WIND RIVER

Education Services

Application Stacks Developing Quick Start Guide for Intel Gateway Solutions

Agenda

Application Stacks

- OpenJDK
- Lua/MQTT
- Python
- SQLite
- OSGi
- C

Objectives

By the end of this chapter you will be able to:

- Configure OpenJDK into your target
- Configure MQTT and Lua into your target
- Identify why you would use an SQLite3 database in your target
- Identify the advantages of using OSGi in your target

Agenda

Application Stacks

- **OpenJDK**
- Lua/MQTT
- Python
- SQLite
- OSGi
- C

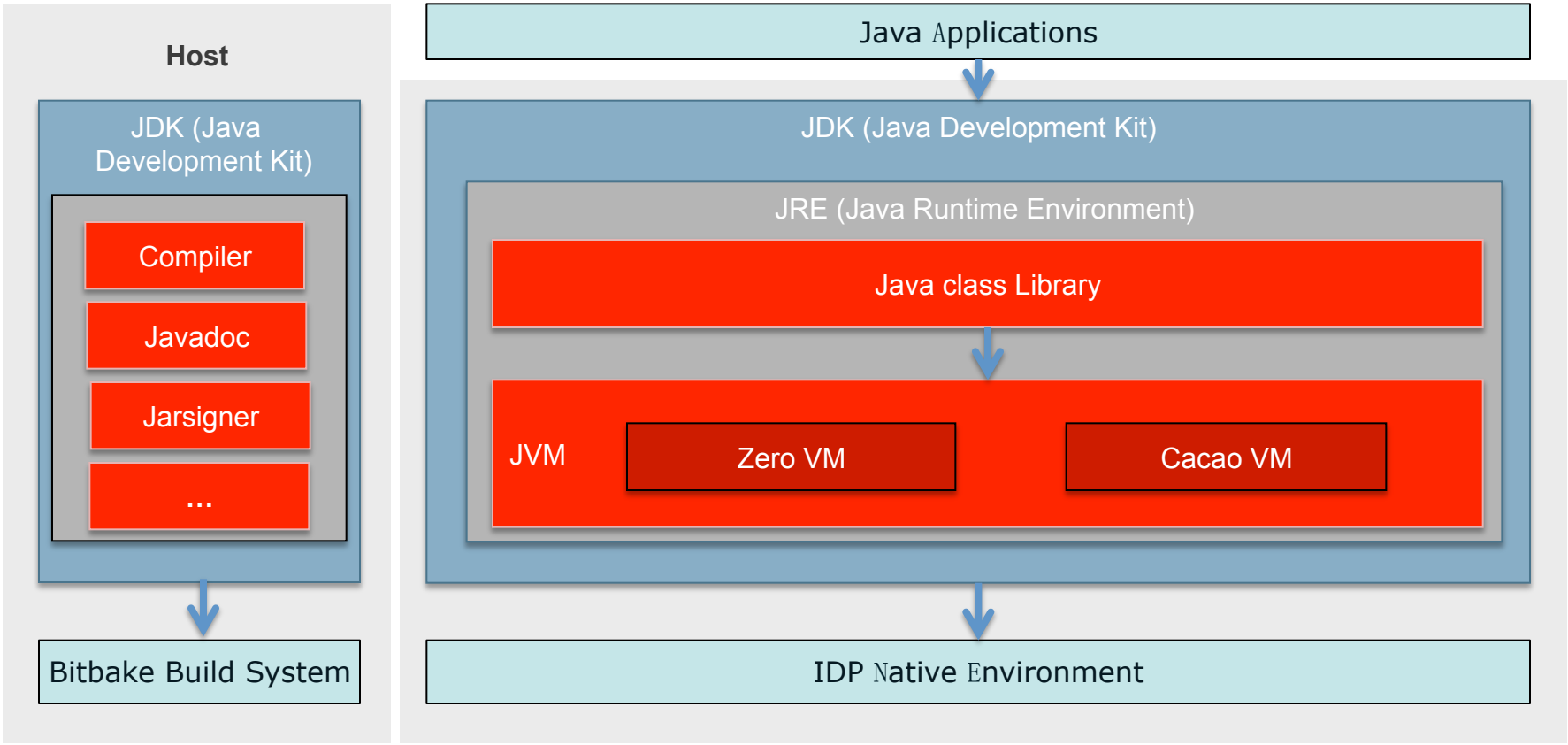
OpenJDK

- Open source implementation of Java SE 7
- IDP provides run-time environments:
  - Java Runtime Environment (JRE) 1.6.0_27
  - Cacao (1.6.0+r68fe50ac34ec)
    - Open source Java virtual machine
    - Includes JIT capability
- To include this in your target image, configure it with:

  --enable-addons=wr-idp is required

  --with-template= feature/openjdk-bin
  - Automatically included when **--enable-rootfs= glibc-idp**

OpenJDK



A full, free, open source edition of Java Standard Edition (SE), Java Virtual Machine (JVM) implementation:

- Supports Java SE versions 6 and 7, leverages system provided libraries (zlib, libpng, ...)
- Choice of JVM – OpenJDK, Zero VM, Cacao VM
- For a full list of features supported by openjdk, please refer to: http://openjdk.java.net/

Using OpenJDK

- Build on your host and download to the target.
- Build on your host and include in an image.
  - Project configuration needs to include a few things:
  - Include **--enable-internet-download=yes**.
  - Your project **local.conf** file needs **REBUILD_OPENJDK = "yes"**.
  - The process takes quite some time.
  - Create a layer to contain your Java project code.
  - Ensure that the **template.conf** file has **IMAGE_INSTALL_append += "*myjavaprj*"**.

OpenJDK – Hello World

Create a file HelloWorld.java with the following contents:

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello, World");
        }
}
```

Compile HelloWorld.java into a HelloWorld class file using the Java complier javac

```
$ javac HelloWorld.java
```

Transfer the HelloWorld.class file to the IDP target

```
$ scp HelloWorld.class root@<target-host-name-or-ip>
```

On the target, execute the HelloWorld program as follows

```
root@WR-IntelligentDevice:~# java HelloWorld
Hello, World
root@WR-IntelligentDevice:~#
```

Agenda

Application Stacks

- OpenJDK
- **Lua/MQTT**
- Python
- SQLite
- OSGi
- C

Lua

- A scripting language that grew out of programs developed for the specialized data entry requirements of petrochemical simulations.
- Created in 1993, first released to the outside in 1996.
- Wind River IDP uses version 5.1.5 by default, though a 5.2 version is also provided.
- Common uses of Lua:
  - a configuration language for applications
  - a standalone scripting language
  - an embedded language in applications to modify run-time behavior
  - complete language fits into 180kB, can go as low as 80k depending on features required.

Lua – Examples

The classic hello world program can be written as follows:

```
print('Hello World!')
```

The factorial function:

```
function factorial(n)
        if n == 0 then return 1 end
        return n * factorial(n - 1)
end
```

Loops:

```
while condition do --statements end
repeat statements until condition
for i = first,last,delta do print(i) end
for key, value in pairs(_G) do print(key, value) end
```

MQTT

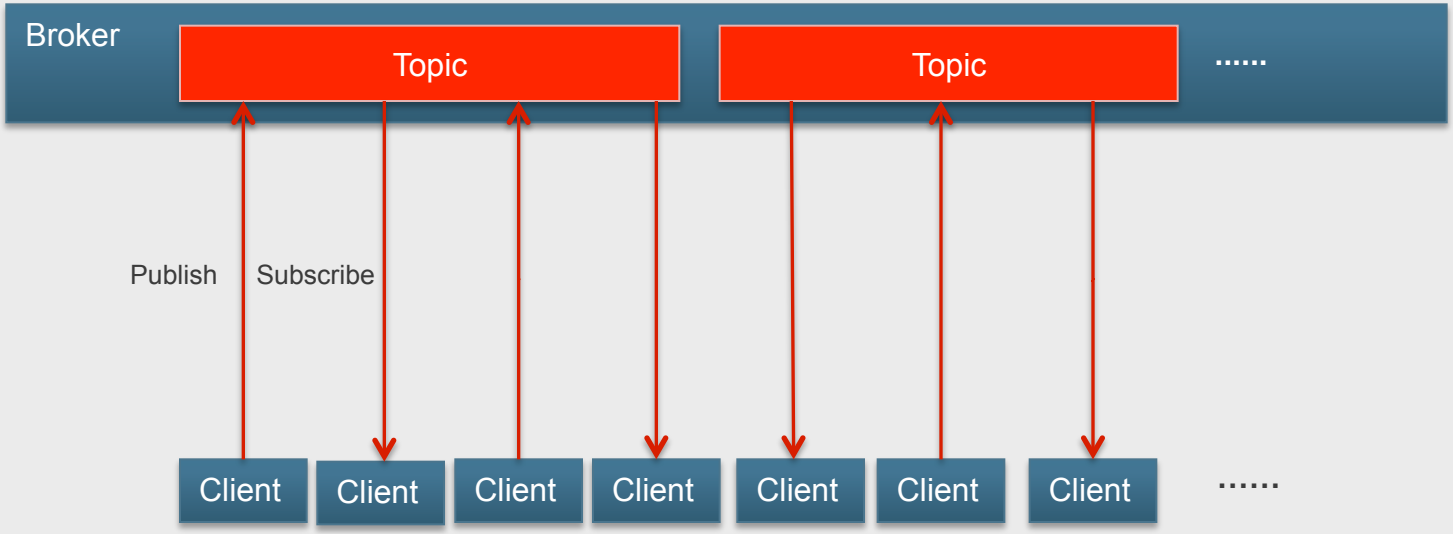
- MQTT = Message Queue Telemetry Transport
- A lightweight (low power, low network bandwidth) publish -and-subscribe messaging protocol for M2M IoT
- Designed for:
  - constrained devices and
  - low bandwidth, or high latency, or unreliable networks
- TCP/IP port 1883 is reserved with IANA for use with MQTT. TCP/IP port 8883 is also registered, for using MQTT over SSL.

MQTT & IDP

- Placed into the image by default with **--enable-rootfs=glibc-idp**.
- Alternatively you need

  --enable-addons=wr-idp

  --with-template=feature/mqtt



WIND RIVER

MQTT & IDP

- MQTT offered by IDP:
  - paho.mqtt.lua: a client-side implementation based on Lua for version 3.1 of the MQTT protocol
  - command-line utilities for publishing and subscribing to MQTT topics
  - mosquitto: server version 3.1 of the MQTT protocol
- A Mosquitto server starts at boot time.
  - version 1.1.3
  - MQTT 3.1 broker
- Includes example programs by default.

  /root/examples/mqtt-client/*
- For more information, go to **http://mosquitto.org**.

MQTT – Example

Statistics about RX&TX packets from a number of devices in a network need to be collected. The number of packets received needs to be sent to two different locations.

Set up two separate topics to capture the data from the devices:

1. Network/packets/sent
2. Network/packets/received

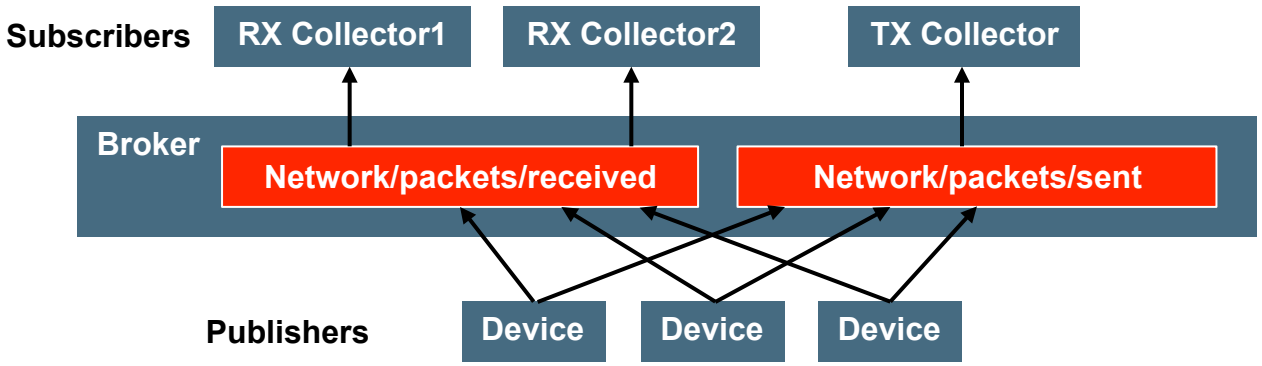
Three subscribers will be set up (two for RX, one for TX) to retrieve the data.

Subscriber Side

```
$ mosquitto_sub –h idp -t network/packets/sent
```

Publisher Side:

```
$ mosquitto_pub –h idp -t network/packets/sent -m “$HOSTNAME: 5“
```



Agenda

Application Stacks

- OpenJDK
- Lua/MQTT
- **Python**
- SQLite
- OSGi
- C

Python

- Open source implementation of Python 2.7
- To include this in your target image, configure it with:

  --enable-addons=wr-idp is required

  --with-template= feature/python

  – Automatically included with **--enable-rootfs= glibc-idp**

Using Python

- Build on your host and include in an image.
  - Start with the default IDP platform project
  - Add the file **setup.py** to the application to manage it by the Python **setuptools** utilities.
  - Set up the build layer for the new package.
    - Define license, add source code and support files, create recipe file
  - Build the directory infrastructure inside the layer and add it to the target file system
    - **`make -C build packagename`**
    - **`make -C build packagename.addpkg`**
    - **`make fs`**

Agenda

Application Stacks

- OpenJDK
- Lua/MQTT
- Python
- **SQLite**
- OSGi
- C

SQLite 3

- SQLite 3 is a terminal based frontend to the SQLite library that can evaluate queries interactively and display the results in multiple formats. It can also be used within scripts.
- SQLite is an embedded relational database engine.
- Its developers call it a self-contained, serverless, zero-configuration, transactional SQL database engine.
- SQLite implements most of the SQL-92 standard for SQL.
- The SQLite engine is statically or dynamically linked into the application, not a standalone process.
- The SQLite library can require less than 300 kB.
- An SQLite database is a single, ordinary disk file that can be located anywhere in the directory hierarchy.

SQLite - Example

Python SQLite Application Example

```
#!/usr/bin/python

import sqlite3 as lite
import sys

con = lite.connect('test.db')

with con:

    cur = con.cursor()
    cur.execute("CREATE TABLE Cars(Id INT, Name TEXT, Price INT)")
    cur.execute("INSERT INTO Cars VALUES(1,'Audi',52642)")
    cur.execute("INSERT INTO Cars VALUES(2,'Mercedes',57127)")
```

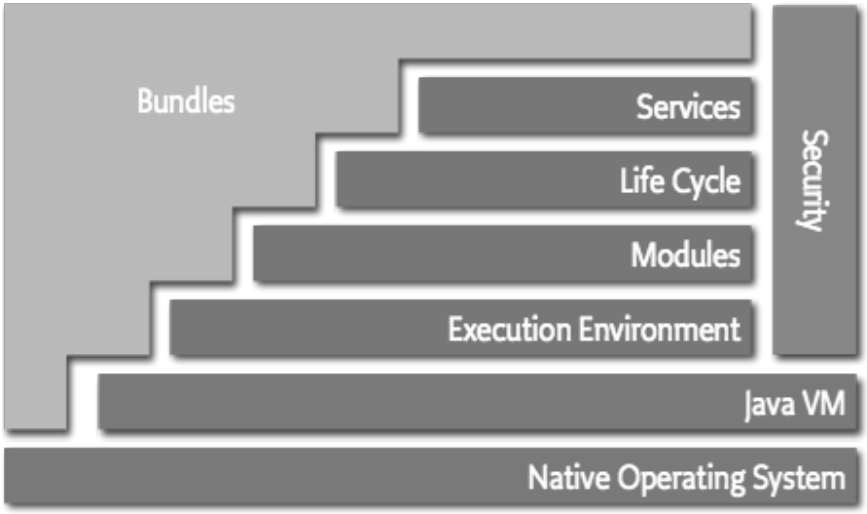
Agenda

Application Stacks

- OpenJDK
- Lua/MQTT
- Python
- SQLite
- **OSGi**
- **C**

OSGi 101

- A set of Java *specifications* that define a dynamic component system and app post-deployment (app store).
- Service provider's own app store enabled
- Vertical market adopted solution



- Enterprise Application Servers (Oracle)
- Mobile Industry (Sprint, Nokia, IBM, …)
- Automotive Industry (BMW, Siemens, Delphi…)
- Telematics (Daimler AG, Bombardier, DB, …)
- Smart Home (Deutsche Telekom, Siemens)

ProSyst mBS OSGi

- The ProSyst mBS Smart Home SDK provides a base from which you can tailor images for specific home device management platforms.
- The OSGi bundle consists of three main components:
  - The OSGi run-time serves as the base for tailored images.
  - Eclipse plug-ins provide facilities for simplified development and testing of OSGi-based projects.
  - The OSGi run-time validator provides an option to validate the components on a specific target platform.

OSGi Development

- Developing on the OSGi platform means first building your application using OSGi APIs, then deploying it in an OSGi container.
- That provides the following advantages:
  - You can install, uninstall, start, and stop different modules of your application dynamically, without restarting the container.
  - An application can have more than one version of a particular module running at a time.
  - OSGi provides very good infrastructure for developing service-oriented applications, as well as embedded, mobile, and rich Internet applications.

ProSyst OSGi Components

- **Toolkit set (Eclipse plug-ins) for development**
  - **mToolkit**: Development environment tools
  - **mBProfiler**: More efficient applications
  - **mBS SH SDK components shared with the run-time**: Specific protocol support for technologies and standards: USB, database services, web services, serial and Parallel communication, UPnP, TLS, OSGi mobile, TEE, and Bluetooth; also included DLNA Server Enabler, email, mGUI, cameras, RMT, config tree, wireless messaging, media players, RSS, notification manager, ZigBee, Z-Wave, X10, KNX, Home Automation Manager, and Home Device Manager
- **ProSyst mBS OSGi Run-Time:** implementation of the OSGi Alliance Specification, ready with a full Smart Home Automation set of prebuilt components
- **ProSyst mBS OSGi Run-Time Validator:** Test and validation tool for OSGi run -time components

ProSyst mBS SmartHome SDK

OSGi (Open Service Gateway Initiative) is the open specifications that enable the modular assembly of software built with Java technology

- Execution Environment: The specification of the Java environment
- Life Cycle: Adds bundles that can be dynamically installed, started, stopped, updated and uninstalled
- Modules: Defines the class loading policies
- Service Registry: Shares objects between bundles
- Bundles: Applications

ProSyst mBS SmartHome SDK:

- Implementation of the "OSGI Service Platform Release 4 Version 4.2“.
- Implements many other more applications based on the OSGi platform:

  ZigBee 2.0.7, DLNA Server Enabler 1.0.3, HAM 2.2.4, eMail 6.3.4, RMT 1.1.19

  Z-Wave 1.2.0, Cameras 2.1.23, Config Tree 1.5.3, RSS 1.0.0, KNX 3.0.0, HDM 4.6.2

  Media Players 2.1.1, Notification Manager 1.0.0, OSGi GWT Ext 1.1.0, X10 3.0.0

  …

Features:

- Optimized the use in commercial embedded products
- Fully integrated with eclipse, including SDK, plug-in, and help
- Integrated web server and web framework for rich web based interfaces

Developing with OSGi

- Step1: Boot Target
- Step2: Start OSGi Java runtime VM

  #cd /opt/prosyst_osgi/mbsa/bin/

  #./mbsa_start

  …

  [mBSA] OSGi framework is started successfully

- Step3: Test OSGi by accessing the OSGi configuration page

  http://targetip/system/console # Login by admin/admin
- Step4: Install Eclipse Plugin
- Step5: Write program from Eclipse plugin
- Step6: Deploy program to Target via Image builder

Agenda

Application Stacks

- OpenJDK
- Lua/MQTT
- Python
- SQLite
- OSGi
- **C**

Developing C application

- Step1: Build a working Platform project by command line or Workbench
- Step2.1: "make export-sdk" to generate SDK cross tool for command line usage
- Step2.2: Use Workbench to import platform project into Workbench
- Step2.3: Write application and build
- Step3: Deploy program to Target

Questions

1. Which Java implementation is part of IDP 2.0?
2. What type of database is SQLite 3?
3. What does MQTT mean?
4. Why would you use OSGi in an IDP target system?

Review

In this chapter you learned to:

- Configure OpenJDK into your target
- Configure MQTT and Lua into your target
- Configure and run Python on your target
- Identify why you would use an SQLite3 database in your target
- Identify the advantages of using OSGi in your target