We had an old code base which runs on a tablet for the report generation. The task is to re-use this for similar tasks in a cloud environment. One can easily spin an EC2 instance and get the things running. However, we decided to choose serverless setup, i.e., lambda for generating individual widgets. More details as follows -

1. The existing code base uses JavaFx for graphic rendering.
2. Report doesn't need any user interaction
3. The final report is just a pdf, which should be a good candidate for lambda considering no user interaction.
4. Historically, this code always was used on Windows tablets.

The very first problem that we encountered was, lambda logs giving error saying JavaFx libraries are missing or cannot find it. How, could that be? We have all the libs in final jar. Looking closer, these libraries are all .dll files, i.e. windows libs. Why? Since the project is compiled on windows box. However, lambda is nothing but linux containers or a close cousin would be docker container for amazon linux.

This led us to try compiling these javaFx projects on linux. I used ubuntu 20.04 and could see the same set of libraries being available as .so files. Now, the javaFx libraries missing error is gone. However, it still complains about few other libraries not being found.
Any guess ?
JavaFx libraries depend on OS native libraries and lambda is a super minimal subset of actual linux. You can not find these libraries listed somewhere, since JavaFx community wouldn't know either.
Only option is to install various packages and see if things work. Where are we going to install these packages? Write a code in java to invoke package install commands ?
E.g. yum install libc
Or a few more things like that.
However, lambda doesn't have access to the public internet, which means you can't invoke these commands.
Ever heard of AWS toolkit? Install it as a plugin on for your intellij or eclipse setup.
Make sure the following two commands work from inside intellij terminal.

```
$ sam --version
$ docker ps
```

You would find loads of online documentation to figure this out.

Create a simple docker instance for amazon linux 2. Which is something similar to centOS or RHEL, i.e., default package manager is "yum"

Pick up any javafx library that was missing and find it's OS native dependencies.
I was required to get following javaFx libraries -
libavplugin-54.so libavplugin-56.so libavplugin-57.so libavplugin-ffmpeg-56.so libavplugin-ffmpeg-57.so libfxplugins.so libgstreamer-lite.so libjfxmedia.so libjfxwebkit.so

These libraries will be available once you compile for linux x86-64

To find dependencies individual libraries use the following command on linux build machine.

$ ldd libfxplugins.so

linux-vdso.so.1 (0x00007fffb03f8000)
libgstreamer-lite.so => not found
libgobject-2.0.so.0 => /lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0x00007f9a94869000)
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f9a94740000)
***libc.so.6** => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9a9454e000)*
libffi.so.7 => /lib/x86_64-linux-gnu/libffi.so.7 (0x00007f9a94542000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f9a9451f000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f9a944aa000)

$ whatprovides *libc.so.6*
<this gives list of packages which have this library.>

$ yum install <package found in above list>
$ rpm -ql <package name > <= this gives location all files being installed as a part of this package.

Now you can hand pick the required library and copy it to your jar.

By default the jar is extracted @ /var/task inside the lambda environment.
Embed the libraries in your jar so that they will all be extracted under this location. This way you don't have to worry about the library paths to be manipulated, which may not work.

With these changes your library related errors in lambda are gone. It still doesn't work and fails with an error saying "JavaFx toolkit not found". This can be easily fixed by calling "new JFXPanel()" as your first line of code. The only problem is this works when you have an actual monitor or a display device. Lambda is a headless setup and we would never have such devices there. The javafx monocle library comes to your rescue here. More details here - https://wiki.openjdk.java.net/display/OpenJFX/Monocle
You need to download monocle jar and load it inside your lambda before any other code execute. Possible place is a static block of your class.

```
public static void loadMonocle(){
    System.out.println("Setting monocle command line options.");
    System.setProperty("glass.platform", "Monocle");
    System.setProperty("monocle.platform", "Headless");
    System.setProperty("prism.order", "sw");

    System.out.println("Loading monocle.");
    try {
        File AGENT_JAR = new File("/var/task/openjfx-monocle-jdk.jar");
        Method addUrl = URLClassLoader.class.getDeclaredMethod("addURL", URL.class);
        addUrl.setAccessible(true);
        addUrl.invoke(PlatformFactory.class.getClassLoader(), AGENT_JAR.toURI().toURL());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

This will load your monocle and lambda setup will think it has required devices. However, you need to load the javaFx libraries manually before doing this. As long as javaFx and corresponding native libs are present in /var/task you are safe. Just load those using absolute paths.

```
public static void loadlibs(){
    loadLib("/var/task/libX11.so.6");
    loadLib("/var/task/libGL.so.1");
    loadLib("/var/task/libXrender.so.1");
    loadLib("/var/task/libfreetype.so.6");
    //        loadLib("/var/task/libfontconfig.so.1");
}

private static void loadLib(String lib){
    try {
        System.load(lib);
        System.out.println("done loading : " + lib);
    }catch (Exception e){
        System.out.println("Failed loading " + lib);
        e.printStackTrace();
    }
}
```

You are yet to solve one more issue with JavaFx setup, i.e., fonts. Lambda being minimal doesn't have any fonts installed and one can not install them for the similar reasons discussed earlier.
Again docker container is at your rescue. I have installed font dejavu and embedded all required data inside final jar. This gets extracted to folder /var/task/font. Update your /var/task/fonts.conf as follows -

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
  <dir>/var/task/fonts/</dir>
  <cachedir>/tmp/fonts-cache/</cachedir>
  <config></config>
</fontconfig>
```

Make sure you set the following variable for your lamda.
FONTCONFIG_FILE = /var/task/fonts.conf

This will create your basic infrastructure to start using JavaFx code. Which worked in my case and hopefully for everybody.

One last pesky issue I faced was the javaFx widgets are rendered in background and the report is just a snapshot of these widgets after they are built completely. One has to wait for random time for widgets to be rendered. If this is a last command in your setup then lamda would silently terminate ignoring the fact that render is yet to finish. I have used countdown latch to wait for rendering to finish and then take a snapshot. However, this is tricky and one has to use PauseTransition instead of regular thread sleep. I haven't investigate why this is so, however it works for my case.