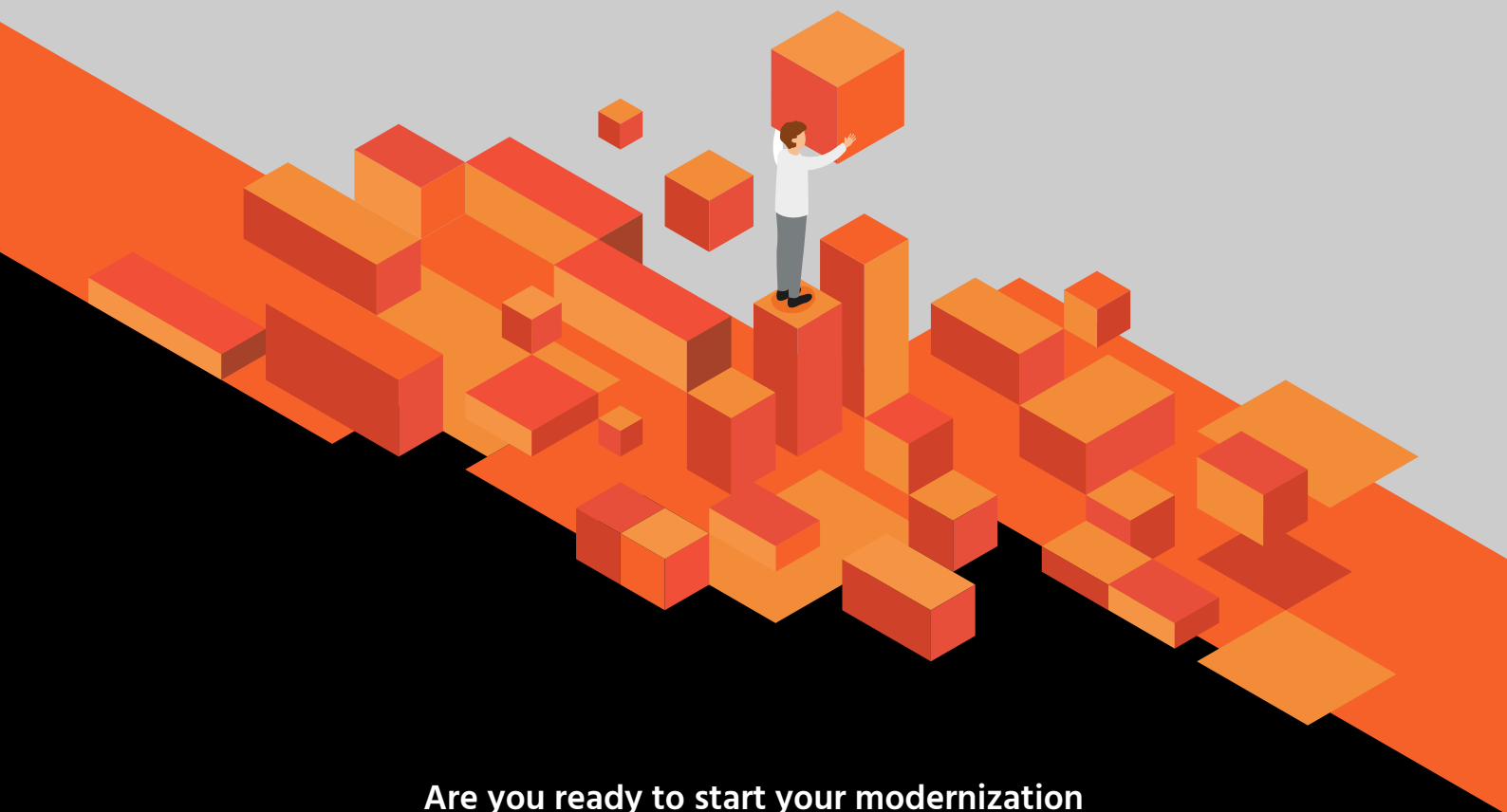
## 4.2 About the author

I am mostly hoping this whitepaper was interesting and of some use to you.

Unlike most IT professionals, I am not specialized in any specific field. Being an IT-generalist has helped me and customers I've worked for greatly in the past. My interests lie in many fields and while it is difficult to maintain knowing a bit about everything, I would not change it for the world.

I currently work as a DevOps engineer at **FlowFactor**, a company providing automation, DevOps services and Managed Services (24/7). FlowFactor is a **Cronos Group** subsidiary and an IBM platinum business partner.

-  
**Stefaan De Geyter**



Are you ready to start your modernization  
journey with FlowFactor?

LET'S TALK!

EXPLORE IBM CLOUD PAKS

**Telefoon** 32 (0) 496 57 41 98

**Email** [hello@flowfactor.be](mailto:hello@flowfactor.be)

**Website** [www.flowfactor.be](http://www.flowfactor.be)

**Adres** Veldkant 33a  
2550 Kontich