Image building infrastructure (obsolete)

Contents

Introduction **2**

Technology overview **2**

Jenkins master setup **2**

Jenkins slave setup **3**

Docker registry setup **3**

Docker images for the build environment **3**

Image building process **4**

Jenkins jobs instantiation **4**

OSTree support (server side) **4**

Appendix: List of plugins installed on the Jenkins master **4**

This document provides and overview of the image build pipeline prior to the migration to GitLab CI/CD that has been completed during the v2021 development cycle. Refer to the documentation in the infrastructure/apertis-image-recipes[1] project for information about the current pipeline.

Introduction

The Apertis infrastructure supports continuous building of reference images, hwpacks and ospacks. This document explains the infrastructure setup, configuration and concepts.

Technology overview

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

For orchestrating Apertis uses the well-known Jenkins[3] automation server. Following current best practices the Apertis image build jobs use Jenkins pipelines (introduced in Jenkins 2.0) to drive the build process as well as doing the actual

[1] https://gitlab.apertis.org/infrastructure/apertis-image-recipes/
[2] https://github.com/go-debos/debos
[3] https://jenkins.io

build inside Docker images[4] to allow for complete control of the job specific build-environment without relying on job-specific Jenkins slave configuration. As an extra bonus the Docker images used by Jenkins can be re-used by developers for local testing in the same environment.

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

Jenkins master setup

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

The plugins that are installed on the master can be found in the [plugins appendix][Appendix: List of plugins installed on the Jenkins master]

Jenkins slave setup

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

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

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

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

Documentation for installing Docker on Debian can be found as part of the

[4] https://jenkins.io/doc/book/pipeline/docker/
[5] https://gitlab.apertis.org/infrastructure/apertis-docker-images
[6] https://jenkins.io/download/

Docker documentation[7]. To allow Docker to be usable by Jenkins, the Jenkins slave user should be configured as part of the `docker` group.

Documentation on how to setup Jenkins slaves can be found as part of the Jenkins documentation[8].

Docker registry setup

To avoid building Docker images for every image build round and to make it easier for Jenkins and developers to share the same Docker environment for build testing, it is recommended to run a Docker registry. The Docker registry documentation[9] contains information on how to setup a registry.

Docker images for the build environment

The Docker images defining the environment for building the images can be found in the apertis-docker-images git repository[10].

The toplevel Jenkinsfile is setup to build a Docker image based on the Dockerfile[11] defined in the Apertis-image-builder directory and upload the result to the public Apertis Docker registry `docker-registry.apertis.org` through the authenticated upload channel `auth.docker-registry.apertis.org`.

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

Image building process

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

The various recipes provide the configuration for debos, documentation about the available actions can be found in the Debos documentation[13].

[7] https://docs.docker.com/install/linux/docker-ce/debian/
[8] https://wiki.jenkins.io/display/JENKINS/Distributed+builds
[9] https://docs.docker.com/registry/deploying/
[10] https://gitlab.apertis.org/infrastructure/apertis-docker-images
[11] https://docs.docker.com/engine/reference/builder/
[12] https://gitlab.apertis.org/infrastructure/apertis-image-recipes
[13] https://godoc.org/github.com/go-debos/debos/actions

Jenkins jobs instantiation

Jenkins needs to be pointed to the repositories hosting the Jenkinsfiles by creating matching jobs on the master instance. This can be done either manually from the web UI or using the YAML templates supported by the `jenkins-jobs` command-line tool from the `jenkins-job-builder` package, version 2.0 or later for the support of pipeline jobs.

For that purpose Apertis uses a set of job templates hosted in the `apertis-jenkins-jobs`[14] repository.

OSTree support (server side)

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

Appendix: List of plugins installed on the Jenkins master

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

- ace-editor
- analysis-model-api
- ant
- antisamy-markup-formatter
- apache-httpcomponents-client-4-api
- artifactdeployer
- authentication-tokens
- blueocean
- blueocean-autofavorite
- blueocean-bitbucket-pipeline
- blueocean-commons
- blueocean-config
- blueocean-core-js
- blueocean-dashboard
- blueocean-display-url
- blueocean-events
- blueocean-executor-info
- blueocean-git-pipeline
- blueocean-github-pipeline
- blueocean-i18n
- blueocean-jira

[14] https://gitlab.apertis.org/infrastructure/apertis-jenkins-jobs

- blueocean-jwt
- blueocean-personalization
- blueocean-pipeline-api-impl
- blueocean-pipeline-editor
- blueocean-pipeline-scm-api
- blueocean-rest
- blueocean-rest-impl
- blueocean-web
- bouncycastle-api
- branch-api
- build-flow-plugin
- build-name-setter
- build-token-root
- buildgraph-view
- cloudbees-bitbucket-branch-source
- cloudbees-folder
- cobertura
- code-coverage-api
- command-launcher
- conditional-buildstep
- copyartifact
- credentials
- credentials-binding
- cvs
- display-url-api
- docker-commons
- docker-custom-build-environment
- docker-workflow
- durable-task
- email-ext
- embeddable-build-status
- envinject
- envinject-api
- external-monitor-job
- favorite
- forensics-api
- git
- git-client
- git-server
- git-tag-message
- github
- github-api
- github-branch-source
- github-organization-folder
- gitlab-plugin
- handlebars

- handy-uri-templates-2-api
- htmlpublisher
- hudson-pview-plugin
- icon-shim
- jackson2-api
- javadoc
- jdk-tool
- jenkins-design-language
- jira
- jquery
- jquery-detached
- jsch
- junit
- ldap
- lockable-resources
- mailer
- mapdb-api
- matrix-auth
- matrix-project
- mattermost
- maven-plugin
- mercurial
- metrics
- modernstatus
- momentjs
- multiple-scms
- pam-auth
- parameterized-trigger
- phabricator-plugin
- pipeline-build-step
- pipeline-github-lib
- pipeline-graph-analysis
- pipeline-input-step
- pipeline-milestone-step
- pipeline-model-api
- pipeline-model-declarative-agent
- pipeline-model-definition
- pipeline-model-extensions
- pipeline-rest-api
- pipeline-stage-step
- pipeline-stage-tags-metadata
- pipeline-stage-view
- plain-credentials
- pollscm
- promoted-builds
- publish-over

- publish-over-ssh
- pubsub-light
- repo
- resource-disposer
- run-condition
- scm-api
- scoring-load-balancer
- script-security
- sse-gateway
- ssh-agent
- ssh-credentials
- ssh-slaves
- structs
- subversion
- timestamper
- token-macro
- translation
- trilead-api
- variant
- versionnumber
- view-job-filters
- warnings-ng
- windows-slaves
- workflow-aggregator
- workflow-api
- workflow-basic-steps
- workflow-cps
- workflow-cps-global-lib
- workflow-durable-task-step
- workflow-job
- workflow-multibranch
- workflow-scm-step
- workflow-step-api
- workflow-support
- ws-cleanup

To retrieve the list, access the script console[15] and enter the following Groovy script:

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

[15] https://jenkins.apertis.org/script