cubensquare
REDHAT ADVANCED BUSINESS PARTNER

Migration : WebSphere Liberty VM → OpenShift Container

WebSphere Liberty Core Parameters Overview

Parameter Type	File / Location	Example / Usage
JVM Options	`jvm.options`	`-Xms512m`, `-Xmx1024m`, `-Xgcpolicy:gencon`, `-verbose:gc`
Server Configuration	`server.xml`	`<feature>servlet-4.0</feature>`, `<httpEndpoint host="*" httpPort="9080"/>`
Logging	`server.xml`	`<logging traceSpecification="*=info" consoleLogLevel="AUDIT"/>`
Thread Pools	`server.xml`	`<executor coreThreads="10" maxThreads="50"/>`
Connection Pools	`server.xml`	`<connectionManager maxPoolSize="40" purgePolicy="FailingConnectionOnly"/>`
Data Source	`server.xml`	`<dataSource statementCacheSize="60" isolationLevel="TRANSACTION_READ_COMMITTED"/>`
Class Loading	`server.xml`	`<library id="sharedLib" filesetRef="sharedLibFiles"/>`
Application Deployment	`dropins/` or `<application>`	WAR/EAR files placed in `dropins/` or defined explicitly in XML
Monitoring	`server.xml`	`<feature>monitor-1.0</feature>`
Security	`server.xml`, `server.env`	`<keyStore id="defaultKeyStore" location="key.jks"/>`
CDI Optimization	`server.xml`	`<cdi12 enableImplicitBeanArchives="false"/>`
Update Triggers	`server.xml`	`<config updateTrigger="disabled"/>`

Category	VM-Based Liberty	OpenShift Containerized Liberty	Migration Considerations
Deployment Model	Manual install on OS	Container image via Liberty Operator	Build Docker image or use IBM-provided base image
Configuration Files	server.xml, jvm.options	Mounted ConfigMaps or baked into image	Externalize configs for portability
Class Loading	Shared classloader hierarchy	Isolated container classpath	Package dependencies inside container or use shared volumes
JNDI Resources	Defined in server.xml or via admin console	Defined in Kubernetes manifests or ConfigMaps	Map JNDI names to OpenShift secrets or environment variables
Secrets & Credentials	Stored in config files or OS keystores	Managed via Kubernetes Secrets or CSI drivers	Inventory all secrets and migrate to OpenShift-compatible formats

Category	VM-Based Liberty	OpenShift Containerized Liberty	Migration Considerations
Certificates	Java keystores (.jks) on VM	Mounted volumes or Kubernetes Secrets	Convert and mount keystores securely
Logging	File-based logs	stdout/stderr or sidecar logging	Use OpenShift logging stack (EFK or Loki)
Monitoring	OS-level tools or Liberty admin center	Prometheus, Grafana, Liberty Operator metrics	Integrate with OpenShift monitoring stack
Session Management	In-memory or WAS ND clustering	Stateless or Infinispan/JCache	Re-architect for stateless or use distributed cache
Java Version	Java 8/11 installed on host	Java bundled in container	Ensure compatibility with Liberty container base image
Persistence	Local disk or NFS	Persistent Volumes (PVCs)	Use OpenShift PVCs and StorageClasses
Networking	Static IPs, firewall rules	OpenShift Routes, Services	Define Services and Routes for internal/external access
Build & CI/CD	Manual deployment	Pipelines (Tekton, Jenkins, GitOps)	Containerize app and integrate with CI/CD
Scaling	Manual VM provisioning	Horizontal Pod Autoscaler (HPA)	Define resource limits and autoscaling policies
Licensing	Traditional WAS license	Liberty BYOL or Open Liberty (free)	Validate license terms for container deployment

---

Traditional vs Containerized Liberty Deployment

Traditional Liberty (VM)	Containerized Liberty (OpenShift)
Step 1: Create JVM with server.xml	**Create Liberty server config (`server.xml`) and embed it in Docker image or mount via ConfigMap**
Step 2: Copy JAR/WAR to dropins folder	**Add application to `/config/dropins` or `/config/apps` in Dockerfile**
Step 3: Restart JVM manually	**Container starts automatically with Liberty runtime; restart = redeploy pod or rollout update**

Containerization Steps (Liberty → OpenShift)

Step	Action	Tool/Location
1	Prepare `server.xml` with required features and endpoints	`/config/server.xml`

Step	Action	Tool/Location
2	Place WAR/JAR in `dropins/` or define `<application>` in `server.xml`	`/config/dropins` or `/config/apps`
3	Create a `Dockerfile` using IBM Liberty base image	`FROM icr.io/appcafe/websphere-liberty`
4	Copy config and app into image	`COPY server.xml /config/` `COPY app.war /config/dropins/`
5	Build image	`docker build -t my-liberty-app .`
6	Push image to registry	`docker push` or `oc image mirror`
7	Deploy to OpenShift using Deployment + Service + Route	YAML manifests or Liberty Operator
8	Monitor logs and metrics via OpenShift console or Prometheus	OpenShift UI / CLI

Key Concepts for Middleware Team

- **JVM = Container runtime**: No need to manually create JVMs; each container runs its own Liberty instance.
- **Dropins = Image layer**: WAR files are baked into the image or mounted at runtime.
- **Restart = Pod lifecycle**: Restarting Liberty = restarting the container or rolling out a new deployment.
- **server.xml = portable config**: Can be version-controlled and reused across environments.

Traditional Liberty JVM Setup (on VM)

```
# Step 1: Create Liberty server
/opt/ibm/wlp/bin/server create myAppServer

# Step 2: Configure server.xml
vi /opt/ibm/wlp/usr/servers/myAppServer/server.xml
```

Sample `server.xml`:

```
<server description="Traditional Liberty JVM">
  <featureManager>
    <feature>servlet-4.0</feature>
    <feature>jsp-2.3</feature>
  </featureManager>

  <httpEndpoint host="*" httpPort="9080" httpsPort="9443" />

  <webApplication location="myApp.war" contextRoot="/myApp"/>
</server>
# Step 3: Deploy application
```

```
cp myApp.war /opt/ibm/wlp/usr/servers/myAppServer/dropins/

# Step 4: Start JVM
/opt/ibm/wlp/bin/server start myAppServer
```

Containerized Liberty Setup (Dockerfile)

```
# Step 1: Use Liberty base image
FROM icr.io/appcafe/websphere-liberty:kernel-java17-openj9-ubi

# Step 2: Add server.xml
COPY --chown=1001:0 server.xml /config/

# Step 3: Add WAR file
COPY --chown=1001:0 myApp.war /config/dropins/

# Step 4: Install required features
RUN configure.sh
```

Same `server.xml` reused:

```
<server description="Containerized Liberty JVM">
  <featureManager>
    <feature>servlet-4.0</feature>
    <feature>jsp-2.3</feature>
  </featureManager>

  <httpEndpoint host="*" httpPort="9080" httpsPort="9443" />

  <webApplication location="myApp.war" contextRoot="/myApp"/>
</server>
# Step 5: Build and run container
docker build -t liberty-app .
docker run -d -p 8080:9080 liberty-app
```

Points to be noted by Middleware Team

Concept	Traditional VM	Containerized
JVM Instance	Created manually	Defined by container runtime
server.xml	Edited in filesystem	Baked into image or mounted via ConfigMap
WAR Deployment	Copied to dropins/	Added via Dockerfile COPY
Restart	Manual server restart	Container restart or rollout
Scaling	Manual provisioning	Automated via OpenShift HPA

Reusing Liberty Base Image for Multiple Deployments

Step 1: Build a base Liberty image

This image contains:

- Liberty runtime
- `server.xml` with required features
- Any shared libraries or configs

```
# liberty-base.Dockerfile
FROM icr.io/appcafe/websphere-liberty:kernel-java17-openj9-ubi
COPY --chown=1001:0 server.xml /config/
RUN configure.sh
```

Build it:

```
docker build -t liberty-base -f liberty-base.Dockerfile .
```

Step 2: Create app-specific image using base

Now for each application, you just COPY the JAR/WAR:

```
# liberty-app.Dockerfile
FROM liberty-base
COPY --chown=1001:0 myApp.jar /config/dropins/
```

Build it:

```
docker build -t liberty-myapp .
```

Repeat this for each app — just change the JAR file.
