



Build infrastructure on Intel x86-64

# 1 Contents

2	<b>Build infrastructure on Intel x86-64</b>	<b>2</b>
3	Introduction . . . . .	2
4	Why host the whole build infrastructure on Intel x86-64 . . . . .	2
5	Why OBS workers need a native environment . . . . .	3
6	CPU emulation . . . . .	3
7	Using emulation to target foreign architectures from x86-64 . . . . .	4
8	Mitigating the impact on performance . . . . .	5
9	Risks . . . . .	5
10	Limited maturity of the support for cross-builds in OBS . . . . .	5
11	Versioning mismatches between emulated and injected native	
12	components . . . . .	6
13	Impact of performance loss on timing-depended tests . . . . .	6
14	Emulation bugs . . . . .	6
15	Approach . . . . .	6
16	Evaluation Report . . . . .	7

## 17 Build infrastructure on Intel x86-64

### 18 Introduction

19 The current Apertis infrastructure is largely made of hosts based on the Intel  
20 x86-64 architecture, often using virtualized machines.

21 The only exceptions are:

- 22 • OBS workers used to build packages natively for the ARM 32 bit and  
23 ARM 64 bit architectures
- 24 • LAVA workers, which match the [reference hardware platforms](#)<sup>1</sup>

25 While LAVA workers are by nature meant to be hosted separately from the  
26 rest of the infrastructure and are handled via [geographically distributed LAVA](#)  
27 [dispatchers](#)<sup>2</sup>, the constraint on the OBS workers is problematic for adopters  
28 that want to host downstream Apertis infrastructure.

### 29 Why host the whole build infrastructure on Intel x86-64

30 Being able to host the build infrastructure solely on Intel x86 64 bit (usually  
31 referred to as `x86-64` or `amd64`) machines enables downstream Apertis to be hosted  
32 on a standard public or private cloud solution as these usually only offer x86-64  
33 machines.

34 Deploying the OBS workers on cloud providers would also allow for the imple-  
35 mentation of elastic workload handling.

---

<sup>1</sup>[https://em.pages.apertis.org/apertis-website/reference\\_hardware/](https://em.pages.apertis.org/apertis-website/reference_hardware/)

<sup>2</sup><https://gitlab.apertis.org/infrastructure/apertis-lava-docker/blob/master/apertis-lava-dispatcher/README.md>

36 Elastic scaling, and the desire to ensure that the cloud approach is tested and  
37 viable for downstreams, means that the deployment approach described in this  
38 document is of interest for the main Apertis infrastructure, not just for down-  
39 streams.

40 Some cloud providers like Amazon Web Services have recently started offering  
41 ARM 64 bit servers as well. As a result it should be possible to adopt a hybrid  
42 approach, mixing foreign builds on x86-64 and native ones on ARM machines.

43 In particular, Apertis is currently committed to maintain native workers for all  
44 the supported architectures, but is aiming for a hybrid set up where foreign  
45 packages get built on a mix of native and non-native Intel x86 64 bit machines.

46 Downstreams will be able to opt for fully native, hybrid or Intel-only OBS  
47 worker setups.

## 48 **Why OBS workers need a native environment**

49 Development environments for embedded devices often rely on cross-compilation  
50 to build software targeting a foreign architecture from x86-64 build hosts.

51 However, pure cross-compilation prevents running the unit tests that are shipped  
52 with the projects being built, since the binaries produced do not match that of  
53 the build machine.

54 In addition, supporting cross-compilation across all the projects that compose  
55 a Linux distribution involves a considerable effort, since not all build systems  
56 support cross-compilation, and where it is supported some features may still be  
57 incompatible with it.

58 From the point of view of upstream projects, cross-compilation is in generally a  
59 less tested path, which often leads cross-building distributors to ship a consid-  
60 erable amount of patches adding fixes and workarounds.

61 For this reason all the major package-based distributions like Fedora, Ubuntu,  
62 SUSE and in particular Debian, the upstream distribution from which Apertis  
63 sources most of its packages, choose to only officially support native compilation  
64 for their packages.

65 The Debian infrastructure thus hosts machines with different CPU architectures,  
66 since the build workers must run hardware that matches the architecture of the  
67 binary packages being built.

68 Apertis inherits this requirement, and currently has build workers with Intel 64  
69 bit, ARM 32 and 64 bit CPUs.

## 70 CPU emulation

71 Using the right CPU is fortunately not the only way to execute programs for  
72 non-Intel architectures: the [QEMU project](#)<sup>3</sup> provides the ability to emulate a  
73 multitude of platforms on an x86-64 machine.

74 QEMU offers two main modes:

- 75 • system mode: emulates a full machine, including the CPU and a set of  
76 attached hardware devices;
- 77 • user mode: translates CPU instructions on a running Linux system, run-  
78 ning foreign binaries as if they were native.

79 The system mode is useful when running entire operating systems, but it has a  
80 severe performance impact.

81 The user mode has a much lighter impact on performance as it only deals with  
82 translating the CPU instructions in a Linux executable. For instance, running  
83 an ARMv7 ELF binary on top of the x86-64 kernel running on a x86-64 host.

## 84 Using emulation to target foreign architectures from x86-64

85 The build process on the OBS workers already involves setting up a chroot  
86 where the actual compilation happens. By combining it with the static variant  
87 of the QEMU user mode emulator it can be used to build software on a x86-64  
88 host targeting a foreign architecture as if it were a native build.

89 The `binfmt_misc`<sup>4</sup> subsystem in the kernel can be used to make the emulation  
90 transparent so that emulation happens automatically and transparently when a  
91 foreign binary is executed. Packages can then be built for foreign architectures  
92 without any changes.

93 The emulation-based compilation is also known as [Type 4 cross-build](#)<sup>5</sup> in the  
94 OBS documentation.

95 The following diagram shows how the OBS backend can distribute build jobs to  
96 its workers.

97 Each CPU instruction set is marked by the code name used by OBS:

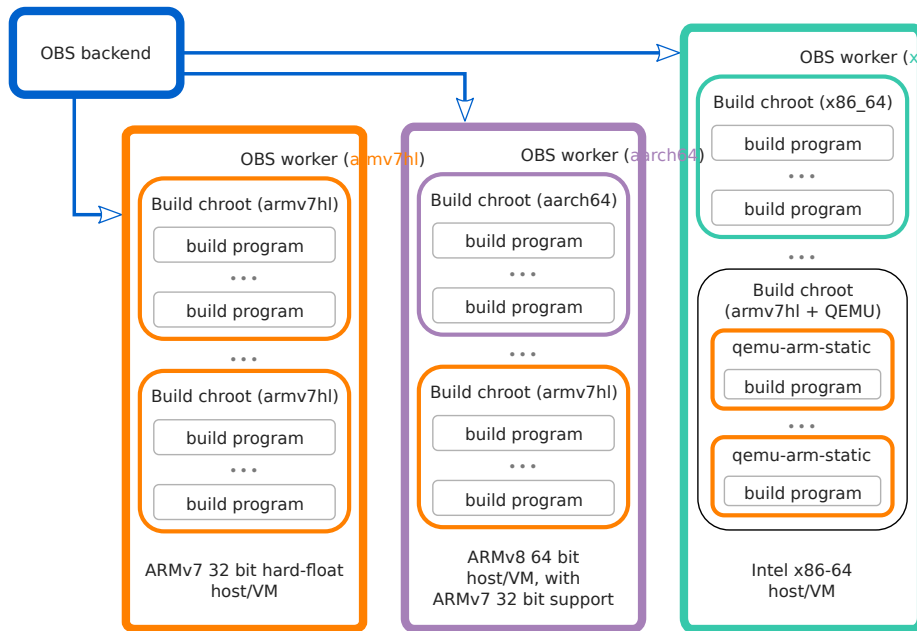
- 98 • `x86_64`: the Intel x86 64 bit ISA, also known as `amd64` in Debian
- 99 • `armv7hl`: the ARMv7 32 bit Hard Float ISA, also known as `armhf` in Debian
- 100 • `aarch64`: the ARMv8 64 bit ISA, also known as `arm64` in Debian

---

<sup>3</sup><https://www.qemu.org/>

<sup>4</sup>[https://en.wikipedia.org/wiki/Binfmt\\_misc](https://en.wikipedia.org/wiki/Binfmt_misc)

<sup>5</sup>[https://en.opensuse.org/openSUSE:Build\\_Service\\_Concept\\_CrossDevelopment#Types\\_of\\_crossbuild](https://en.opensuse.org/openSUSE:Build_Service_Concept_CrossDevelopment#Types_of_crossbuild)



101

102 Particularly relevant here are the `armv7hl` jobs building ARMv7 32 bit packages  
 103 that can be dispatched to:

- 104 1. the native `armv7hl` worker machine;
- 105 2. the `aarch64` worker machine, which supports the ARMv7 32 bit ISA na-
- 106 tively and thus can run binaries in `armv7hl` chroots natively;
- 107 3. the `x86_64` worker machine, which uses the `qemu-arm-static` binary transla-
- 108 tor to run binaries in `armv7hl` chroots via emulation.

109 Some ARM 64 bit server systems do not support the ARMv7 32 bit ISA, and  
 110 would thus require the same emulation-based approach used on the x86-64  
 111 machines to execute the ARM 32 bit jobs.

## 112 Mitigating the impact on performance

113 The most obvious way to handle the performance penalty is to use faster CPUs.  
 114 Cloud providers offer a wide range of options for x86-64 machines, and establis-  
 115 hing the appropriate cost/performance balance is the first step. It is possible that  
 116 the performance of an emulated build on a fast x86-64 CPU may be comparable  
 117 or even faster than a native build on a older ARMv7 machine.

118 In addition, compilation is often a largely parallel task:

- 119 1. big software projects like WebKit are made of many compilation units that  
 120 can be built in parallel
- 121 2. during large scale rebuilds each package can be built in parallel

122 Even if some phases of the build process do not benefit from multiple cores,

123 most of the time is spent on processing the compilation units which means that  
124 increasing the numbers of cores on the worker machines can effectively mitigate  
125 the slowdown due to emulation on large packages.

126 For large scale rebuilds, scaling the number of machines is already helpful, as  
127 the build process for each package is isolated from the others.

128 A different optimization would be to use some selected binaries for the native  
129 architecture during the qemu-linux-user emulation. For instance, a real cross-  
130 compiler can be injected in the build chroot and make it pretend to be the  
131 “native” compiler in the otherwise emulated environment.

132 This would give the best possible performance as the compilation is done with  
133 native `x86-64` code, but care has to be taken to ensure that the cross-compiler  
134 can run reliably in the foreign chroot, and keeping the native and emulated  
135 versions synchronized can be challenging.

## 136 **Risks**

### 137 **Limited maturity of the support for cross-builds in OBS**

138 Support for injecting the QEMU static emulator in the OBS build chroot seems  
139 to be only well tested on RPM-based systems, and there may be some issues  
140 with the DEB-based approach used by Apertis.

141 A feasibility study was done by Collabora in the past demonstrating the viability  
142 of the approach, but some issues may need to be dealt with to deploy it at scale.

### 143 **Versioning mismatches between emulated and injected native com- 144 ponents**

145 If native components are injected in the otherwise emulated cross-build environ-  
146 ment to mitigate the impact on performance, particular care must be made to  
147 ensure that the versions match.

### 148 **Impact of performance loss on timing-depended tests**

149 Some unit tests shipped in upstream packages can be very sensitive to timing  
150 issues, failing on slower machines. If the performance impact is non-trivial, the  
151 emulated environment may be subject to the same failures.

152 However, this is not specific to the emulated environment: Apertis often faces  
153 this kind of issues where some tests that pass on the main Apertis infrastructure  
154 fail due to timing issues on the slower workers that downstream distributions  
155 may use.

156 To mitigate the impact on downstream distributors, the flaky tests usually get  
157 fixed or, if the effort required is too large, disabled.

## 158 **Emulation bugs**

159 The emulator may have bugs that may get triggered by the build process of  
160 some packages.

161 Since upstream distributors use native workers those issues may not be caught  
162 before the triggering package is built on the Apertis infrastructure.

163 Debugging this kind of issues is often not trivial.

## 164 **Approach**

165 These are the high level steps to be undertaken to be able to run the whole  
166 Apertis build infrastructure on x84-64 machines:

- 167 • Set up an OBS test instance with a single `x86-64` worker
- 168 • Configure the test instance and worker for `armhf` and `aarch64` emulated  
169 builds
- 170 • Test a selected set of packages by building them for `armhf` and `aarch64`
- 171 • Set up other `x86-64` workers and test a rebuild of the whole archive, ensur-  
172 ing that all the packages can be build from using the emulated approach
- 173 • Devise mitigations in case some packages fail to build in the emulated  
174 environment
- 175 • Measure and evaluate performance impact comparing build times with  
176 those on the native workers currently in use in Apertis, to decide whether  
177 scaling the number of workers is sufficient to compensate the impact
- 178 • Test mitigation approaches over a selected set of packages and evaluate  
179 the gains
- 180 • Do another rebuild of the whole archive to ensure that the mitigations  
181 didn't introduce regressions
- 182 • Refine and deploy the chosen mitigation approaches to, for instance, en-  
183 sure that the injected native binaries are kept synchronized with the em-  
184 ulated ones they replace

185 There's a risk that no mitigation end up being effective on some packages so  
186 they keep failing in the emulated approach. In the short term those packages  
187 will be required to be built on the native workers in a hybrid set up, but they  
188 would be more problematic in a hypothetical downstream setup with no native  
189 workers as they can't be built there. In that case, pre-built binaries coming  
190 from an upstream with native workers will have to be injected in the archive.

191 Alternatively, it may be possible to mix [type 3 and 4 crossbuilds](#)<sup>6</sup> by modifying  
192 the failing packages to make them buildable with a real cross-compiler. This  
193 solution requires a much higher maintenance cost as packages do not generally  
194 support being built in that way, but it may be an option to be able to do full  
195 builds on x86-64 in the few cases where emulation fails.

---

<sup>6</sup>[https://en.opensuse.org/openSUSE:Build\\_Service\\_Concept\\_CrossDevelopment#Types\\_of\\_crossbuild](https://en.opensuse.org/openSUSE:Build_Service_Concept_CrossDevelopment#Types_of_crossbuild)

## 196 Evaluation Report

197 A full archive-wide build was run on the Azure Cloud setup, using `x86-64` virtual  
198 machines. A cloud optimized setup was built, comprising of the following major  
199 components:

- 200 • Azure provided Linux Virtual Machines (Debian Buster)
- 201 • Docker (as provided by the Linux distribution vendor)
- 202 • Linux 4.19 and above
- 203 • binfmt-support
- 204 • QEMU Emulator

205 Given the task at hand, to run emulation for `ARM` architecture on `x86-64`, we  
206 chose the following cloud hardware class for our OBS worker setup.

- 207 • OBS-Server VM: Standard DS14 v2 (16 vcpus, 112 GiB memory)
- 208 • Worker VM: Standard F32s\_v2 (32 vcpus, 64 GiB memory)

209 The provisioned `OBS-Server` VM hosted all of the OBS services, dockerized to run  
210 easily and efficiently in a cloud environment. For the workers, we provisioned  
211 3 `Worker` VMs, each VM running 5 worker instances per architecture, with 3  
212 architectures this resulted in a total of 15 worker instances per virtual machine.  
213 In total, we ran 45 worker instances for our build farm. This includes 30 worker  
214 instances doing emulated builds, 15 for the 32-bit ARM architecture and 15 for  
215 the 64 bit architecture. The remaining 15 worker instances were allocated for  
216 native `x86` builds.

217 All services used Azure provided *Premium SSD* disk storage. Azure Networking  
218 was tweaked to allow full intercommunication in-between the VMs.

219 The OBS Build setup was populated with the Apertis v2021dev3 release for the  
220 `development`, `target` and `sdk` components. The combined number of packages for  
221 the 3 repository components is: 4121

- 222 • `development` => 3237 packages
- 223 • `target` => 465 packages
- 224 • `sdk` => 419 packages

225 Of the mentioned repositories, `development` and `target` repository are built for 3  
226 architectures: `x86-64`, `armv7hl` and `aarch64`, while `sdk` repository is built only for  
227 the `x86-64` architecture.

228 The full archive-wide rebuild of Apertis v2021dev3 was completed in around 1  
229 week, with the above mentioned setup. There weren't any build failure specific  
230 to the setup above, to the `emulated build` setup in particular. Some packages  
231 failed to build while running their respective build time tests.

232 To summarize, *Emulated Builds* worked fine with 2 caveats mentioned below

- 233 • Performance: Given the emulation penalty, builds were 4-5 times slower  
234 than native.



235  
236

- Failing packages: Given the performance penalty due to emulation, some of the tests failed due to timeouts