Liferay DEVCON

Dave Nebinger

Liferay 7.4: The Developer Experience

Embracing the new while retaining the old

Liferay® DEVCON

Liferay 7.4: The Developer Experience

Embracing the New While Retaining the Old

The Developer Experience

There are now two ways to develop for Liferay:

- Classic Developer Experience - The way we have developed for Liferay since 7.0 using OSGi and jars and wars.
- CX Developer Experience - Client Extensions (CX) are the new way to develop using external customizations and apps compatible with LXC, LXC-SM and on-prem solutions.

The Classic Developer Experience

The classic ways of developing for Liferay are still supported:

- OSGi Module Development
- JSP Fragment Bundles
- Portlet WARs (Liferay MVC, Spring, and JSF)
- JS-based Portlets (React, Angular, Vue.js, etc)

Note that these are options for self-hosting or LXC-SM, but are not available for LXC.

Classic Pros vs Cons

PROS

Same process we've been using since 7.0, so it is well known.

Access to full Liferay API.

Easy to diagnose deployment issues.

CONS

May be impacted by Liferay API updates.

Customizations should be verified against new release.

Buggy customizations affect the portal.

Rework necessary for Liferay upgrades.

The CX Developer Experience

Covered a lot during DEVCON:

- Objects for no-code/low-code custom data persistence.
- Client Extensions to extend Liferay without modifying Liferay.
- Custom fragments for custom UI building.
- JS-based applications (React, Angular, Vue.js, etc).
- Custom Docker-based custom services (i.e. Spring Boot, Node.js, etc).

Note that these are options for self-hosted, LXC-SM *and* LXC.

CX Pros vs Cons

PROS

Uses any technology you want.

Isolated from Liferay updates and upgrades.

Works in On-Prem, LXC-SM and LXC environments.

Buggy CX will not break Liferay functionality.

CONS

Headless and JSONWS API access only.

Limited ability to customize Liferay.

Extra monitoring and management.

One Artifact, Three Deployments

Self-Hosted

Drop the .zip file into the osgi/client-extensions folder.

Any Docker-based services (i.e. Spring Boot) must be started manually.

LXC-SM

The .zip file is added to the deployment image for LXC-SM by putting it in the osgi/client-extensions folder.

LXC

Use the lcp tool to upload the .zip file to the target instance.

Three Primary Files

client-extension.yaml

Defines the client extension for Liferay.

LCP.json

Defines the client extension for LXC.

Dockerfile

Defines the Docker container to build for the extension.

Source of Truth

https://github.com/liferay/liferay-portal/tree/master/workspaces/liferay-sample-workspace

Liferay Docker Bases

Find the list here: https://github.com/liferay/liferay-docker/tree/master/templates

Primary containers are:

- liferay/caddy - For serving static resources like CSS, JS, etc.
- liferay/jar-runner - For serving java-based solutions like Spring Boot.
- liferay/node-runner - For serving Node.js based solutions.
- liferay/batch - For loading batch data.

Example Time

Final Thoughts

- CX is very much a work in progress.
- Classic OSGi module development is still a valid path for on-prem and LXC-SM solutions, although preferring CX has advantages.
- Questions, Comments, Other? - Post to Ask or Liferay Community Slack.
- “Missing” Client Extension? - Open a support ticket or feature request on issues.liferay.com.

Questions?

How was this session?

Please share your rating in
the event app. Thank you!