Liberty InstantOn start-up for cloud native Java applications

What you'll learn

You will learn how you can start up your cloud-native applications in milliseconds without compromising on throughput, memory, development-production parity, or Java language features using InstantOn.

What is a Checkpoint?

This checkpoint is a snapshot of the running application process that can be persisted and then quickly restored to bring the application process back into the state it was in when the checkpoint was taken. This process enables the Liberty instance, along with any configured application to be restored multiple times into distinct instances of your application.

How Open Liberty enables InstantOn start-up?

To enable InstantOn, Open Liberty uses new features of the OpenJ9 JVM and a Linux technology called Checkpoint/Restore In Userspace (CRIU) to take a checkpoint of the application process as it starts. This checkpoint is a snapshot of the running application process that can be persisted and then quickly restored to bring the application process back into the state it was in when the checkpoint was taken. This process enables the Liberty instance, along with any configured applications, to be restored multiple times into distinct instances of your application.

Getting started

To make it quicker and easier for you to create an application that uses InstantOn firstly you will have to set up a working example. For the purpose of this demo a working example has already been setup for you within the `/home/sysadmin/Desktop/guide-getting-started/finish`. This means that you will not have to do any code changes to enable InstantOn. If you are welcome to check out the `Dockerfile` which is the only thing that has been changed to pull in the InstantOn container image.

You should firstly open the terminal which you can do by pressing the icon on the desktop. It looks like the icon in the following screenshot:

Now that you have the terminal open you should login as root user and opening the `guide-getting-started/finish` directory. Do this by entering the following command into the open terminal.

```
sudo -i
cd ../home/sysadmin/Desktop/guide-getting-started/finish
```

Try out the application

To try out the application before enabling InstantOn, you can run the following Maven goal to build the application and deploy it to Open Liberty.

```
mvn liberty:run
```

After you see the following message in the terminal, your server is ready:

```
[INFO] [AUDIT   ] CWWKF0011I: The defaultServer server is ready to run
a smarter planet. The defaultServer server started in 6.543 seconds.
```

Check out the service by visiting http://localhost:9080/dev/system/properties.

After you finish checking out the application, stop it by pressing **CTRL+C** in the terminal where you ran the server, or by typing **q** and then pressing the **enter/return** key.

Enabling InstantOn

Firstly, you need to build the WAR for the application. To do this run:

```
mvn package
```

This command builds a `target/guide-getting-started.war` archive. We can now include this WAR in a container image that uses the InstantOn feature.

Which you can find by running the following `ls` command:

```
ls /home/project/guide-getting-started/finish/target
```

Building the application container image

For CRIU to be able to take a checkpoint of and restore a process, the CRIU binary must be granted additional Linux capabilities. Specifically for Open Liberty, it needs to be granted `cap_checkpoint_restore`, `cap_net_admin` and `cap_sys_ptrace`. The Open Liberty InstantOn beta image includes the CRIU binary with the necessary capabilities already granted to the CRIU binary file. For the CRIU binary to be given access to its assigned capabilities at run time, the container that is running CRIU must also be granted the necessary capabilities when it is launched. You can grant these capabilities to the container in one of two ways:

1. Use a privileged container using the `-privileged` option

2. Assign specific capabilities using `--cap-add` options

When you use podman, the daemon typically has root authority. This authority allows it to grant any requested capabilities when it launches a container.

With that understanding, we can now build the container image by using the `podman build` command. From the `finish/` directory, run the following command to build the container image for the application:

```
podman build -t getting-started .
```

This command creates the `getting-started` container image. However, this container image does not contain any checkpoint image files that can be used for InstantOn startup. You can run this application container image with the following command:

```
podman run \
  --name getting-started \
  --rm -p 9080:9080 getting-started
```

Wait for the following message, which indicates that the server startup is complete:

```
[INFO] [AUDIT   ] CWWKF0011I: The defaultServer server is ready to run
a smarter planet. The defaultServer server started in 6.543 seconds.
```

Note that the amount of time Liberty takes to report it has been started and check out the service running in the container at the `/dev/system/properties` endpoint by clicking the visiting the http://localhost:9080/dev/system/properties.

After you finish checking out the application, stop it by pressing **CTRL+C** in the terminal where you ran the server, or by typing **q** and then pressing the **enter/return** key.

Checkpoint an application in-container

Open Liberty has three phases during the startup process where a checkpoint can occur:

1. ***features*** - This is the earliest phase where a checkpoint can happen. The checkpoint occurs after all of the configured Open Liberty features are started, but before any processing occurs for the installed applications.

2. ***deployment*** - The checkpoint happens after processing the configured application metadata. If the application has any components that get run as part of the application starting, the checkpoint is taken before executing any code from the application.

3. ***applications*** - This is the last phase where a checkpoint can happen, so it has the potential to provide the fastest startup time when restoring the application instance. The checkpoint happens after all configured applications are reported as started. This phase happens before opening any ports for listening to incoming requests for the applications.

The ***applications*** phase typically provides the quickest startup time for an application, but it also might cause some application code to run before the server process checkpoint happens. This might lead to undesired behavior when restoring the checkpoint process if the application holds on to some state that should not be restored into more than one concurrent instance of the application. For example, connecting to an outside resource such as a database before the checkpoint is taken results in a failure to restore many instances of such a process since this would try to restore the same connection multiple times. However, If your application initialization does not perform operations such as opening database connections, you might be able to use the ***applications*** phase for the checkpoint.

Of these three options, the ***features*** phase offers the least value in terms of improving an application startup time. Scenarios where an application deployment would want to use ***features*** over deployment are very limited and are not good use cases for Liberty InstantOn. For the 23.0.0.2-beta, the ***features*** checkpoint phase has been removed. This change leaves only two options to choose from for the checkpoint: ***deployment*** and ***applications***.

Now that the application is built you can create a checkpoint for the application. Use the **applications** phase by running the following podman command:

Perform a checkpoint in container

```
podman run \
  --name getting-started-checkpoint-container \
  --privileged \
  --env WLP_CHECKPOINT=applications \
  getting-started
```

- The `--privileged` option is required to perform the CRIU checkpoint in-container.
- The `WLP_CHECKPOINT` environment variable is used to specify the checkpoint phase. The ***applications*** checkpoint phase will provide the fastest restore time.

This will start the container with the application running on Open Liberty. After Open Liberty starts, it performs the checkpoint at the phase specified by the `WLP_CHECKPOINT` environment variable. After the container process data has been persisted, the container will stop, leaving you with a stopped container that contains the checkpoint process data.

The output will look something like this:

```
Performing checkpoint --at=applications

Launching defaultServer (Open Liberty 22.0.0.11-beta/wlp-1.0.69.cl221020220912-
1100) on Eclipse OpenJ9 VM, version 17.0.5-ea+2 (en_US)
CWWKE0953W: This version of Open Liberty is an unsupported early release version.
[AUDIT   ] CWWKE0001I: The server defaultServer has been launched.
[AUDIT   ] CWWKG0093A: Processing configuration drop-ins resource:
/opt/ol/wlp/usr/servers/defaultServer/configDropins/defaults/checkpoint.xml
[AUDIT   ] CWWKG0093A: Processing configuration drop-ins resource:
/opt/ol/wlp/usr/servers/defaultServer/configDropins/defaults/keystore.xml
[AUDIT   ] CWWKG0093A: Processing configuration drop-ins resource:
/opt/ol/wlp/usr/servers/defaultServer/configDropins/defaults/open-default-
port.xml
[AUDIT   ] CWWKZ0058I: Monitoring dropins for applications.
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://f5edff273d9c:9080/ibm/api/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://f5edff273d9c:9080/metrics/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://f5edff273d9c:9080/health/
[AUDIT   ] CWWKT0016I: Web application available (default_host):
http://f5edff273d9c:9080/dev/
[AUDIT   ] CWWKZ0001I: Application guide-getting-started started in 0.986
seconds.
[AUDIT   ] CWWKC0451I: A server checkpoint was requested. When the checkpoint
completes, the server stops.
```

Creating the application checkpoint image

So far, you have created the checkpoint process data for the getting-started application and stored it in a stopped container named `getting-started-checkpoint-container`. The final step is to create a new container image that contains the checkpoint process data. When this container image is started, it will resume the application process right from the point that the checkpoint was created, resulting in an InstantOn application. You can create the new image by running the following `podman commit` operation:

```
podman commit \
  getting-started-checkpoint-container \
  getting-started-instanton
```

Now you have two application images named `getting-started` and `getting-started-instanton`. Starting a container with the `getting-started-instanton` container image will show a much faster startup time than the original `getting-started` image.

Running the instanton application image

Running with --privileged option

You can start the application from the container image with the following command. This command uses `--privileged` flag to allow CRIU to be able to restore the process in-container.

```
podman run \
  --rm --privileged -p 9080:9080 \
  getting-started-instanton
```

If successful you should see output like this:

```
[AUDIT   ] CWWKZ0001I: Application guide-getting-started started in 0.059
seconds.
[AUDIT   ] CWWKC0452I: The Liberty server process resumed operation from a
checkpoint in 0.088 seconds.
[AUDIT   ] CWWKF0012I: The server installed the following features: [cdi-3.0,
checkpoint-1.0, concurrent-2.0, distributedMap-1.0, jndi-1.0, json-1.0, jsonb-
2.0, jsonp-2.0, monitor-1.0, mpConfig-3.0, mpHealth-4.0, mpMetrics-4.0,
restfulWS-3.0, restfulWSClient-3.0, servlet-5.0, ssl-1.0, transportSecurity-1.0].
[AUDIT   ] CWWKF0011I: The defaultServer server is ready to run a smarter planet.
The defaultServer server started in 0.098 seconds.
```

Notice that the server starting time is reduced significantly.

After you finish checking out the application, stop it by pressing **CTRL+C** in the terminal where you ran the server, or by typing **q** and then pressing the **enter/return** key.

Running with an unprivileged container

Running fully privileged containers is not recommended. The best practice is to instead reduce the elevated privileges down to only what is required to run the container. You can use the following command to grant the container the necessary privileges without running a fully privileged container:

`podman run` with unconfined `--security-opt` options

```
podman run \
  --rm \
  --cap-add=CHECKPOINT_RESTORE \
  --cap-add=NET_ADMIN \
  --cap-add=SYS_PTRACE \
  --security-opt seccomp=unconfined \
  --security-opt systempaths=unconfined \
  --security-opt apparmor=unconfined \
  -p 9080:9080 \
  getting-started-instanton
```

The `--cap-add` options grant the container the three Linux capabilities that CRIU requires. The `--security-opt` options are necessary to grant CRIU access to the required system calls and access to `/proc/sys/kernel/ns_last_pid` from the host.

After you finish checking out the application, stop it by pressing **CTRL+C** in the terminal where you ran the server, or by typing **q** and then pressing the **enter/return** key.

Running with an unprivileged container with confined security

You can further simplify the checkpoint process by reducing the need for the `--security-opt` options that use `unconfined`. By default, container does not grant access to all the system calls that CRIU needs (defaults specified in the file *`/usr/share/containers/seccomp.json`*). First, you need an additional configuration file that grants all the required system calls that CRIU needs to the container. Second, the host `/proc/sys/kernel/ns_last_pid` needs to be mounted. You can do both these steps with the following command:

```
podman run \
  --rm \
  --cap-add=CHECKPOINT_RESTORE \
  --cap-add=NET_ADMIN \
  --cap-add=SYS_PTRACE \
  --security-opt seccomp=criuRequiredSysCalls.json \
  -v /proc/sys/kernel/ns_last_pid:/proc/sys/kernel/ns_last_pid \
  -p 9080:9080 \
  getting-started-instanton
```

The `--security-opt seccomp=...` option refers to a file called `criuRequiredSysCalls.json`. This file specifies the system calls required by CRIU. The `-v` option mounts the host `/proc/sys/kernel/ns_last_pid` for access by the container.

Once you are finished checking out the application, press **CTRL+C** to stop the container in which Liberty is running.

The `criuRequiredSysCalls.json` can be found at the bottom of this blog post.

Summary

Nice Work!

You just learned how to use the Liberty InstantOn beta feature to run an application in container.

What did you think of this lab?

We want to hear from you. To provide feedback, click the following link.

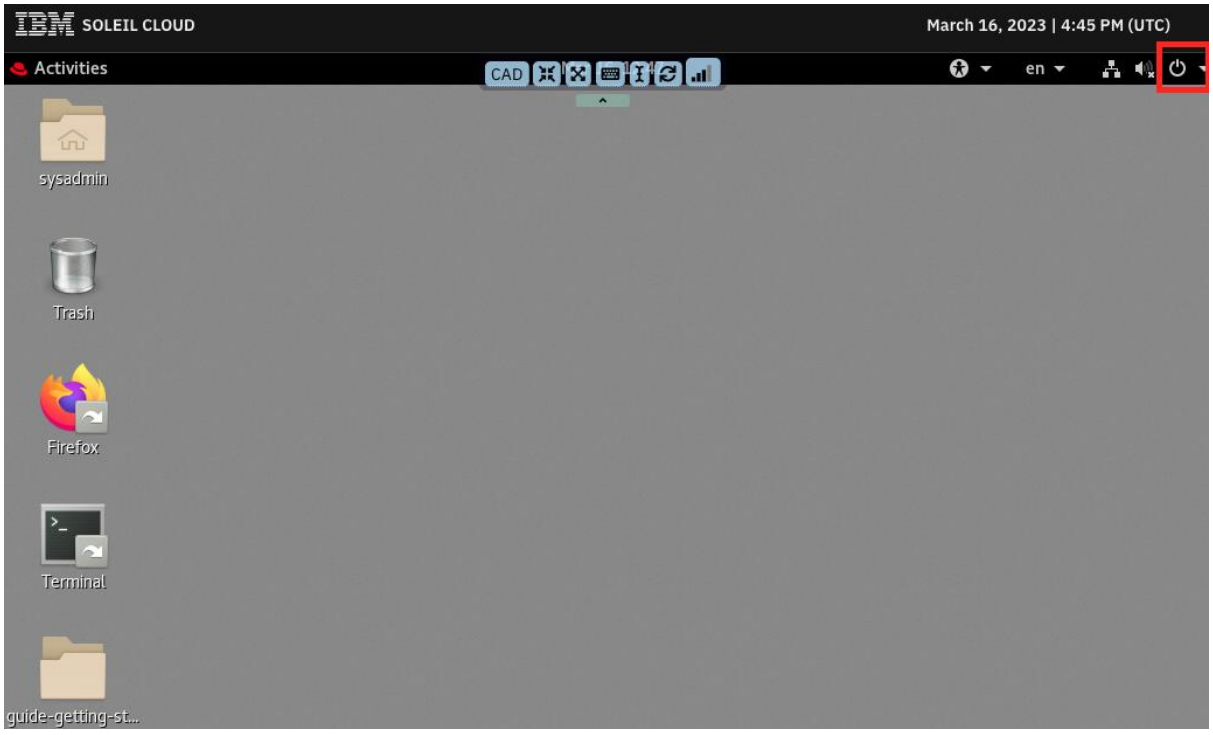
- Give us feedback by opening a new issue in the open-liberty repository on GitHub

CLEAN UP THE ENVIRONMENT

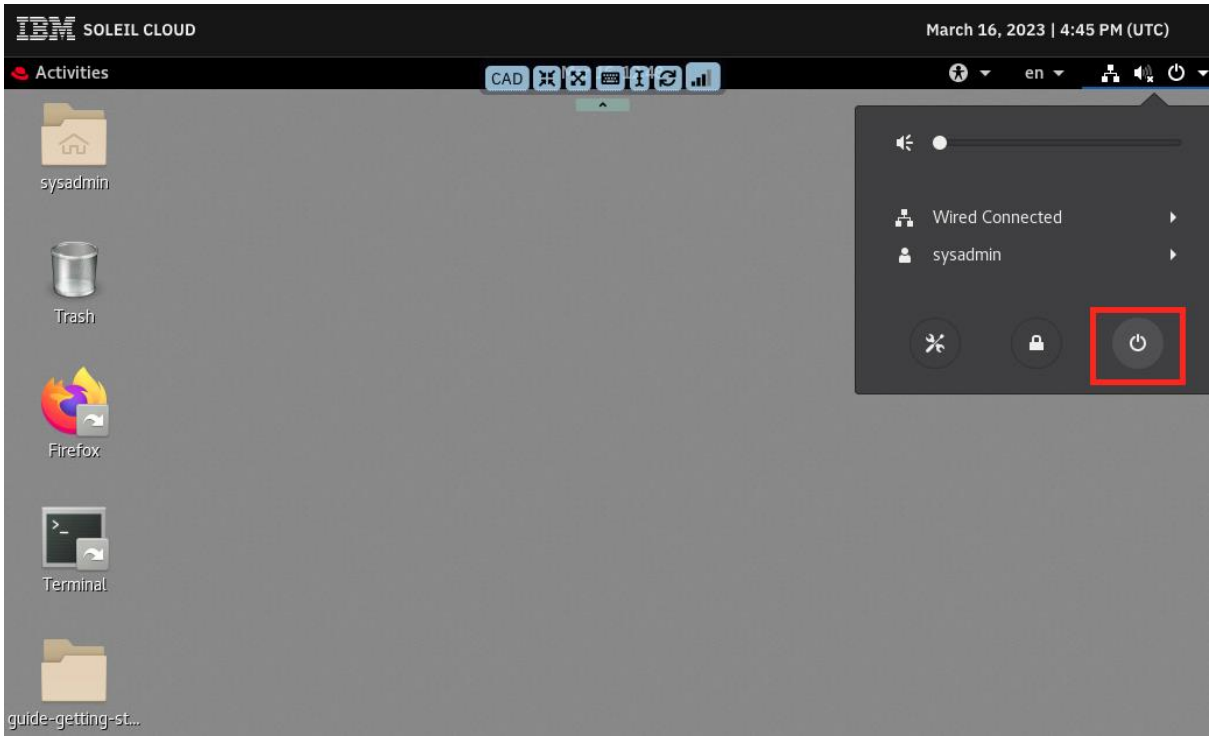
Run the following command to delete all the podman images and containers:

```
podman system prune --all --force && podman rmi --all
```

In the top right corner press the power button.



Click the power button again in the drop down.



Click the “Power off” button and close the window.