



Infrastructure maintenance automation

# 1 Contents

2	<b>Goals</b>	<b>2</b>
3	Data-driven . . . . .	2
4	Git-controlled . . . . .	2
5	Idempotent . . . . .	3
6	Scalable . . . . .	3
7	Single source of truth . . . . .	3
8	Reproducible . . . . .	3
9	Explicit . . . . .	3
10	<b>Basic approach</b>	<b>3</b>
11	Add test mode for current branching scripts . . . . .	3
12	Improve coverage of current branching scripts . . . . .	4
13	<b>Longer term approach</b>	<b>4</b>
14	Centralized metadata . . . . .	4
15	Per-repository branching operations . . . . .	5
16	<b>Implementation</b>	<b>5</b>
17	Add test mode for current branching scripts . . . . .	5
18	Improve coverage of current branching scripts . . . . .	6
19	Centralized metadata . . . . .	6
20	Per-repository branching operations . . . . .	8

21 This document describes the goals and the approaches for automating the man-  
22 agement of the infrastructure used by Apertis. It will focus in particular on  
23 release branching since the new [release flow](#)<sup>1</sup> implies that Apertis will need  
24 to go through that process two or three times more than in the past on each  
25 quarter.

## 26 Goals

### 27 Data-driven

28 Separating the description of the desired infrastructure state from the tools to  
29 apply it nicely separates the two concerns: in most cases the tools won't need  
30 to be updated during branching, only the desired infrastructure state changes.

### 31 Git-controlled

32 Basing everything on configuration stored in a Git repository has several advan-  
33 tages:

- 34 • all the changes are tracked over time

---

<sup>1</sup><https://www.apertis.org/policies/release-flow/#apertis-release-flow>

- 35 • the standard Apertis workflows based on GitLab merge requests can be  
36 used to review changes
- 37 • fine access controls can be configured via GitLab

## 38 **Idempotent**

39 Every tool should compare the current state with the desired one and not pro-  
40 duce errors when they already match. Administrators should be able to run the  
41 tools at any time, multiple times, without any ill effect.

## 42 **Scalable**

43 The Apertis infrastructure is composed of enough services that a centralized list  
44 of things to update when branching is doomed to be outdated every quarter.

## 45 **Single source of truth**

46 The duplication of the same information between modules should be minimized,  
47 such that updating the single source of truth automatically produces effects on  
48 the depending modules.

## 49 **Reproducible**

50 Running the tools in a standardized, easily reproducible environment enables  
51 all the administrators to easily deploy changes without any special setup.

## 52 **Explicit**

53 All the needed information should be explicitly encoded in metadata repository.  
54 The tools using it should strive to not make any assumption on the data and  
55 derive more information out of it. This is another facet of ensuring that the  
56 metadata repository remains the single source of truth.

## 57 **Basic approach**

58 The basic approach aims at improving the current branching scripts to make  
59 them easier to test by developers, enabling more people to work on them, and  
60 to extend them to fully handle the complete branching process.

## 61 **Add test mode for current branching scripts**

62 At the moment the quarterly release branching is done through a *set of scripts*<sup>2</sup>  
63 that get invoked manually by one the Apertis infrastructure team member from  
64 their machine.

---

<sup>2</sup><https://gitlab.apertis.org/infrastructure/apertis-infrastructure/-/tree/main/release-scripts>

65 They act directly on the live services using the caller's accounts.

66 The first step for improving the branching automation is to add a “dry-run”  
67 mode to all the current release scripts to let developers and admin run them

## 68 **Improve coverage of current branching scripts**

69 The scripts currently in charge of reducing the manual intervention during the  
70 branching process do not cover all services and repositories which are part of  
71 Apertis.

72 Once the “dry-run”mode is in place, new steps need to be added to the branching  
73 scripts to cover the missing services and repositories.

## 74 **Longer term approach**

75 Larger refactorings are needed to align the current infrastructure to the goals  
76 previously described.

77 The following sections describe the steps needed to further improve the infras-  
78 tructure maintenance to make it more robust and require less effort to manage.

## 79 **Centralized metadata**

80 A new git repository contains the principal metadata about the whole Apertis  
81 infrastructure describing:

- 82 • the currently active release branches
  - 83 – e.g. v2020pre, v2019, etc.
- 84 • their phase in the release lifecycle
  - 85 – e.g. development, preview, stable
- 86 • their release status
  - 87 – e.g. frozen, release-candidate, released
- 88 • the release from which they get branched from:
  - 89 – e.g. 2019pre for both v2019 and v2020dev0
- 90 • the matching git branch name
  - 91 – e.g. apertis/v2019
- 92 • the APT components they ship
  - 93 – e.g. target, development, sdk, hmi
- 94 • etc.

95 This provides a git-controlled single source of truth: tools are updated to fetch  
96 the information they need from this repository.

97 For instance, the creation of OBS projects should be handled by a tool that:

- 98 • fetches the above YAML
- 99 • checks the current OBS configuration
- 100 • computes the changes needed compared to the desired state, if any

- 101 • applies the changes, if any, to align the actual state to the desired state,  
102 providing an idempotent solution
- 103 • runs from a GitLab pipeline, providing a reproducible environment that  
104 can be either triggered by changes in the main data repository or manually

105 The current infrastructure encodes a lot of information about the releases in  
106 several places: tools should be changed to fetch such information on the fly from  
107 the main data repository or GitLab pipelines should be configured to monitor  
108 the main data repository and automatically apply changes accordingly.

109 For instance, the LAVA job templates encode the branch name of the release  
110 they track in multiple places. Either the templates can be enhanced to fetch  
111 the information on the fly from the main data repository, or a pipeline should  
112 be configured in a dedicated branch in the repository to monitor the main data  
113 repository and branch/update the repository accordingly.

114 The change compared to the current approach is to minimize the amount of  
115 information that needs to be branched and distribute the branching logic closer  
116 to the entity to be branched. This is meant to avoid the recurring issues where  
117 the current centralized branching scripts failed to branch things properly or did  
118 not include new components to be branched at all.

## 119 **Per-repository branching operations**

120 For most repositories it is sufficient to add a new git ref when branching for a  
121 new release. In particular, nearly all the the packaging ones do not need any  
122 change to the repository contents and creating a new ref is enough.

123 Other repositories need instead some changes to be applied to the contents once  
124 a new release branch is created. A common reason is that the release name is  
125 encoded in some file, which means that the file needs to be updated and the  
126 change needs to be committed and pushed.

127 By making branching self-contained in the repositories, moving and renaming  
128 them no longer cause breakage. It also gives full control over the branching  
129 of a repository to the people managing that repository, rather than those who  
130 manage the centralized repository. This can be especially useful for components  
131 not managed by the core Apertis team, owned instead by product-specific teams.

132 In general, keeping the branching operation in the same place as the rest of the  
133 contents helps in keeping them coherent and makes testing easier.

## 134 **Implementation**

### 135 **Add test mode for current branching scripts**

136 Setting the `NOACT=y` environment variable causes the branching scripts to run in  
137 test mode, without actually launching the branching commands.

138 **Improve coverage of current branching scripts**

139 New actions need to be taken when branching a new release.

140 This is a non exhaustive list:

- 141 • branch LAVA job templates;
- 142 • update the configuration on GitLab repositories to create the new release  
143 branch, make it the default, etc.;
- 144 • create the relevant `:snapshots` repositories on OBS;
- 145 • add support for the `security`, `updates` and `backports` repositories when  
146 branching stable releases.

147 **Centralized metadata**

148 The centralized information can be modeled as YAML, for instance:

---

```
1 .common_components: &common_components
2   - target
3   - development
4   - sdk
5   - hmi
6
7 projects:
8   apertis:
9     releases:
10      v2019:
11        lifecycle: stable
12        status: released
13        branched-from: v2019pre
14        branch-name: apertis/v2019
15        upstream: debian/buster
16        obs-build-suffix: v2019.0
17        suites:
18          v2019:
19            obs-pattern: '$project:$release:$component'
20            components: *common_components
21          v2019-updates:
22            obs-pattern: '$project:$release:updates:$component'
23            components: *common_components
24          v2019-security:
25            obs-pattern: '$project:$release:security:$component'
26            components: *common_components
27        infrastructure-packages:
28          obs: apertis:infrastructure:v2019
29          suite: infrastructure-v2019
30          components:
31            - buster
32      v2020dev0:
33        lifecycle: development
34        status: frozen
35        branched-from: v2019pre
36        branch-name: apertis/v2020dev0
37        upstream: debian/buster
38        obs-build-suffix: v2020dev0
39        suites:
40          v2020dev0:
41            obs-pattern: '$project:$release:$component'
42            components: *common_components
43        infrastructure-packages:
44          obs: apertis:infrastructure:v2019
45          suite: infrastructure-v2019
46          components:
47            - buster
48
```

---

---

149 **Per-repository branching operations**

150 A `release-branching` step should be added to the GitLab CI pipeline YAML in  
151 the repository with the purpose of ensuring that the release-specific contents  
152 match the branch name.

153 GitLab does not provide any way to execute an action only when a new ref  
154 is created so the best strategy is to ensure that the `release-branching` script is  
155 idempotent and gets run when changes land to any `apertis/*` refs: if no changes  
156 are detected the step succeeds with no further operations, otherwise it commit  
157 and push the changes automatically, or it submits a MR to be reviewed before  
158 landing the changes.