



Image building infrastructure (obsolete)

1	<b>Contents</b>	
2	<b>Introduction</b>	<b>2</b>
3	<b>Technology overview</b>	<b>2</b>
4	<b>Jenkins master setup</b>	<b>2</b>
5	<b>Jenkins slave setup</b>	<b>3</b>
6	<b>Docker registry setup</b>	<b>3</b>
7	<b>Docker images for the build environment</b>	<b>3</b>
8	<b>Image building process</b>	<b>4</b>
9	<b>Jenkins jobs instantiation</b>	<b>4</b>
10	<b>OSTree support (server side)</b>	<b>4</b>
11	<b>Appendix: List of plugins installed on the Jenkins master</b>	<b>4</b>

12 This document provides an overview of the image build pipeline prior to the  
13 migration to GitLab CI/CD that has been completed during the v2021 devel-  
14 opment cycle. Refer to the documentation in the [infrastructure/apertis-image-  
15 recipes<sup>1</sup>](#) project for information about the current pipeline.

## 16 Introduction

17 The Apertis infrastructure supports continuous building of reference images,  
18 hwpacks and ospacks. This document explains the infrastructure setup, config-  
19 uration and concepts.

## 20 Technology overview

21 To build the various packs (hardware, os) as well as images, Apertis uses [Debos<sup>2</sup>](#),  
22 a flexible tool to configure the build of Debian-based operating systems. Debos  
23 uses tools like `debootstrap` already present in the environment and relies on  
24 virtualisation to securely do privileged operations without requiring root access.

25 For orchestrating Apertis uses the well-known [Jenkins<sup>3</sup>](#) automation server. Fol-  
26 lowing current best practices the Apertis image build jobs use Jenkins pipelines  
27 (introduced in Jenkins 2.0) to drive the build process as well as doing the actual

---

<sup>1</sup><https://gitlab.apertis.org/infrastructure/apertis-image-recipes/>

<sup>2</sup><https://github.com/go-debos/debos>

<sup>3</sup><https://jenkins.io>

28 build inside [Docker images](#)<sup>4</sup> to allow for complete control of the job specific  
29 build-environment without relying on job-specific Jenkins slave configuration.  
30 As an extra bonus the Docker images used by Jenkins can be re-used by devel-  
31 opers for local testing in the same environment.

32 For each Apertis release there are two relevant Jenkins jobs to build images;  
33 The first job builds a Docker image which defines the build environment and  
34 uploads the resulting image to the Apertis Docker registry. This is defined in  
35 the [apertis-docker-images git repository](#)<sup>5</sup>. The second job defines the build steps  
36 for the various ospacks, hardware packs and images which are run in the Docker  
37 image build by the previous job; it also uploads the results to images.apertis.org.

## 38 Jenkins master setup

39 Instructions to install Jenkins can be can be found on the [Jenkins download](#)  
40 [page](#)<sup>6</sup>. Using the Long-Term support version of Jenkins is recommended. For  
41 the Apertis infrastructure Jenkins master is being run on Debian 9.3 (stretch).

42 The plugins that are installed on the master can be found in the [plugins ap-  
43 pendix][Appendix: List of plugins installed on the Jenkins master]

## 44 Jenkins slave setup

45 Each Jenkins slave should be installed on a separate machine (or VM) in line  
46 with the Jenkins best practices. As the image build environment is contained  
47 in a Docker image, the Jenkins slave requires only a few tools to be installed.  
48 Apart from running a Jenkins slave itself, the following requirements must be  
49 satisfied on slave machines:

- 50 • git client installed on the slave
- 51 • Docker installed on the slave and usable by the Jenkins slave user
- 52 • /dev/kvm accessible by the Jenkins slave user (for hw acceleration support  
53 in the image builder)

54 For the last requirement on Debian systems this can be achieved by dropping  
55 a file called `/etc/udev/rules.d/99-kvm-perms.rules` in place with the following  
56 content.

```
57 SUBSYSTEM=="misc", KERNEL=="kvm", GROUP="kvm", MODE="0666"
```

58 Documentation for installing Docker on Debian can be found as part of the  
59 [Docker documentation](#)<sup>7</sup>. To allow Docker to be usable by Jenkins, the Jenkins  
60 slave user should be configured as part of the `docker` group.

---

<sup>4</sup><https://jenkins.io/doc/book/pipeline/docker/>

<sup>5</sup><https://gitlab.apertis.org/infrastructure/apertis-docker-images>

<sup>6</sup><https://jenkins.io/download/>

<sup>7</sup><https://docs.docker.com/install/linux/docker-ce/debian/>

61 Documentation on how to setup Jenkins slaves can be found as part of the  
62 [Jenkins documentation](#)<sup>8</sup>.

## 63 Docker registry setup

64 To avoid building Docker images for every image build round and to make it  
65 easier for Jenkins and developers to share the same Docker environment for  
66 build testing, it is recommended to run a Docker registry. The [Docker registry  
67 documentation](#)<sup>9</sup> contains information on how to setup a registry.

## 68 Docker images for the build environment

69 The Docker images defining the environment for building the images can be  
70 found in the [apertis-docker-images git repository](#)<sup>10</sup>.

71 The toplevel Jenkinsfile is setup to build a Docker image based on the [Dock-  
72 erfile](#)<sup>11</sup> defined in the Apertis-image-builder directory and upload the result  
73 to the public Apertis Docker registry `docker-registry.apertis.org` through the  
74 authenticated upload channel `auth.docker-registry.apertis.org`.

75 For Apertis derivatives this file should be adjusted to upload the Docker image  
76 to the Docker registry of the derivative.

## 77 Image building process

78 The image recipes and configuration can be found in the [apertis-image-recipes  
79 git repository](#)<sup>12</sup>. As with the Docker images, the top-level `Jenkinsfile` defines  
80 the Jenkins job. For each image type to be built a parallel job is started which  
81 runs the image-building toolchain in the Docker-defined environment.

82 The various recipes provide the configuration for debos, documentation about  
83 the available actions can be found in the [Debos documentation](#)<sup>13</sup>.

## 84 Jenkins jobs instantiation

85 Jenkins needs to be pointed to the repositories hosting the Jenkinsfiles by cre-  
86 ating matching jobs on the master instance. This can be done either manually  
87 from the web UI or using the YAML templates supported by the `jenkins-jobs`

---

<sup>8</sup><https://wiki.jenkins.io/display/JENKINS/Distributed+builds>

<sup>9</sup><https://docs.docker.com/registry/deploying/>

<sup>10</sup><https://gitlab.apertis.org/infrastructure/apertis-docker-images>

<sup>11</sup><https://docs.docker.com/engine/reference/builder/>

<sup>12</sup><https://gitlab.apertis.org/infrastructure/apertis-image-recipes>

<sup>13</sup><https://godoc.org/github.com/go-debos/debos/actions>

88 command-line tool from the `jenkins-job-builder` package, version 2.0 or later  
89 for the support of pipeline jobs.

90 For that purpose Apertis uses a set of job templates hosted in the `apertis-`  
91 `jenkins-jobs`<sup>14</sup> repository.

## 92 OSTree support (server side)

93 The image build jobs prepare OSTree repository to be installed server side.  
94 In order to properly support OSTree server side, `ostree-push` package must be  
95 installed in the OSTree repository server.

## 96 Appendix: List of plugins installed on the Jenkins 97 master

98 At the time of this writing the following plugins are installed on the Apertis  
99 Jenkins master:

- 100 • `ace-editor`
- 101 • `analysis-model-api`
- 102 • `ant`
- 103 • `antisamy-markup-formatter`
- 104 • `apache-httpcomponents-client-4-api`
- 105 • `artifactdeployer`
- 106 • `authentication-tokens`
- 107 • `blueocean`
- 108 • `blueocean-autofavorite`
- 109 • `blueocean-bitbucket-pipeline`
- 110 • `blueocean-commons`
- 111 • `blueocean-config`
- 112 • `blueocean-core-js`
- 113 • `blueocean-dashboard`
- 114 • `blueocean-display-url`
- 115 • `blueocean-events`
- 116 • `blueocean-executor-info`
- 117 • `blueocean-git-pipeline`
- 118 • `blueocean-github-pipeline`
- 119 • `blueocean-i18n`
- 120 • `blueocean-jira`
- 121 • `blueocean-jwt`
- 122 • `blueocean-personalization`
- 123 • `blueocean-pipeline-api-impl`
- 124 • `blueocean-pipeline-editor`

---

<sup>14</sup><https://gitlab.apertis.org/infrastructure/apertis-jenkins-jobs>

- 125 • blueocean-pipeline-scm-api
- 126 • blueocean-rest
- 127 • blueocean-rest-impl
- 128 • blueocean-web
- 129 • bouncycastle-api
- 130 • branch-api
- 131 • build-flow-plugin
- 132 • build-name-setter
- 133 • build-token-root
- 134 • buildgraph-view
- 135 • cloudbees-bitbucket-branch-source
- 136 • cloudbees-folder
- 137 • cobertura
- 138 • code-coverage-api
- 139 • command-launcher
- 140 • conditional-buildstep
- 141 • copyartifact
- 142 • credentials
- 143 • credentials-binding
- 144 • cvs
- 145 • display-url-api
- 146 • docker-commons
- 147 • docker-custom-build-environment
- 148 • docker-workflow
- 149 • durable-task
- 150 • email-ext
- 151 • embeddable-build-status
- 152 • eninject
- 153 • eninject-api
- 154 • external-monitor-job
- 155 • favorite
- 156 • forensics-api
- 157 • git
- 158 • git-client
- 159 • git-server
- 160 • git-tag-message
- 161 • github
- 162 • github-api
- 163 • github-branch-source
- 164 • github-organization-folder
- 165 • gitlab-plugin
- 166 • handlebars
- 167 • handy-uri-templates-2-api
- 168 • htmlpublisher
- 169 • hudson-pview-plugin
- 170 • icon-shim

- 171 • jackson2-api
- 172 • javadoc
- 173 • jdk-tool
- 174 • jenkins-design-language
- 175 • jira
- 176 • jquery
- 177 • jquery-detached
- 178 • jsch
- 179 • junit
- 180 • ldap
- 181 • lockable-resources
- 182 • mailer
- 183 • mapdb-api
- 184 • matrix-auth
- 185 • matrix-project
- 186 • mattermost
- 187 • maven-plugin
- 188 • mercurial
- 189 • metrics
- 190 • modernstatus
- 191 • momentjs
- 192 • multiple-scms
- 193 • pam-auth
- 194 • parameterized-trigger
- 195 • phabricator-plugin
- 196 • pipeline-build-step
- 197 • pipeline-github-lib
- 198 • pipeline-graph-analysis
- 199 • pipeline-input-step
- 200 • pipeline-milestone-step
- 201 • pipeline-model-api
- 202 • pipeline-model-declarative-agent
- 203 • pipeline-model-definition
- 204 • pipeline-model-extensions
- 205 • pipeline-rest-api
- 206 • pipeline-stage-step
- 207 • pipeline-stage-tags-metadata
- 208 • pipeline-stage-view
- 209 • plain-credentials
- 210 • pollscm
- 211 • promoted-builds
- 212 • publish-over
- 213 • publish-over-ssh
- 214 • pubsub-light
- 215 • repo
- 216 • resource-disposer

- 217 • run-condition
- 218 • scm-api
- 219 • scoring-load-balancer
- 220 • script-security
- 221 • sse-gateway
- 222 • ssh-agent
- 223 • ssh-credentials
- 224 • ssh-slaves
- 225 • structs
- 226 • subversion
- 227 • timestamper
- 228 • token-macro
- 229 • translation
- 230 • trilead-api
- 231 • variant
- 232 • versionnumber
- 233 • view-job-filters
- 234 • warnings-ng
- 235 • windows-slaves
- 236 • workflow-aggregator
- 237 • workflow-api
- 238 • workflow-basic-steps
- 239 • workflow-cps
- 240 • workflow-cps-global-lib
- 241 • workflow-durable-task-step
- 242 • workflow-job
- 243 • workflow-multibranch
- 244 • workflow-scm-step
- 245 • workflow-step-api
- 246 • workflow-support
- 247 • ws-cleanup

248 To retrieve the list, access the script console and enter the following Groovy  
249 script:

```
250 Jenkins.instance.pluginManager.plugins.toList()
251     .sort{plugin -> plugin.getShortName()}
252     .each{plugin -> println ("* ${plugin.getShortName()}")}
```